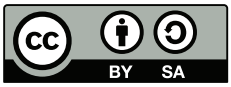
FEMoctave, Finite Element Algorithms in Octave

Andreas Stahel, Bern University of Applied Sciences

Version 2.1.11, created on October 14, 2026



©Andreas Stahel, 2026
"FEMoctave" by Andreas Stahel, BFH, Biel, Switzerland is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.
You are free: to copy, distribute, transmit the work, to adapt the work and to make commercial use of the work. Under the following conditions: You must attribute the work to the original author (but not in any way that suggests that the author endorses you or your use of the work). Attribute this work as follows: Andreas Stahel: FEMoctave, FEM algorithms in Octave.
If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

Table of Contents **1**

1 Introduction **8**

2 The Problems to be Examined by `FEMoctave` **10**
2.1 Selection trees for the FEMoctave solvers . . . 10
2.2 The domain $\Omega \subset \mathbb{R}^2$ and its boundary $\Gamma = \partial\Omega = \Gamma_1 \cup \Gamma_2$ . . . 13
2.3 The general elliptic problem . . . 13
2.4 The symmetric elliptic problem . . . 14
2.5 The symmetric eigenvalue problem . . . 15
2.6 The nonlinear elliptic problem . . . 15
2.7 The general parabolic problem . . . 15
2.8 The symmetric parabolic problem . . . 15
2.9 The semilinear parabolic problem . . . 16
2.10 The hyperbolic problem . . . 16
2.11 The semilinear hyperbolic problem . . . 16
2.12 1D boundary value problems . . . 16
2.13 1D initial boundary value problems of order 1 . . . 16
2.14 1D initial boundary value problems of order 2 . . . 17
2.15 1D eigenvalue value problems . . . 17
2.16 Nonlinear 1D boundary value problems . . . 17
2.17 Dynamic semilinear 1D initial boundary value problems . . . 17
2.18 Dynamic semilinear 1D wave equations . . . 17
2.19 Plane elasticity . . . 17
2.19.1 Description of strain . . . 18
2.19.2 Description of stress and Hooke's law . . . 18
2.19.3 Thermal expansion . . . 20
2.19.4 Plane stress problems . . . 21
2.19.5 Plane strain problems . . . 23
2.20 Elasticity problems for axisymmetric solids, using cylindrical coordinates . . . 24

3 Illustrative Examples **26**
3.1 Solving elliptic problems, static heat equations . . . 26
3.1.1 A symmetric problem . . . 26
3.1.2 Laplace equation in cylindrical coordinates . . . 27
3.1.3 Diffusion on an L-shaped domain . . . 28
3.1.4 A diffusion convection problem . . . 28
3.1.5 Solving nonisotropic problems . . . 29
3.1.6 Solving a nonlinear problem . . . 31
3.2 Solving eigenvalue problems . . . 32
3.3 Solving parabolic problems, dynamic heat equations . . . 35
3.4 Solving semilinear parabolic problems, dynamic heat equations . . . 36
3.5 Solving hyperbolic problems, wave equations . . . 36
3.6 Solving semilinear hyperbolic problems, wave equations . . . 38
3.7 Solving 1D steady state boundary value problems . . . 40
3.8 Solving 1D dynamic initial boundary value problems of order 1, a heat equation . . . 42
3.9 Solving 1D dynamic initial boundary value problems of order 2, a wave equation . . . 42
3.10 Solving nonlinear 1D boundary value problems . . . 43
3.10.1 A nonlinear 1D BVP solved by `BVP1DNL()` . . . 43
3.10.2 A nonlinear 1D BVP solved by successive substitution . . . 45
3.11 A dynamic nonlinear initial boundary value problem . . . 46
3.12 A dynamic nonlinear wave equation . . . 47
3.13 Plane elasticity . . . 49
3.13.1 A plane stress example . . . 49
3.13.2 A plane strain example . . . 51
3.13.3 A plane stress problem with thermal expansion . . . 54
3.13.4 A plane stress eigenvalue problem and a dynamic problem . . . 56
3.14 An axially symmetric elasticity example . . . 58

4 The Commands of FEMoctave **60**
4.1 Commands for 2D meshes: creation and modification . . . 60
4.1.1 Structure of a mesh . . . 60
4.1.2 Create a uniform mesh on a rectangle: `CreateMeshRect()` . . . 62
4.1.3 Using triangle: `CreateMeshTriangle()` and `ReadMeshTriangle()` . . . 63
4.1.4 Adapting meshes and creating holes by using options of `CreateMeshTriangle()` . . . 65
4.1.5 Adding constraints to a node in the mesh . . . 68
4.1.6 Converting meshes: upgrading and downgrading . . . 68
4.1.7 Use `delaunay()` to create a mesh: `Delaunay2Mesh()` . . . 70
4.1.8 Deforming meshes by `MeshDeform()` . . . 70
4.2 Evaluation and displaying results . . . 71
4.2.1 Display results on meshes, `FEMtrimesh()`, `FEMtrisurf()`, and `FEMtricontour()` . . . 71
4.2.2 Evaluate the gradient of a function at the nodes: `FEMEvaluateGradient()` . . . 72
4.2.3 Evaluate a function and its gradient at the Gauss points: `FEMEvaluateGP()` . . . 73
4.2.4 Integrate a function over the domain: `FEMIntegrate()` . . . 73
4.2.5 Evaluation at arbitrary points or along curves, integration along curves: `FEMgriddata()` . . . 74
4.3 How to define functions . . . 75
4.3.1 Functions for static problems . . . 76
4.3.2 Functions for dynamic problems . . . 77
4.3.3 Functions for nonisotropic problems . . . 77
4.4 Solving elliptic problems . . . 78
4.4.1 Symmetric elliptic problems: `BVP2Dsym()` . . . 78
4.4.2 General elliptic problems: `BVP2D()` . . . 79
4.5 Solving 2D eigenvalue problems: `BVP2Deig()` . . . 79
4.6 Solving nonlinear problems: `BVP2DNL()` . . . 81
4.7 Solving parabolic problems: `IBVP2D()` and `IBVP2Dsym()` . . . 82
4.8 Solving semilinear parabolic problems: `IBVP2DNL()` . . . 83
4.9 Solving hyperbolic problems: `I2BVP2D()` . . . 84
4.10 Solving semilinear hyperbolic problems: `I2BVP2DNL()` . . . 85
4.11 Solving 1D steady state problems, `BVP1D()` . . . 85
4.12 Solving 1D dynamic problems of order 1, `IBVP1D()` . . . 88
4.13 Solving 1D dynamic problems of order 2, `I2BVP1D()` . . . 90
4.14 Solving 1D eigenvalue problems: `BVP1Deig()` . . . 91
4.15 Solving nonlinear 1D boundary value problems: `BVP1DNL()` . . . 92
4.16 Solving dynamic nonlinear 1D boundary value problems: `IBVP1DNL()` . . . 94
4.17 Solving second order dynamic nonlinear 1D boundary value problems: `I2BVP1DNL()` . . . 95
4.18 Plane elasticity problems . . . 96
4.18.1 Solving plane stress and plane strain problems: `PlaneStress()`, `PlaneStrain()` . . . 97
4.18.2 Eigenvalue problems, `PlaneStressEig()`, `PlaneStrainEig()` . . . 98
4.18.3 Dynamic elasticity problems, `PlaneStressDynamic()`, `PlaneStrainDynamic()` . . . 99
4.18.4 Evaluating plane stress and plane strain solutions . . . 100
4.18.5 Displaying the deformed domain, `ShowDeformation()` . . . 100
4.18.6 Evaluation of basic strain and stress: `EvaluateStrain()`, `EvaluateStress()` . . . 100
4.18.7 Evaluation of stress expressions: von Mises stress, principal stresses and Tresca stress . . . 102
4.18.8 Evaluation of the elastic energy density, `EvaluateEnergyDensity()` . . . 104
4.19 Solving axisymmetric elasticity problems, `AxiStress()` . . . 105
4.19.1 Evaluating axisymmetric solutions . . . 105
4.19.2 Evaluation of strains and stress for axisymmetric problems . . . 106
4.19.3 Evaluation of the elastic energy density, `EvaluateEnergyDensityAxi()` . . . 108
4.20 Internal commands in FEMoctave . . . 108
4.20.1 Linear elements: `FEMEquation.cc`, `FEMEquationM.m` and `FEMEquationComplex.cc` 109
4.20.2 Quadratic elements: `FEMEquationQuad.cc`, `FEMEquationQuadM.m` and `FEMEquationQuadComplex.cc` . . . 110
4.20.3 Cubic elements: `FEMEquationCubic.cc`, `FEMEquationCubicM.m` and `FEMEquationCubicComplex.cc` . . . 110
4.20.4 Hermite elements: `FEMEquationHermite.m` . . . 110
4.20.5 The command `FEMSolve.m` to solve the linear system . . . 110
4.20.6 Effect of right hand side for dynamic problems: `FEMInterpolWeight()` . . . 111

4.20.7 Effect of the Dirichlet values: `FEMInterpolBoundaryWeight()` . . . 111
4.20.8 Determine a few small eigenvalues: `eigSmall()` . . . 111
4.20.9 Generating the equations for elasticity problems . . . 112
4.21 External programs . . . 113

5 Tools for Didactical Purposes **114**
5.1 Observe the convergence of the error as $h \to 0$ . . . 114
5.2 Some element stiffness matrices . . . 116
5.2.1 Element contributions for equilateral triangles . . . 116
5.2.2 From FEM to a finite difference approximation . . . 119
5.2.3 Element stiffness matrices for 1D problems . . . 121
5.2.4 Element stiffness matrices for elasticity problems . . . 122
5.3 Behavior of a FEM solution within triangular elements . . . 123
5.4 Estimate the number of nodes and triangles in a mesh and the effect on the sparse matrix . . . 126
5.5 Compare linear, quadratic and cubic elements . . . 128
5.6 Minimizing the energy, depending on type of element . . . 129
5.7 Are second order elements C1 conforming? . . . 131
5.8 Contribution to the error by domain approximation . . . 134
5.9 Superconvergence for a 1D BVP . . . 136
5.10 Convergence of the modified Crank–Nicolson time steppers for semilinear problems . . . 138
5.11 Stability of the time steppers, or lack thereof . . . 140
5.12 Conditional stability of the explicit time stepper for a wave equation . . . 142
5.13 The shear–locking effect caused by linear elements . . . 144
5.14 Bending of an Euler beam . . . 149
5.15 Eigenvalues and eigenmodes of a slender beam . . . 152
5.16 Stress concentration on a plate with a slit . . . 154
5.17 Adding missing constraints . . . 156
5.17.1 Adding a constraint for a steady state heat problem . . . 156
5.17.2 Adding constraints for an elasticity problem . . . 157
5.18 Missing boundary constraints and null spaces . . . 159

6 The Mathematics of the Algorithms for 2D FEM **161**
6.1 Classical solutions and weak solutions . . . 161
6.2 A few triangular elements . . . 163
6.3 Transformation of coordinates, interpolation and Gauss integration . . . 164
6.3.1 Transformation of coordinates and integration over a general triangle . . . 164
6.3.2 Gauss integration on the standard triangle with 3 Gauss points . . . 165
6.3.3 Gauss integration on the standard triangle with 7 Gauss points . . . 166
6.4 Construction of first order elements . . . 167
6.4.1 Linear interpolation on a triangle . . . 168
6.4.2 Integration of $f\,\phi$ . . . 168
6.4.3 Integration of $b_0\,u\,\phi$ . . . 169
6.4.4 Integration of $a\,\nabla u \cdot \nabla\phi$ . . . 169
6.4.5 Integration of $\mathbf{a} \cdot \nabla u \cdot \nabla\phi$ with a coefficient matrix $\mathbf{a}$ . . . 170
6.4.6 Integration of $u\,\vec{b} \cdot \nabla\phi$ . . . 170
6.4.7 Integration over boundary segments . . . 171
6.5 Construction of second order elements . . . 173
6.5.1 The basis functions for a second order element and quadratic interpolation . . . 173
6.5.2 Determine values at the Gauss points and apply Gauss integration . . . 174
6.5.3 Integration of $f\,\phi$ . . . 175
6.5.4 Integration of $b_0\,u\,\phi$ . . . 175
6.5.5 Transformation of the gradient to the standard triangle . . . 176
6.5.6 Partial derivatives at the nodes . . . 179
6.5.7 Integration of $u\,\vec{b} \cdot \nabla\phi$ and $a\,\nabla u \cdot \nabla\phi$ . . . 179
6.5.8 Integration of $\mathbf{a} \cdot \nabla u \cdot \nabla\phi$ with a coefficient matrix $\mathbf{a}$ . . . 180
6.5.9 Integration over boundary segments . . . 181
6.6 Construction of third order elements . . . 182
6.6.1 The basis functions for a third order element and cubic interpolation . . . 183

6.6.2 Determine values at the Gauss points and apply Gauss integration . . . 184
6.6.3 Integration of $f\,\phi$ and $b_0\,u\,\phi$ . . . 186
6.6.4 Transformation of the gradient to the standard triangle . . . 186
6.6.5 Integration of $u\,\vec{b}\cdot\nabla\phi$ and $a\,\nabla u\cdot\nabla\phi$ . . . 189
6.6.6 Partial derivatives at the nodes . . . 189
6.6.7 Integration over boundary segments . . . 190
6.6.8 From a polynomial interpolation to the Gauss integration points on an interval . . . 192
6.7 Construction of Hermite elements . . . 193
6.7.1 The Hermite triangle . . . 193
6.7.2 The polynomial basis functions . . . 194
6.7.3 The expressions required for the element stiffness matrix . . . 198
6.7.4 Integration of $f\,\phi$ . . . 198
6.7.5 Integration of $b_0\,u\,\phi$ . . . 199
6.7.6 Integration of $u\,\vec{b}\cdot\nabla\phi$ and $a\,\nabla u\cdot\nabla\phi$ . . . 199
6.7.7 Integration over boundary segments . . . 199
6.7.8 Upgrading a linear mesh to a Hermite mesh . . . 200
6.7.9 Convert a Hermit mesh to a mesh with linear elements . . . 200
6.8 Convergence of the approximate solutions u_h to the exact solution u . . . 200
6.9 Solving semilinear problems . . . 202
6.10 Dynamic problems . . . 203
6.10.1 Dynamic problems of the heat equation type . . . 203
6.10.2 Using eigenvalues for dynamic problems of the heat equation type . . . 205
6.10.3 Semilinear dynamic problems of the heat equation type . . . 205
6.10.4 Dynamic problems of the wave equation type . . . 206
6.10.5 Using eigenvalues for dynamic problems of the wave equation type . . . 207
6.11 Inverse power iteration or `eigs()` to determine small eigenvalues of positive definite matrices . . . 208

7 The Algorithms for 1D FEM **209**
7.1 The problems to be examined . . . 209
7.2 Interpolation, Gauss integration and the element stiffness matrices . . . 210
7.3 Taking boundary conditions into account . . . 213
7.4 Solving the BVP with a system of linear equations . . . 214
7.5 Evaluation of the solution between nodes and evaluation of derivatives . . . 214
7.6 The first order dynamic problem . . . 215
7.6.1 An explicit time step . . . 216
7.6.2 An implicit time step . . . 217
7.6.3 A Crank–Nicolson time step . . . 217
7.6.4 An L–stable Runge–Kutta solver, DIRK . . . 217
7.6.5 A Crank–Nicolson solver for semilinear dynamic problems . . . 217
7.6.6 Two semi–explicit solvers for semilinear dynamic problems . . . 218
7.7 The second order dynamic problem . . . 219
7.7.1 An implicit solver . . . 219
7.7.2 An explicit solver . . . 220
7.7.3 Solvers for problems with a nonlinear expressions $f(x,t,u)$ . . . 221
7.8 Nonlinear boundary value problems, Newton's method and partial substitution . . . 222

8 The Algorithms for Plane Elasticity and Axially Symmetric Elasticity **224**
8.1 The plane stress problem . . . 224
8.2 The plane stress eigenvalue and dynamic problem . . . 225
8.3 Construction of first order elements . . . 227
8.3.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$ . . . 227
8.3.2 Integration of the terms involving derivatives of u_1, u_2, ϕ_1 and ϕ_2 . . . 227
8.3.3 The boundary integral . . . 228
8.3.4 Construct a weight matrix $\mathbf{W}$ . . . 228
8.3.5 The contribution by a thermal expansion . . . 229
8.4 Construction of second order elements . . . 229
8.4.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$ . . . 229
8.4.2 Integration of the terms involving derivatives of $\vec{u}$, ϕ_1 and ϕ_2 . . . 230

8.4.3 The boundary integral . . . 231
8.4.4 Construct a weight matrix $\mathbf{W}$ . . . 231
8.4.5 The contribution by a thermal expansion . . . 232
8.5 Construction of third order elements . . . 232
8.5.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$ . . . 232
8.5.2 Integration of the terms involving derivatives of $\vec{u}$, ϕ_1 and ϕ_2 . . . 233
8.5.3 The boundary integral . . . 234
8.5.4 Construct a weight matrix $\mathbf{W}$ . . . 234
8.6 The plane strain problem . . . 235
8.7 Elasticity for axially symmetric setups . . . 236
8.8 Construction of first order elements . . . 237
8.8.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$ . . . 237
8.8.2 Integration of the terms involving derivatives of ϕ_z and ϕ_z . . . 237
8.8.3 The boundary integral . . . 239
8.9 Construction of second order elements . . . 239
8.9.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$ . . . 239
8.9.2 Integration of the terms involving derivatives of ϕ_r and ϕ_z . . . 240
8.9.3 The boundary integral . . . 241
8.10 Construction of third order elements . . . 241
8.10.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$ . . . 241
8.10.2 Integration of the terms involving derivatives of ϕ_z and ϕ_z . . . 241
8.10.3 The boundary integral . . . 242

9 Examples, Examples, Examples 243
9.1 An elliptic problem with variable coefficients . . . 243
9.2 An animated wave . . . 244
9.3 An elliptic problem with radial symmetry, superconvergence . . . 245
9.4 An example with limited regularity . . . 248
9.5 Accuracy of Hermite elements . . . 249
9.5.1 A first Hermite example . . . 250
9.5.2 A second Hermite example . . . 252
9.6 Solving a Liouville equation . . . 254
9.7 Solving Bratu's equation . . . 256
9.7.1 Solving the Bratu equation on $[0,1]$ . . . 256
9.7.2 Solving the Bratu equation on $[0,1]\times[0,1]\subset\mathbb{R}^2$ . . . 259
9.8 Poiseuille flow through a pipe . . . 262
9.8.1 Poiseuille flow through an annular tube, 1D solution . . . 262
9.8.2 Poiseuille flow through an annular tube, 2D solution . . . 263
9.8.3 Poiseuille flow through an annular tube with an offset . . . 264
9.9 A potential flow problem . . . 264
9.10 A potential flow around a cylinder, using polar coodinates . . . 267
9.11 A potential flow problem in a circular pipe . . . 270
9.12 A potential flow around a wing profile . . . 272
9.13 A minimal surface problem . . . 276
9.14 Computing a capacitance . . . 277
9.14.1 State the problem . . . 277
9.14.2 Create the mesh and solve the BVP . . . 278
9.14.3 Compute the capacitance . . . 280
9.15 Torsion of beams, Prandtl stress function . . . 280
9.15.1 The setup with the warp function and the Prandtl stress function . . . 281
9.15.2 On a disk with radius R . . . 282
9.15.3 On a square . . . 283
9.15.4 On a rectangle . . . 284
9.16 Dynamic heat conduction problems . . . 284
9.16.1 With a narrow section in the domain . . . 285
9.16.2 With a section of lower thermal conductivity . . . 286
9.16.3 Cooling of a cylinder . . . 289
9.16.4 Heat waves . . . 291

9.16.5 Static heat equation in a ball in $\mathbb{R}^3$, solved as a 1D problem . . . 293
9.16.6 Dynamic heat equation in a cylinder, solved as a 1D problem . . . 296
9.17 Wave propagation, Kirchhoff diffraction . . . 296
9.18 Sound waves in $\mathbb{R}^2$ and $\mathbb{R}^3$ . . . 300
9.18.1 A sound wave in $\mathbb{R}^3$ with cylindrical coordinates . . . 300
9.18.2 A sound wave in $\mathbb{R}^2$ . . . 301
9.18.3 Sound waves in $\mathbb{R}^3$ and $\mathbb{R}^2$ as 1D problems . . . 302
9.19 Reflection and transmission of a wave by a change of impedance . . . 304
9.20 Sine–Gordon equation . . . 306
9.20.1 Traveling kink solutions of the Sine–Gordon equation . . . 306
9.20.2 Breather solutions of the Sine–Gordon equation . . . 307
9.21 Scattering, Helmholtz equation . . . 308
9.22 The Black–Scholes equation of mathematical finance . . . 313
9.23 Spherical harmonics . . . 316
9.24 Schrödinger's equation of quantum mechanics . . . 318
9.24.1 A free particle moving and tunneling through a potential wall . . . 318
9.24.2 Schrödinger's harmonic oscillator . . . 321
9.24.3 Energies of the hydrogen atom . . . 322
9.24.4 Ground state energy of a helium atom . . . 326
9.25 The EIT forward problem . . . 331
9.26 A catenary . . . 336
9.27 Brachistochrone problem . . . 338
9.28 Stretching of a beam . . . 340
9.29 How a Fata Morgana is appearing . . . 340
9.30 Keller's nonlinear boundary value problems . . . 342
9.30.1 Partial successive substitution . . . 342
9.30.2 Newton's method . . . 342
9.30.3 Using `BVP1DNL()` . . . 344
9.30.4 A similar problem with multiple solutions . . . 344
9.31 A pendulum problem . . . 345
9.32 A BVP with multiple nonlinear contributions . . . 346
9.33 Fisher's equation . . . 347
9.33.1 A travelling wave solution . . . 348
9.33.2 A dynamic solution . . . 349
9.33.3 A dynamic solution of the radially symmetric setup . . . 350
9.33.4 A dynamic solution in the plane . . . 351
9.34 From Salt Lake City to Zürich, the shortest connection on a sphere . . . 353
9.34.1 A solution based on successive substitution . . . 353
9.34.2 A solution using `BVP1DNL()` . . . 355
9.35 A 1D nonlinear bending beam problem . . . 356
9.35.1 Solving the BVP using Newton's algorithm . . . 357
9.35.2 Solving the BVP with the command `BVP1DNL()` . . . 358
9.35.3 Solving the BVP as final value of a dynamic problem, using `IBVP1DNL()` . . . 359
9.36 Mass transfer in a porous catalyst . . . 360
9.37 Troesch's equation . . . 362
9.38 Motion of a string . . . 363
9.39 A plane stress example by Wait and Mitchell . . . 363
9.40 Thermal bending of a bimetallic strip . . . 367
9.41 A copper plate stretched by thermal expansion . . . 370
9.42 Deformation of a plate . . . 373
9.42.1 Loading on the left boundary . . . 373
9.42.2 Loading inside of the hole . . . 375
9.42.3 Loading inside of the hole, with a thermal prestress . . . 378
9.43 A pipe under pressure . . . 378
9.43.1 As a plane strain problem . . . 378
9.43.2 As an axisymmetric problem . . . 381
9.43.3 The analytical solution . . . 382
9.44 A sphere under hydrostatic pressure . . . 383

9.45 A crook with a weight attached . . . 385
9.46 A wrench . . . 389
9.47 A rotating rubber cylinder . . . 390
9.48 A washer fastener examined as spring . . . 393
9.48.1 The setup . . . 393
9.48.2 Evaluate the force by integrating the normal stress . . . 395
9.48.3 Evaluate the force by an energy argument . . . 397
9.48.4 Comparison of linear, quadratic and cubic elements . . . 397
9.48.5 Effect of different boundary conditions . . . 397
9.49 A water dam . . . 398
9.50 A tuning fork . . . 400
9.51 Vibrations of a ring . . . 402
9.52 Hertz contact of a rigid cylinder with an elastic half space . . . 405
9.52.1 The model and the algorithm . . . 406
9.52.2 Evaluation and visual results . . . 410
9.52.3 The analytical solution based on the Hertz theory . . . 411
9.52.4 Parameter studies for different penetration depths . . . 413
9.53 Hertz contact of a rigid sphere with an elastic half space . . . 415
9.54 Elastic waves in solids . . . 419
9.54.1 A cylindrical elastic wave . . . 419
9.54.2 A planar elastic wave in a canal . . . 420
9.54.3 A planar wave moving around a turn . . . 426

Bibliography **429**

List of Figures **430**

List of Tables **435**

Index **436**

There is no such thing as "*the perfect notes*" and improvements are always possible. I welcome feedback and constructive criticism. Please let me know if you use/like/dislike the package and its essential documentation. Please send your observations and remarks to Andreas.Stahel@gmx.com .

1 Introduction

- Goals of this project:
  - Provide support material for teaching FEM. The material provided might help other instructors to explain or illustrate the methods and effects of finite element algorithms.
  - Use *Octave* to implement first, second and third order triangular elements in 2D for scalar boundary value problems. For elasticity problems plane stress and plane strain configurations are examined. For linear 1D boundary value problems and initial boundary value problems second order elements are used. A nonlinear solver for 1D problems is implemented. This leads to the *Octave* package **FEMoctave**.
  - Provide examples on how to solve steady state and dynamic heat equations, wave equations and 2D elasticity equations. A few nonlinear 1D examples are provided too.
  - The source code for all demos and examples is includes in the distribution.
- Tools provided by this project:
  - Find this document on the internet at https://andreasstahel.github.io/FEMoctave/FEMdoc.pdf and the complete *Octave* package at https://andreasstahel.github.io/FEMoctave/FEMoctave.tgz.
  - Documentation and codes are also on GitHub at https://github.com/AndreasStahel/FEMoctave and with *Octave* you should be able to install `FEMoctave` with the command `pkg install -forge femoctave`. If this fails try
    `pkg install https://github.com/AndreasStahel/FEMoctave/archive/v2.1.10.tar.gz`
  - I work exclusively with Unix systems, but it is possible to use the package on other systems by modifying the `Makefile`.
  - The only external program used in `FEMoctave` is `triangle`, an excellent mesh generator by Jonathan Shewchuk. On Debian or Ubuntu based system use the package `triangle-bin`. Instructions on how to install `triangle` on a Linux system using the source code are given in Section 4.21 on page 113. The source code of `triangle` is not included. Find source code and documentation on the web site www.cs.cmu.edu/∼quake/triangle.html.

This is **not**:

- an introduction to *Octave* (or `MATLAB`). Users are assumed to be familiar with the basics of using *Octave*. If this is not the case, may I use the occasion for a shameless add for my book *Octave and Matlab for Engineering Applications* by Springer, [Stah22].
- an introduction to FEM algorithms. For a basic (and affordable) introduction consider [TongRoss08]. The basic concept is not explained in these notes for `FEMoctave`, but many details are spelled out. I used some of the presentations for a class *Numerical Methods* for biomedical engineers at the University of Bern. In this class the main ideas of FEM are spelled out. Find the lecture notes at andreasstahel.github.io/Notes/NumMethods.pdf.
- an introduction for engineers on when and how to use the tool finite element analysis. No attempt is made to explain the mechanical, physical or electrical background of the examples.
- an introduction to partial differential equations (PDE). The user of `FEMoctave` is assumed to know which boundary and initial boundary value problems are well posed, i.e. will have a unique solution.

The structure of this document is as follows:

1 **Introduction**: a self reference.

2 **The Problems to be Examined**: for each type of problem solvable by `FEMoctave` one example is presented. This is a good starting point to find out what type of problems are examined in these notes.

3 **Illustrative Examples**: a few examples are are worked out, code and results shown. Read this section if you want to start working with `FEMoctave`.

4 **The Commands of FEMoctave**: all commands of `FEMoctave` are briefly explained and some documentation is provided. This is comparable to a manual.

5 **Tools for Didactical Purposes**: some results and illustrations that might be useful when teaching a class on the mathematical basics of FEM.

6 **The Mathematics of the Algorithms**: the mathematics of the 2D FEM algorithms is spelled out. Linear, quadratic and cubic elements on triangles are constructed. A matrix formulation is used wherever possible.

7 **The Algorithms for 1D FEM**: the mathematics for second order element of one dimensional problems are spelled out. Examined are static problems and dynamic problems of order 1 and 2, i.e. heat and wave equations. Many nonlinear boundary value problems can be solved.

8 **Elasticity**: the mathematical aspects of an FEM algorithm to solve plane stress and plane strain problems are presented. The algorithms for axially symmetric elasticity problems are explained, all leading to a matrix formulation.

9 **Examples, Examples, Examples**: as the title says.

To generate the project `FEMoctave` help was provided by: Carlo de Falco, Kai Torben Ohlhus, Heiri Schwarzenbach and Guiseppe Gonzalves.

2 The Problems to be Examined by FEMoctave

This section consists of a brief list all types of problems that can be solved with this software. A list of the necessary commands is shown in the selection trees in Figures 1, 2 below and in Table 1 on page 13. The instructions on how to use the commands are given in Section 4, starting on page 60. Many examples are worked out in Section 9, starting on page 243.

2.1 Selection trees for the FEMoctave solvers

- In Figure 1 find a selection tree for the commands of FEMoctave to solve boundary value problems and initial boundary value problems with two space dimensions.
- In Figure 2 find a selection tree for the commands of FEMoctave to solve boundary value problems and initial boundary value problems with one space dimension.
- In Figure 3 find a selection tree for the commands of FEMoctave to solve boundary value problems and initial boundary value problems for elasticity problems.

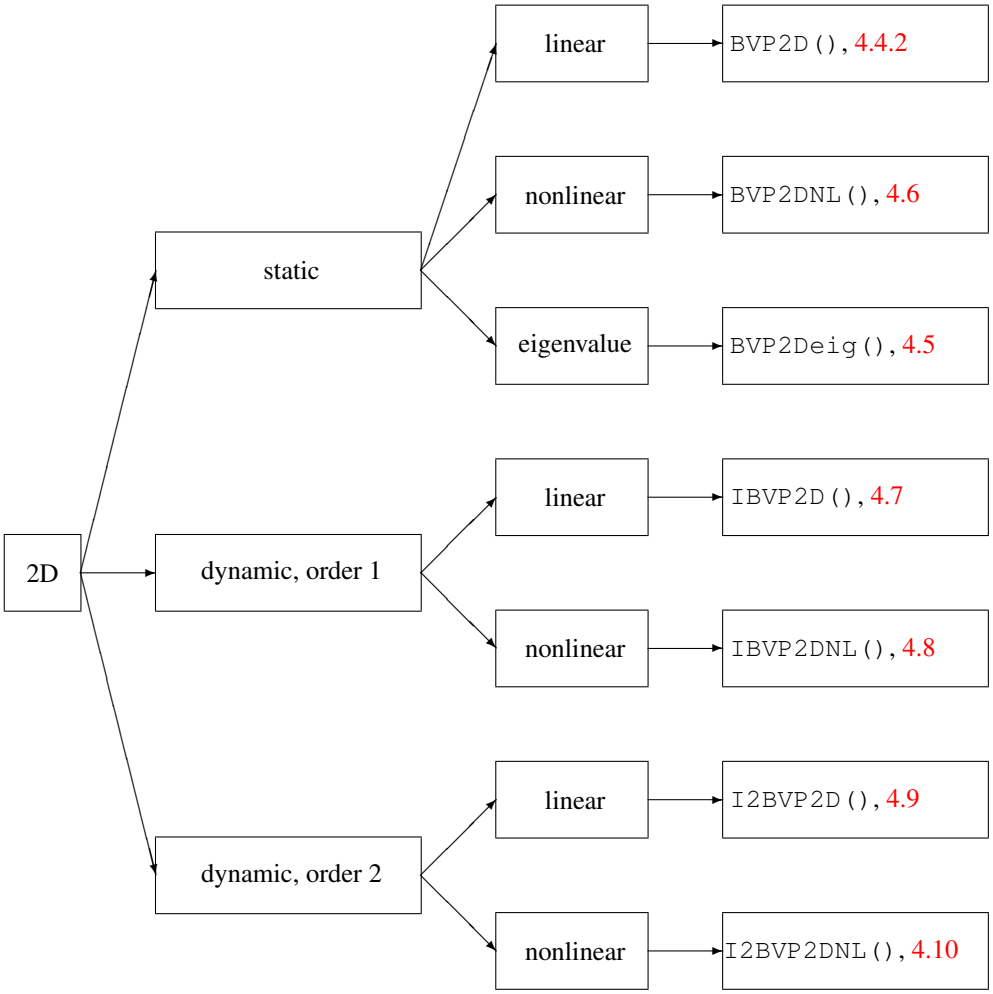


Figure 1: The selection tree for two dimensional problems. Working from left to right find the appropriate command and the section with detailed information on the problem to be solved.

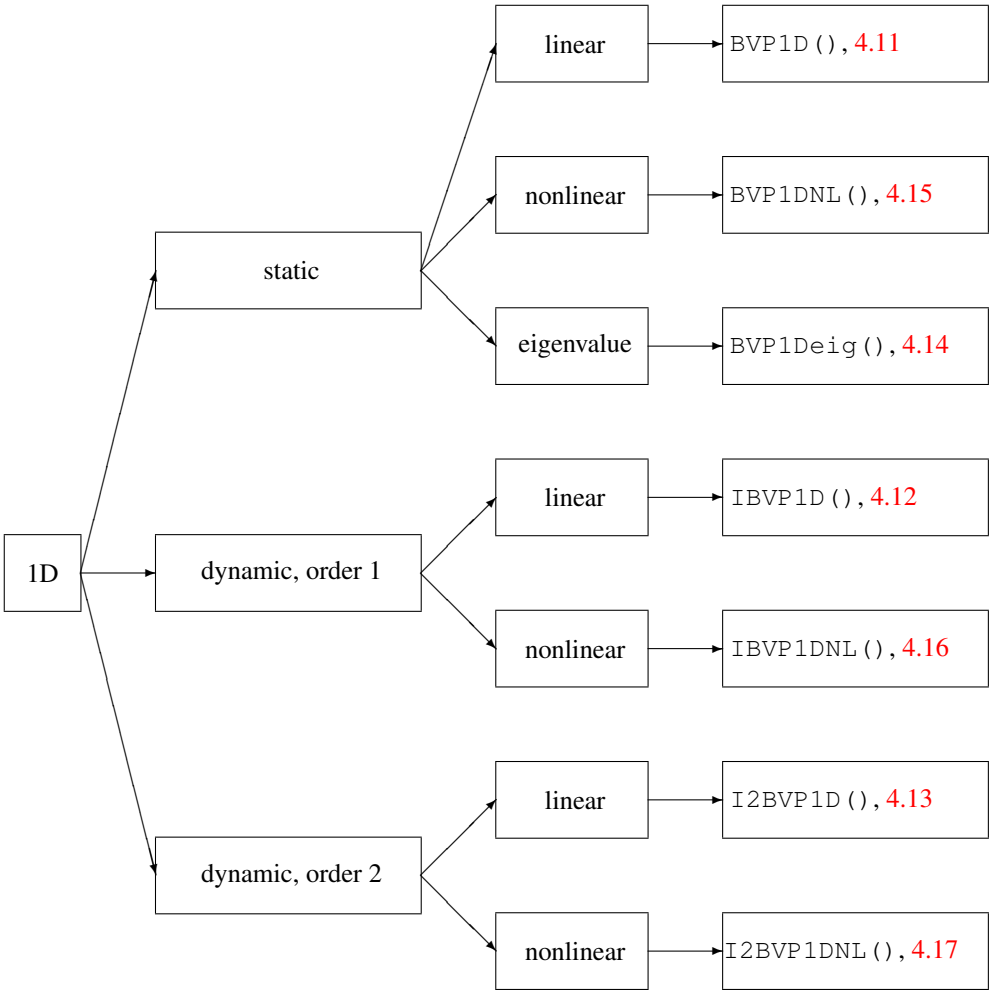


Figure 2: The selection tree for one dimensional problems. Working from left to right find the appropriate command and the section with detailed information on the problem to be solved.

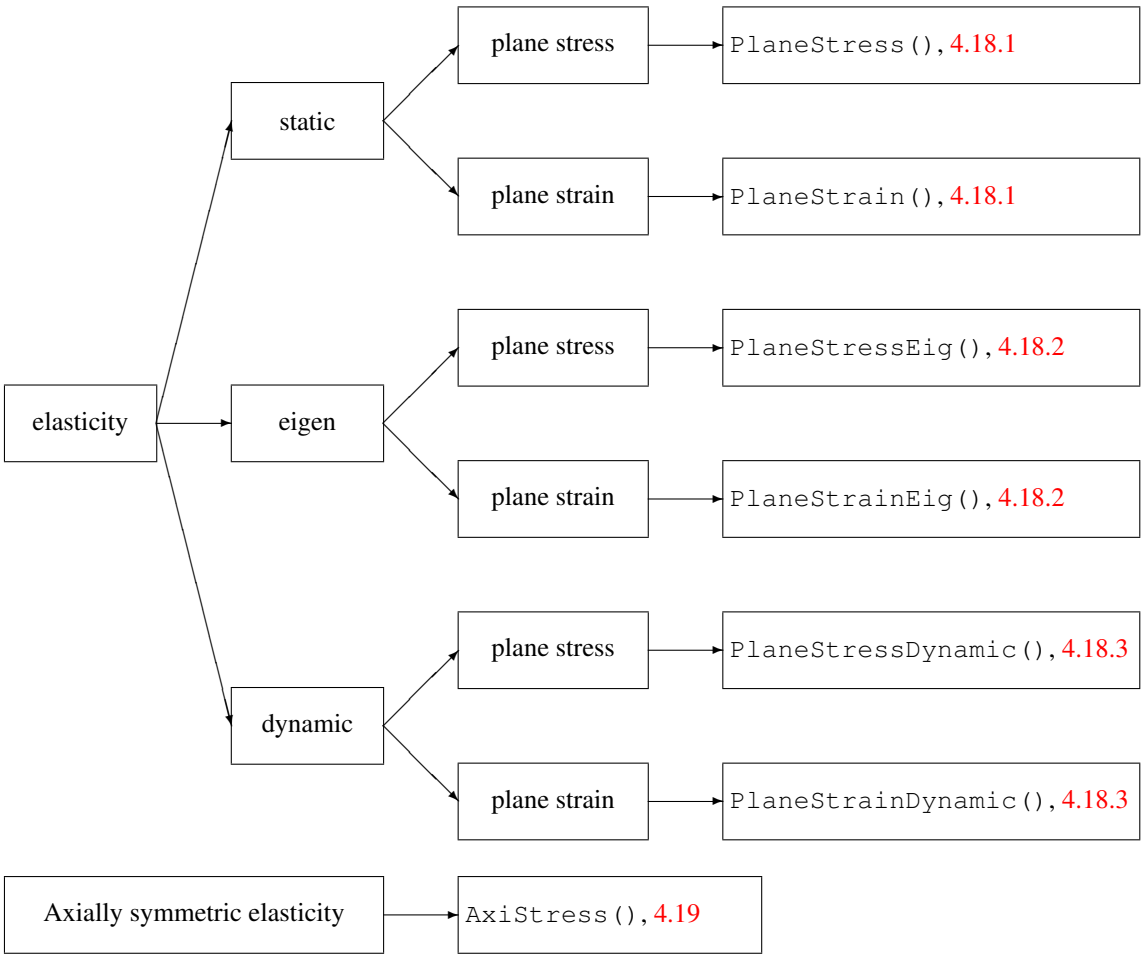


Figure 3: The selection tree for elasticity problems. Working from left to right find the appropriate command and the section with detailed information on the problem to be solved.

command	type of problem	section
BVP2Dsym()	solve a symmetric elliptic BVP	4.4.1
BVP2D()	solve a general elliptic BVP	4.4.2
BVP2Deig()	solve a symmetric elliptic eigenvalue problem	4.5
BVP2DNL()	solve a nonlinear static problem	4.6
IBVP2D()	solve a parabolic IBVP	4.7
IBVP2Dsym()	solve a symmetric parabolic IBVP	4.7
IBVP2DNL()	solve a semilinear parabolic IBVP	4.8
I2BVP2D()	solve a hyperbolic IBVP	4.9
GenerateWeight2D()	helper function for BVP2DNL()	
BVP1D()	solve a static 1D BVP	4.11
IBVP1D()	solve a first order 1D IBVP	4.12
I2BVP1D()	solve a second order 1D IBVP	4.13
BVP1Deig()	solve a 1D eigenvalue BVP	4.14
BVP1DNL()	solve a nonlinear 1D BVP	4.15
IBVP1DNL()	solve a first order semilinear 1D IBVP	4.16
I2BVP1DNL()	solve a second order semilinear 1D IBVP	4.17
PlaneStress()	solve a plane stress problem	4.18.1
PlaneStrain()	solve a plane strain problem	4.18.1
PlaneStressEig()	solve a plane stress eigenvalue problem	4.18.2
PlaneStrainEig()	solve a plane strain eigenvalue problem	4.18.2
PlaneStressDynamic()	solve a dynamic plane stress problem	4.18.3
PlaneStrainDynamic()	solve a dynamic plane strain problem	4.18.3
AxiStress()	solve an axially symmetric elasticity problem	4.19

Table 1: Commands to solve boundary value problems, initial boundary value problems and elasticity problems

2.2 The domain $\Omega \subset \mathbb{R}^2$ and its boundary $\Gamma = \partial\Omega = \Gamma_1 \cup \Gamma_2$

Throughout this presentation work with bounded domains $\Omega \subset \mathbb{R}^2$ with two disjoint sections Γ_1 and Γ_2 of the boundary $\Gamma = \partial\Omega$.

- On the section Γ_1 a Dirichlet boundary condition is applied, i.e. $u(x,y) = g_1(x,y)$ for a known function g_1.
- On the section Γ_2 a Neumann or Robin boundary condition is applied, i.e. the outer normal derivative of u equals $g_2 + g_3\, u$ for known functions g_2 and g_3.

In the example shown in Figure 4 the solution satisfies $u = +3$ on the circular part Γ_1 and $\frac{\partial}{\partial y}u = -1$ along the x–axis. The solution $u(x,y)$ solves $\Delta u = \nabla \cdot \nabla u = \operatorname{div} \operatorname{grad} u = 0$ and minimizes the functional

$$F(u) = \iint_\Omega \frac{1}{2}\|\nabla u\|^2 \, dA - \int_{\Gamma_2} u \, ds$$

amongst all functions u which satisfy the boundary condition $u(x,y) = +3$ on Γ_1.

2.3 The general elliptic problem

Let $\Omega \subset \mathbb{R}^2$ be a bounded domain with a nice boundary Γ, consisting of two disjoint sections Γ_1 and Γ_2. For given functions a, b_0, $\vec{b}$, f and g_i we seek a solution of the second order **boundary value problem** (BVP)

$$\begin{aligned} -\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\, u &= f && \text{for} \quad (x,y) \in \Omega \\ u &= g_1 && \text{for} \quad (x,y) \in \Gamma_1 \\ \vec{n} \cdot (a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\, u && \text{for} \quad (x,y) \in \Gamma_2 \end{aligned} \quad . \tag{1}$$

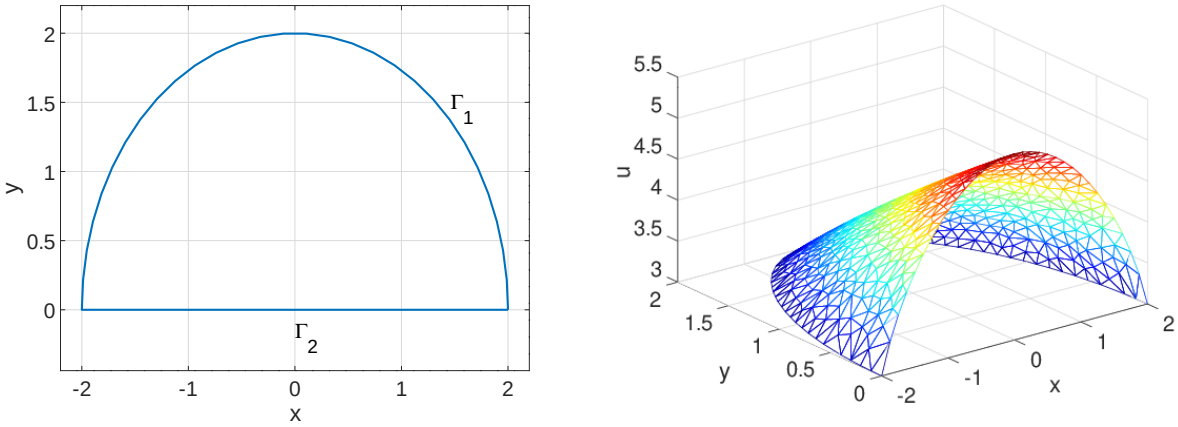


Figure 4: A semi–disk as domain in $\mathbb{R}^2$ and a solution of a BVP

It is assumed that there is a unique solution u. Consult your book on the theory of PDEs to determine whether the BVP has in fact a unique solution. Examples of this type of equation are given in Section 3.1.

2.4 The symmetric elliptic problem

If there is no convection contribution $\vec{b}$ in (1) one ends up with a self-adjoint problem.

$$\begin{aligned} -\nabla\cdot(a\,\nabla u)+b_0\,u &= f && \text{for} \quad (x,y)\in\Omega \\ u &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ a\,\tfrac{\partial u}{\partial n} &= g_2+g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned} \quad . \tag{2}$$

The resulting matrix $\mathbf{A}$ will be symmetric and if $a>0$, $b_0\geq 0$ and $\Gamma_1\neq\emptyset$ or $b_0>0$, then the BVP has a unique solution and the resulting matrix is strictly positive definite.

Using Calculus of Variations one can show that solving (2) is equivalent to minimizing a functional F among all functions u satisfying $u=g_1$ on Γ_1. For (2) use the functional

$$F(u)=\iint\limits_{\Omega}\frac{1}{2}\,a\,\langle\nabla u,\nabla u\rangle+\frac{1}{2}\,b_0\,u^2-f\,u\,dA-\int_{\Gamma_2}g_2\,u+\frac{1}{2}\,g_3\,u^2\,ds\,.$$

Examples of this type are given in Sections 3.1.1, 3.1.2, 3.1.3, 9.4, 9.21 and 9.25.

There are applications where the coefficient a is required to be a symmetric, positive definite matrix, i.e.

$$\mathbf{a}=\begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}\,.$$

In these cases the physical properties are not isotropic, i.e. they depend on the direction. The conditions for the matrix $\mathbf{a}$ to be positive definite are $a_{11}>0$, $a_{22}>0$ and $a_{12}^2<a_{11}\,a_{22}$. The boundary value problem (2) is replaced by the slightly more general problem with the modified outer normal vector $\vec{n}_{\mathbf{a}}=\mathbf{a}\,\vec{n}$.

$$\begin{aligned} -\nabla\cdot(\mathbf{a}\,\nabla u)+b_0\,u &= f && \text{for} \quad (x,y)\in\Omega \\ u &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ \tfrac{\partial u}{\partial n_{\mathbf{a}}} &= g_2+g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned} \tag{3}$$

For this non isotropic case the differential operator is modified.

$$\frac{\partial}{\partial x}\,(a\,\frac{\partial u}{\partial x})+\frac{\partial}{\partial y}\,(a\,\frac{\partial u}{\partial y})\quad\longrightarrow\quad\frac{\partial}{\partial x}\,(a_{11}\,\frac{\partial u}{\partial x}+a_{12}\,\frac{\partial u}{\partial y})+\frac{\partial}{\partial y}\,(a_{22}\,\frac{\partial u}{\partial y}+a_{12}\,\frac{\partial u}{\partial x})$$

Using Calculus of Variations solving the BVP is equivalent to minimizing the functional

$$F(u) = \iint_\Omega \frac{1}{2}\,\langle \mathbf{a}\nabla u, \nabla u\rangle + \frac{1}{2}\,b_0\,u^2 - f\,u\;dA - \int_{\Gamma_2} g_2\,u + \frac{1}{2}\,g_3\,u^2\;ds\;.$$

The energy expression is modified similarly.

$$a\left((\frac{\partial\,u}{\partial x})^2 + (\frac{\partial\,u}{\partial y})^2\right) \quad\longrightarrow\quad \langle \mathbf{a}\nabla u, \nabla u\rangle = a_{11}\,(\frac{\partial\,u}{\partial x})^2 + 2\,a_{12}\,\frac{\partial\,u}{\partial x}\,\frac{\partial\,u}{\partial y} + a_{22}\,(\frac{\partial\,u}{\partial y})^2$$

Find examples in Sections 3.1.5, 9.10 and 9.23.

2.5 The symmetric eigenvalue problem

For given functions a, b_0, f and g_3 seek values of λ and nontrivial solutions u of the **eigenvalue problem** below.

$$\begin{aligned} -\nabla\cdot(a\,\nabla u) + b_0\,u &= \lambda\,f\,u \quad &\text{for}\quad (x,y)\in\Omega \\ u &= 0 \quad &\text{for}\quad (x,y)\in\Gamma_1 \\ a\,\tfrac{\partial\,u}{\partial n} &= g_3\,u \quad &\text{for}\quad (x,y)\in\Gamma_2 \end{aligned} \tag{4}$$

An example of this type is given in Section 3.2.

2.6 The nonlinear elliptic problem

A semilinear elliptic problem is given by

$$\begin{aligned} -\nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,u) \quad &\text{for}\quad (x,y)\in\Omega \\ u &= g_1 \quad &\text{for}\quad (x,y)\in\Gamma_1 \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u \quad &\text{for}\quad (x,y)\in\Gamma_2 \end{aligned} \tag{5}$$

Examples are shown in Sections 3.1.6, 9.6 and 9.7.

2.7 The general parabolic problem

If all functions depend on time t and the spacial variables x and y consider the general dynamic heat equation.

$$\begin{aligned} \rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t) \quad &\text{for}\quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 \quad &\text{for}\quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u \quad &\text{for}\quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 \quad &\text{on}\quad \Omega\quad\text{at}\quad t=t_0 \end{aligned} \tag{6}$$

This is an Initial Boundary Value Problem (IBVP). Mathematicians call this a parabolic problem, engineers think of dynamic heat equations. Examples are shown in Sections 3.3 and 9.16.4.

2.8 The symmetric parabolic problem

Consider the symmetric situation of (6) to find the symmetric parabolic problem below.

$$\begin{aligned} \rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u) + b_0\,u &= f(x,y,t) \quad &\text{for}\quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 \quad &\text{for}\quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ a\,\tfrac{\partial\nabla u}{\partial n} &= g_2 + g_3\,u \quad &\text{for}\quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 \quad &\text{on}\quad \Omega\quad\text{at}\quad t=t_0 \end{aligned} \tag{7}$$

If $u(x,y)$ and λ are solutions of the eigenvalue problem (4) with $f = g_1 = g_2 = 0$, then the dynamic problem (7) is solved by $e^{-\lambda t}\,u(x,y)$. See also Section 6.10.2.

2.9 The semilinear parabolic problem

The general semilinear, parabolic problem is of the form

$$\begin{aligned} \rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t,u) && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \end{aligned} \tag{8}$$

2.10 The hyperbolic problem

Examine an IBVP of hyperbolic type, with the standard wave equation $\ddot{u} = \Delta u$ as the typical example.

$$\begin{aligned} \rho\,\tfrac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t) && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \\ \tfrac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \end{aligned} \tag{9}$$

Examples are shown in Sections 3.5, 9.2 and 9.17. The effect of eigenvalues is described in Section 6.10.5.

2.11 The semilinear hyperbolic problem

Examine a semilinear IBVP of hyperbolic type, with the standard wave equation $\ddot{u} = \Delta u$ as the typical example.

$$\begin{aligned} \rho\,\tfrac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t,u) && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \\ \tfrac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \end{aligned} \tag{10}$$

An example is shown in Section 3.6.

2.12 1D boundary value problems

A 1D BVP (boundary value problem) is an ordinary differential equation for the dependent variable $u(x)$ with the independent varaiable x in an interval $x_0 \le x \le x_n$ of the form

$$-\frac{d}{dx}\left(a(x)\,\frac{d\,u(x)}{dx}\right) + b(x)\,\frac{d\,u(x)}{dx} + c(x)\,u(x) = d(x)\,f(x) \tag{11}$$

with some boundary conditions at $x = x_0$ and $x = x_n$.

$$\begin{aligned} u(x_i) &= g_D && \text{Dirichlet boundary condition} \\ a(x_i)\,\tfrac{d\,u(x_i)}{dx} &= g_{N1} + g_{N2}\,u(x_i) && \text{Neumann boundary condition} \end{aligned} \tag{12}$$

The coefficient functions $a(x)$, $b(x)$, $c(x)$ and $d(x)$ will be evaluated at the Gauss points. The function $f(x)$ and the solution $u(x)$ are evaluated (resp. determined) at the nodes.

Find examples in Sections 3.7, 5.7, 5.9 and 9.28.

2.13 1D initial boundary value problems of order 1

The dynamic problem is of the form

$$w(x)\,\frac{\partial\,u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t) \tag{13}$$

with the initial condition $u(x,t_0) = u_0(x)$ and appropriate boundary conditions. These have to be independent on time t.

Find examples in Sections 3.8, 5.11, 9.16.6, 9.22 and 9.24.1.

2.14 1D initial boundary value problems of order 2

The dynamic problem is of the form

$$w_2(x)\,\frac{\partial^2 u(x,t)}{\partial t^2} + w_1(x)\,\frac{\partial\, u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\, u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\, u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t) \quad (14)$$

with the initial conditions $u(x,t_0) = u_0(x)$, $\frac{\partial}{\partial t}\,u(x,t_0) = v_0$ and appropriate boundary conditions. These have to be independent on time t.

Find examples in Sections 3.9, 5.12, 9.18.3, 9.19 and 9.38.

2.15 1D eigenvalue value problems

For given functions a, b, c, w and g_{N2} seek values of λ and nontrivial solutions u of the **eigenvalue problem** below.

$$-\frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\, u(x)}{\partial x}\right) + b(x)\,\frac{\partial\, u(x)}{\partial x} + c(x)\,u(x) = \lambda\, w(x)\,u(x) \quad (15)$$

with appropriate boundary conditions

$$\begin{array}{rcll} u(x_i) & = & 0 & \text{Dirichlet} \\ a(x_i)\,\frac{\partial\, u(x_i)}{\partial x} & = & g_{N2}\,u(x_i) & \text{Neumann} \end{array} \quad (16)$$

Find examples in Sections 3.2 and 9.24.2.

2.16 Nonlinear 1D boundary value problems

For given functions a, b, c and d and a function $f(x,u)$ or $f(x,u,u')$ search for solutions of the BVP

$$-\frac{\partial}{\partial x}\left(a(x,u(x),u'(x))\,\frac{\partial\, u(x)}{\partial x}\right) + b(x)\,\frac{\partial\, u(x)}{\partial x} + c(x)\,u(x) = d(x)\,f(x,u(x),u'(x)) \quad (17)$$

with linear boundary conditions, Dirichlet or Neumann.

Find examples in Sections 3.10.1, 3.10.2, 9.20 and 9.26–9.37.

2.17 Dynamic semilinear 1D initial boundary value problems

For given functions a, b, c and d and a function $f(t,x,u)$ search for solutions $u(x,t)$ of the IBVP

$$w(x)\,\frac{\partial\, u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\, u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\, u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t,u(x,t)) \quad (18)$$

with linear boundary conditions, Dirichlet or Neumann and a given initial condition $u(x,t_0) = u_0(x)$.

Find examples in Sections 3.11, 9.33 and 9.35.3.

2.18 Dynamic semilinear 1D wave equations

For given functions a, b, c and d and a function $f(t,x,u)$ search for solutions $u(x,t)$ of the IBVP

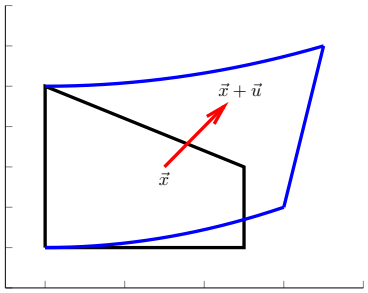
$$w_2(x)\,\frac{\partial\, u}{\partial t} + w_1(x)\,\frac{\partial\, u}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\, u}{\partial x}\right) + b(x)\,\frac{\partial\, u}{\partial x} + c(x)\,u = d(x)\,f(x,t,u(x,t)) \quad (19)$$

with linear boundary conditions and given initial conditions $u(x,t_0) = u_0(x)$ and $\frac{\partial}{\partial t}\,u(x,t_0) = v_0(x)$.

Find examples in Sections 3.12 and 9.20.

2.19 Plane elasticity

With FEMoctave plane elasticity problems can be examined, either plane stress or plane strain configurations.



$$\begin{array}{ccc} \vec{x} & \longrightarrow & \vec{x}+\vec{u} \\ \begin{pmatrix} x \\ y \end{pmatrix} & \longrightarrow & \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_1(x,y) \\ u_2(x,y) \end{pmatrix} \end{array}$$

Figure 5: Deformation of an elastic solid

2.19.1 Description of strain

The initial goal is to determine the displacement function $\vec{u} = (u_1, u_2)$. It describes the displacement of arbitrary points $(x, y) \in \Omega \subset \mathbb{R}^2$ in the solid.

Based on the displacement $\vec{u}(x, y)$ the infinitesimal strain tensor is given by

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right) & \frac{\partial u_2}{\partial y} \end{bmatrix}.$$

This strain tensor contains the essential information of how a small section of the large solid is deformed, see Figure 6 for a small section of the domain.

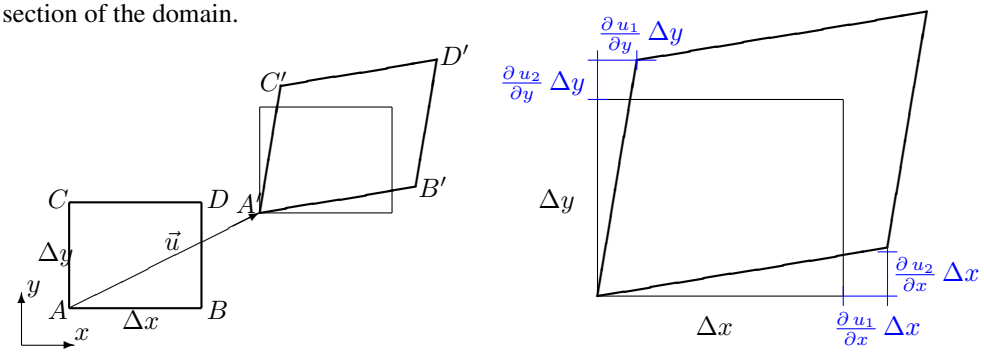


Figure 6: Definition of strain: a small rectangle before and after deformation

Obviously this can be used in the space $\mathbb{R}^3$ too, leading to the 3×3 strain matrix (or tensor of order 2)

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u_1}{\partial z} + \frac{\partial u_3}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right) & \frac{\partial u_2}{\partial y} & \frac{1}{2}\left(\frac{\partial u_2}{\partial z} + \frac{\partial u_3}{\partial y}\right) \\ \frac{1}{2}\left(\frac{\partial u_1}{\partial z} + \frac{\partial u_3}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u_2}{\partial z} + \frac{\partial u_3}{\partial y}\right) & \frac{\partial u_3}{\partial z} \end{bmatrix}$$

and the corresponding geometric interpretations shown in Table 2.

2.19.2 Description of stress and Hooke's law

The deformation of the solid will lead to normal and shearing stress, with the units forces per area, i.e. $\frac{\mathrm{N}}{\mathrm{m}^2}$. Find a graphical interpretation of the 6 strains in space $\mathbb{R}^3$ in Figure 7 and a description in Table 3.

With FEMoctave there are three types of boundary conditions to be examined:

$$\begin{array}{rcll} \vec{u} & = & \vec{g}_D & \text{on Dirichlet boundary } \Gamma_1\text{, i.e. prescribed displacement} \\ \text{force density} & = & \vec{g}_N & \text{on Neumann boundary } \Gamma_2\text{, i.e. prescribed force density} \\ \text{force density} & = & \vec{0} & \text{on free boundary } \Gamma_3 \end{array} \tag{20}$$

symbol	formula	interpretation
ε_{xx}	$\frac{\partial\, u_1}{\partial x}$	ratio of change of length divided by length in x direction
ε_{yy}	$\frac{\partial\, u_2}{\partial y}$	ratio of change of length divided by length in y direction
ε_{zz}	$\frac{\partial\, u_3}{\partial z}$	ratio of change of length divided by length in z direction
$\varepsilon_{xy} = \varepsilon_{yx}$	$\frac{1}{2}\left(\frac{\partial\, u_1}{\partial y} + \frac{\partial\, u_2}{\partial x}\right)$	the angle between the x and y axis is diminished by $2\,\varepsilon_{xy}$
$\varepsilon_{xz} = \varepsilon_{zx}$	$\frac{1}{2}\left(\frac{\partial\, u_1}{\partial z} + \frac{\partial\, u_3}{\partial x}\right)$	the angle between the x and z axis is diminished by $2\,\varepsilon_{xz}$
$\varepsilon_{yz} = \varepsilon_{zy}$	$\frac{1}{2}\left(\frac{\partial\, u_2}{\partial z} + \frac{\partial\, u_3}{\partial y}\right)$	the angle between the y and z axis is diminished by $2\,\varepsilon_{yz}$

Table 2: Normal and shear strains in space

symbol	description
σ_x	normal stress at a surface orthogonal to $x = \text{const}$
σ_y	normal stress at a surface orthogonal to $y = \text{const}$
σ_z	normal stress at a surface orthogonal to $z = \text{const}$
$\tau_{xy} = \tau_{yx}$	tangential stress in y direction at surface orthogonal to $x = \text{const}$ tangential stress in x direction at surface orthogonal to $y = \text{const}$
$\tau_{xz} = \tau_{zx}$	tangential stress in z direction at surface orthogonal to $x = \text{const}$ tangential stress in x direction at surface orthogonal to $z = \text{const}$
$\tau_{yz} = \tau_{zy}$	tangential stress in z direction at surface orthogonal to $y = \text{const}$ tangential stress in y direction at surface orthogonal to $z = \text{const}$

Table 3: Description of normal and tangential stress in space

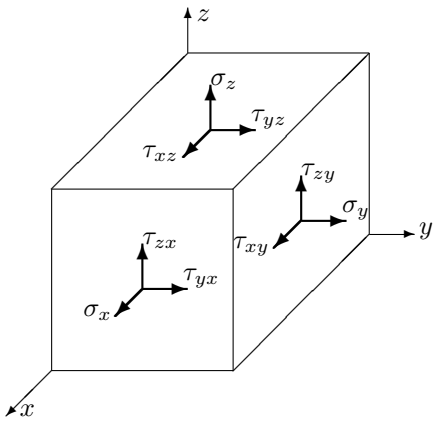


Figure 7: Components of stress in space

The conditions can be set for each component individually, find the codes in Table 6, to be used when creating meshes by `CreateMeshRect()` or `CreateMeshTriangle()`.

For a linear material law the connection between stresses and strains is given by Hooke's law and uses two material parameters:

- E: the Young's modulus of elasticity
- ν: the Poisson ratio, with $0 \leq \nu \leq \frac{1}{2}$

FEMoctave is based on the general form of **Hooke's law** for isotropic (independent on direction) materials. It is a fundamental physical law[1], confirmed by many measurements. The shown formulation is valid as long as all stress and strains are small. Hooke's law is the foundation of linear elasticity and any book on elasticity will show a formulation, e.g. [Prze68, §2.2][2], [Sout73, §2.7], or [Wein74, §10.1]. Hooke's law is given by

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \qquad \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} = \frac{1+\nu}{E} \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} \tag{21}$$

or by inverting the matrix

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \qquad \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} = \frac{E}{1+\nu} \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}. \tag{22}$$

This leads to an elastic energy density of

$$W = \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle \tag{23}$$

$$= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \tag{24}$$

$$+ \frac{E}{1+\nu} \left\langle \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle .$$

2.19.3 Thermal expansion

If a material is heated to a temperature difference ΔT, then most material will expand by a factor of $\alpha \Delta T$. The material parameter α is called the coefficient of thermal expansion. Hooke's law (21) is modified to

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} + \alpha \Delta T \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \vec{\varepsilon}_E + \vec{\varepsilon}_T \,. \tag{25}$$

[1] One can verify that for homogeneous, isotropic materials a linear law must have this form, e.g. [Sege77]

[2] The missing factors of 2 are due to the different definition of the shear strains.

The additional thermal strains ε_T lead to additional stresses, i.e. Hooke's law (22) is modified.

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} = \frac{E}{(1+\nu)\,(1-2\,\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} - \alpha\,\Delta T\,\frac{E}{1-2\,\nu} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{26}$$

See [Prze68, p. 15, (2.12)]. The additional contribution satisfies the key property that strains $\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{zz} = \alpha\,\Delta T$ leads to zero stresses, i.e. a purely thermal expansion does not lead to stress.

2.19.4 Plane stress problems

For a plane stress problem it is assumed that there are no stresses in z–direction, i.e.

$$\sigma_z = \tau_{xz} = \tau_{yz} = 0\ .$$

This leads to a simpler version of Hooke's law

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \quad \text{and} \quad \begin{aligned} \varepsilon_{zz} &= \tfrac{-\nu}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy}) \\ \varepsilon_{xz} &= 0 \\ \varepsilon_{yz} &= 0 \end{aligned}\ . \tag{27}$$

The energy density given by equation (23) simplifies to

$$\begin{aligned} W_{stress} &= \frac{1}{2}\,\Big\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \Big\rangle + \Big\langle \begin{pmatrix} \tau_{xy} \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \Big\rangle = \frac{1}{2}\,\Big\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 2\,\tau_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \Big\rangle \\ &= \frac{E}{2\,(1-\nu^2)}\,\Big\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \Big\rangle \\ &= \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2\right)\ . \end{aligned} \tag{28}$$

Since

$$\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} = \nu\,(\varepsilon_{xx}+\varepsilon_{yy})^2 + (1-\nu)\,(\varepsilon_{xx}^2+\varepsilon_{yy}^2) \geq 0$$

the energy density W_{stress} is assured to be positive. With this the total energy of a plane stress problem can be written in the form[3]

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\Big\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \Big\rangle\, dA - \\ &\quad - \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\, dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\, ds\ . \end{aligned} \tag{29}$$

Using the Bernoulli principle this energy has to be minimized. It is this minimization problem that is solved, subject to the boundary conditions (20). One can verify (e.g. [Stah08]) that the corresponding Euler–Lagrange equations[4] are given by

$$\begin{aligned} -\operatorname{div}\left(\tfrac{E}{1-\nu^2}\begin{pmatrix} \frac{\partial u_1}{\partial x} + \nu\,\frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2}\,(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \end{pmatrix}\right) &= f_1 \\ -\operatorname{div}\left(\tfrac{E}{1-\nu^2}\begin{pmatrix} \frac{1-\nu}{2}\,(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \\ \frac{\partial u_2}{\partial y} + \nu\,\frac{\partial u_1}{\partial x} \end{pmatrix}\right) &= f_2 \end{aligned} \tag{30}$$

[3] We quietly dropped the constant thickness H from all expressions.

[4] This author is convinced that it is easier and more efficient to work with the energy to be minimized and not the system of partial differential equations.

If E and ν are constant, i.e. a homogeneous material, use elementary, tedious operations to find a shorter notation for the above system of PDEs.

$$-\frac{E}{2\,(1+\nu)}\,\left(\Delta\vec{u}+\frac{1+\nu}{1-\nu}\,\vec{\nabla}\left(\vec{\nabla}\vec{u}\right)\right)=\vec{f}$$

The corresponding dynamic problem for $u_1(x,y,t)$ and $u_2(x,y,t)$ with external force densities f_1 and f_2 is given by

$$\begin{aligned}
-\operatorname{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y}\\ \frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\end{pmatrix}\right)+f_1 &= \rho\,\frac{\partial^2}{\partial t^2}\,u_1\\
-\operatorname{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\\ \frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x}\end{pmatrix}\right)+f_2 &= \rho\,\frac{\partial^2}{\partial t^2}\,u_2
\end{aligned}\tag{31}$$

The eigenvalue problem

$$\begin{aligned}
-\operatorname{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y}\\ \frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\end{pmatrix}\right) &= \lambda\,\rho\,u_1\\
-\operatorname{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\\ \frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x}\end{pmatrix}\right) &= \lambda\,\rho\,u_2
\end{aligned}\tag{32}$$

leads to harmonic solutions of the form

$$\cos(\sqrt{\lambda}\,t+\delta)\,\begin{pmatrix}u_1(x,y)\\ u_2(x,y)\end{pmatrix}$$

of the dynamic problem (31) (with $\vec{f}=\vec{0}$) with frequency $\frac{\omega}{2\,\pi}=\frac{\sqrt{\lambda}}{2\,\pi}$.

If thermal expansion is taken into account use Hooke's law in the form

$$\begin{pmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{pmatrix}=\frac{E}{1-\nu^2}\,\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & 1-\nu\end{bmatrix}\,\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}-\frac{E\,\alpha\,\Delta T}{1-\nu}\,\begin{pmatrix}1\\ 1\\ 0\end{pmatrix}$$

See e.g. [Prze68, (2.28)] or [ShamDym95, p.63][5]. For the strain in z–direction find

$$\varepsilon_{zz}=\frac{-\nu}{E}\,(\sigma_x+\sigma_y)+\alpha\,\Delta T=\frac{-\nu}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy})+\frac{1+\nu}{1-\nu}\,\alpha\,\Delta T\;.\tag{33}$$

There is an additional contribution to the energy density W_{stress} in expression (28).

$$\begin{aligned}
W_{stress} &= \frac{1}{2}\,\langle\begin{pmatrix}\sigma_x\\ \sigma_y\\ 2\,\tau_{xy}\end{pmatrix},\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle-\frac{E\,\alpha\,\Delta T}{1-\nu}\,\langle\begin{pmatrix}1\\ 1\\ 0\end{pmatrix},\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle\\
&= \frac{1}{2}\,\frac{E}{1-\nu^2}\,\langle\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix},\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & 2\,(1-\nu)\end{bmatrix}\,\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle-\frac{E\,\alpha\,\Delta T}{1-\nu}\,\langle\begin{pmatrix}1\\ 1\\ 0\end{pmatrix},\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle\\
&= \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy}+2\,(1-\nu)\,\varepsilon_{xy}^2\right)-\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy})
\end{aligned}\tag{34}$$

Find examples in Sections 3.13.1, 3.13.3, 9.39, 5.13 5.14, 5.16, 9.40, 9.42, 9.43, 9.45, 9.46, 9.47, 9.48, 9.49, 9.50, 9.51, 9.52, 9.53 and 9.54 .

[5] With a typo in the formula.

2.19.5 Plane strain problems

For a plane strain problem it is assumed that there are no strains in z–direction, i.e.

$$\varepsilon_{xz} = \varepsilon_{yz} = \varepsilon_{zz} = 0 \; .$$

This leads to a simpler version of Hooke's law

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)\,(1-2\,\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\,\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \quad . \tag{35}$$
$$\sigma_z = \frac{E\,\nu\;(\varepsilon_{xx}+\varepsilon_{yy})}{(1+\nu)\,(1-2\,\nu)} \quad , \quad \tau_{xz} = \tau_{yz} = 0$$

Observe that

$$\sigma_z = \frac{E\,\nu\;(\varepsilon_{xx}+\varepsilon_{yy})}{(1+\nu)\,(1-2\,\nu)} = \nu\,(\sigma_x+\sigma_y) \; .$$

Modify the material parameters ν and E to

$$\nu^\star = \frac{\nu}{1-\nu} > \nu \quad \text{and} \quad E^\star = \frac{E}{1-\nu^2} > E \; . \tag{36}$$

Then use elementary algebra to find

$$\begin{aligned} \nu &= \frac{\nu^\star}{1+\nu^\star} \quad , \quad 1-\nu = 1-\frac{\nu^\star}{1+\nu^\star} = \frac{1}{1+\nu^\star} \\ \frac{1-2\,\nu}{1-\nu} &= \frac{1-2\,\frac{\nu^\star}{1+\nu^\star}}{1-\frac{\nu^\star}{1+\nu^\star}} = 1-\nu^\star \quad \text{and} \quad \frac{\nu}{1-2\,\nu} = \frac{\frac{\nu^\star}{1+\nu^\star}}{1-2\,\frac{\nu^\star}{1+\nu^\star}} = \frac{\nu^\star}{1-\nu^\star} \\ E &= E^\star\,(1-\nu^2) \end{aligned}$$

leading to a different notation for Hooke's law for the plane strain situation.

$$\begin{aligned} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} &= \frac{E}{1+\nu} \begin{bmatrix} \frac{1-\nu}{1-2\,\nu} & \frac{\nu}{1-2\,\nu} & 0 \\ \frac{\nu}{1-2\,\nu} & \frac{1-\nu}{1-2\,\nu} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} = \frac{E^\star\,(1-\nu^2)}{1+\nu} \begin{bmatrix} \frac{1}{1-\nu^\star} & \frac{\nu^\star}{1-\nu^\star} & 0 \\ \frac{\nu^\star}{1-\nu^\star} & \frac{1}{1-\nu^\star} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \\ &= \frac{E^\star}{(1-\nu^\star)\,(1+\nu^\star)} \begin{bmatrix} 1 & \nu^\star & 0 \\ \nu^\star & 1 & 0 \\ 0 & 0 & 1-\nu^\star \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \; . \end{aligned}$$

This is identical to Hooke's law (27) for the plane stress situation, but with $E^\star = \frac{E}{1-\nu^2}$ and $\nu^\star = \frac{\nu}{1-\nu}$ instead of E and ν. The energy density is in this case given by

$$\begin{aligned} W_{strain} &= \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2\,(1-2\,\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle \\ &= \frac{E\,(1-\nu)}{2\,(1+\nu)\,(1-2\,\nu)} \left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\frac{\nu}{1-\nu}\,\varepsilon_{xx}\varepsilon_{yy} + 2\,\frac{1-2\,\nu}{1-\nu}\,\varepsilon_{xy}^2 \right) \\ &= \frac{1}{2}\,\frac{E^\star}{1-(\nu^\star)^2} \left\langle \begin{bmatrix} 1 & \nu^\star & 0 \\ \nu^\star & 1 & 0 \\ 0 & 0 & 2\,(1-\nu^\star) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle \\ &= \frac{E^\star}{2\,(1-(\nu^\star)^2)} \left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu^\star\,\varepsilon_{xx}\varepsilon_{yy} + 2\,(1-\nu^\star)\,\varepsilon_{xy}^2 \right) \end{aligned} \tag{37}$$

Now the plane strain energy density has the same form as the plane stress energy density, but with modified constants.

$$W_{stress} = \frac{E}{2\,(1-\nu^2)} \left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2 \right) \; .$$

For a plane strain problem Bernoulli's principle is used and the corresponding total energy minimized, similar to expression (29). As a consequence the PDE (30), the dynamic equation (31) and the eigenvalue problem (32) can be adapted to the plane strain situation by using $E^\star$ and $\nu^\star$.

If thermal expansion is taken into account use Hooke's law in the form

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} - \frac{E\,\alpha\,\Delta T}{1-2\nu} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

See e.g. [Prze68, (2.37)]. For the stress in z–direction find

$$\sigma_z = \nu(\sigma_x + \sigma_y) - E\,\alpha\,\Delta T = \frac{\nu E(\varepsilon_{xx}+\varepsilon_{yy})}{(1+\nu)(1-2\nu)} - \frac{E\,\alpha\,\Delta T}{1-2\nu}\,. \tag{38}$$

There is an additional contribution to the energy density W_{stress} in expression (37).

$$\begin{aligned} W_{stress} &= \frac{1}{2}\left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 2\tau_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle - \frac{E\,\alpha\,\Delta T}{1-2\nu}\left\langle \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle \\ &= \frac{1}{2}\frac{E}{(1+\nu)(1-2\nu)}\left\langle \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle - \\ &\qquad - \frac{E\,\alpha\,\Delta T}{1-2\nu}\left\langle \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle \\ &= \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)}\left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\frac{\nu}{1-\nu}\varepsilon_{xx}\varepsilon_{yy} + 2\frac{1-2\nu}{1-\nu}\varepsilon_{xy}^2\right) - \frac{E\,\alpha\,\Delta T}{1-2\nu}(\varepsilon_{xx}+\varepsilon_{yy}) \end{aligned} \tag{39}$$

Find examples in Sections 3.13.2, 3.13.3, 3.13.4 and 9.49.

2.20 Elasticity problems for axisymmetric solids, using cylindrical coordinates

Examine a domain $(x,r) = (r,z) \in \Omega \subset \mathbb{R}^2$ and revolve this domain about the z–axis to generate a volume in space $\mathbb{R}^3$. Assume that the displacements are axisymmetric, i.e.

$$\begin{pmatrix} u_1(x,y,z) \\ u_2(x,y,z) \\ u_3(x,y,z) \end{pmatrix} = \begin{pmatrix} u_r(r,z)\,\cos\varphi \\ u_r(r,z)\,\sin\varphi \\ u_z(r,z) \end{pmatrix}\,.$$

To determine the elastic energy in this deformed solid determine[6] the strains in the plane $\varphi = 0$.

$$\begin{aligned} \varepsilon_{xx} &= \frac{\partial u_1}{\partial x} = \cos^2\varphi\,\frac{\partial u_r}{\partial r} + \frac{\sin^2\varphi}{r}\,u_r = \frac{\partial u_r}{\partial r} \\ \varepsilon_{yy} &= \frac{\partial u_2}{\partial y} = \sin^2\varphi\,\frac{\partial u_r}{\partial r} + \frac{\cos^2\varphi}{r}\,u_r = \frac{1}{r}\,u_r \\ \varepsilon_{zz} &= \frac{\partial u_z}{\partial z} \end{aligned}$$

[6] For functions $f(x,y,z) = F(r,\varphi,z)$ (i.e. the identical function written in Cartesian and polar coordinates) use the computational rule (chain rule) to conclude

$$\frac{\partial f}{\partial x} = \cos\varphi\,\frac{\partial F}{\partial r} - \frac{\sin\varphi}{r}\,\frac{\partial F}{\partial \varphi} \quad \text{and} \quad \frac{\partial f}{\partial y} = \sin\varphi\,\frac{\partial F}{\partial r} + \frac{\cos\varphi}{r}\,\frac{\partial F}{\partial \varphi}\,.$$

$$
\begin{aligned}
2\,\varepsilon_{xy} &= \frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x} = \cos\varphi\,\sin\varphi\,\frac{\partial\,u_r}{\partial r} - \frac{\cos\varphi\,\sin\varphi}{r}\,u_r + \cos\varphi\,\sin\varphi\,\frac{\partial\,u_r}{\partial r} - \frac{\sin\varphi\,\cos\varphi}{r}\,u_r = 0 \\
2\,\varepsilon_{xz} &= \frac{\partial\,u_1}{\partial z} + \frac{\partial\,u_3}{\partial x} = \cos\varphi\,\frac{\partial\,u_r}{\partial z} + \cos\varphi\,\frac{\partial\,u_z}{\partial r} - \frac{\sin\varphi}{r}\,\frac{\partial\,u_z}{\partial\varphi} = \frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r} \\
2\,\varepsilon_{yz} &= \frac{\partial\,u_2}{\partial z} + \frac{\partial\,u_3}{\partial y} = \sin\varphi\,\frac{\partial\,u_r}{\partial z} + \sin\varphi\,\frac{\partial\,u_z}{\partial r} + \frac{\cos\varphi}{r}\,\frac{\partial\,u_z}{\partial\varphi} = 0
\end{aligned}
$$

This leads to

$$
\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} = \begin{pmatrix} \frac{\partial\,u_r}{\partial r} \\ \frac{1}{r}\,u_r \\ \frac{\partial\,u_z}{\partial z} \\ 0 \\ \frac{1}{2}\,(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}) \\ 0 \end{pmatrix} = \begin{pmatrix} \varepsilon_{rr} \\ \varepsilon_{\varphi\varphi} \\ \varepsilon_{zz} \\ 0 \\ \varepsilon_{rz} \\ 0 \end{pmatrix} .
$$

Observe that the angular strain $\varepsilon_{\varphi\varphi}$ is given by the displacement $\varepsilon_{\varphi\varphi} = \frac{1}{r}\,u_r$. At nodes with $r = 0$ use de l'HÃ´pital's rule to conclude $\varepsilon_{\varphi\varphi} = \lim_{r\to 0}\frac{u_r(r,z)}{r} = \frac{\partial\,u_r(0,z)}{\partial r} = \varepsilon_{xx}(0,z)$.

The energy density (23) in the rz–plane at angle $\varphi = 0$ is given by

$$
\begin{aligned}
W(r,z) &= \frac{1}{2}\,\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle + \langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} 0 \\ \varepsilon_{xz} \\ 0 \end{pmatrix} \rangle \\
&= \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{rr} \\ \frac{1}{r}\,u_r \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{rr} \\ \frac{1}{r}\,u_r \\ \varepsilon_{zz} \end{pmatrix} \rangle + \frac{E}{1+\nu}\,\varepsilon_{rz}^2 \\
&= \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,(\varepsilon_{rr}^2 + \varepsilon_{zz}^2 + \frac{1}{r^2}\,u_r^2) + 2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz} + \frac{1}{r}\,u_r\,(\varepsilon_{rr} + \varepsilon_{zz}))\right) + \\
&\quad + \frac{E}{1+\nu}\,\varepsilon_{rz}^2 \,.
\end{aligned}
$$

To find the elastic energy in the deformed solid this expression can be integrated with respect to the angle φ, leading to an integral over the domain $\Omega \subset \mathbb{R}^2$. The contributions to the total energy by the volume and surface forces lead to similar expression, and finally to the total energy, similar to (29).

$$
\begin{aligned}
U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \qquad (40) \\
&= \iint_\Omega \frac{2\,\pi\,r\,E}{2\,(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,(\varepsilon_{rr}^2 + \varepsilon_{zz}^2 + \frac{1}{r^2}\,u_r^2) + 2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz} + \frac{1}{r}\,u_r\,(\varepsilon_{rr} + \varepsilon_{zz}))\right)\,dA + \\
&\quad + \iint_\Omega \frac{2\,\pi\,r\,E}{1+\nu}\,\varepsilon_{rz}^2\,dA - \iint_\Omega 2\,\pi\,r\,\vec{f}\cdot\vec{u}\,dA - \int_{\Gamma_2} 2\,\pi\,r\,\vec{g}_N\cdot\vec{u}\,ds \,.
\end{aligned}
$$

Some of the contributions are similar to the elastic energy for plane stress problems (28) or (29), i.e.

$$
W_{stress} = \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2\right) \,,
$$

but there are some more contributions. The factor r will change all expressions for the element stiffness matrices, i.e. new, slightly more complex *Octave* codes are required.

Using the Bernoulli principle this energy has to be minimized. It is this minimization problem that is solved, subject to the boundary conditions (20).

Find examples in Sections 3.14, 9.43.2, 9.44, 9.47, 9.48 and 9.53.

3 Illustrative Examples

Solving a BVP (Boundary Value Problem) or an IBVP (Initial Boundary Value Problem) with FEM (Finite Element Method) usually involves three steps:

1. Generate the domain with the mesh to be used for the problem. With this step the type of element can be selected, i.e. linear, quadratic or cubic. The type of boundary conditions are specified with the mesh.
2. Define the functions describing the problem and then apply the finite element algorithm to generate an approximate solution.
3. Visualize and analyze the obtained solution.

For all three steps FEMoctave provides tools and the following examples illustrate the procedures.

3.1 Solving elliptic problems, static heat equations

3.1.1 A symmetric problem

On a rectangle $\Omega = [0,1] \times [0,2]$ with Dirichlet boundary Γ_1 at $x = 0$ and at $y = 0$ and Neumann boundary Γ_2 at $x = 1$ and at $y = 2$ seek a solution of

$$\begin{array}{rcll} -\Delta u & = & 0.25 & \text{for} \quad (x,y) \in \Omega \\ u & = & 0 & \text{for} \quad (x,y) \in \Gamma_1 \\ \frac{\partial u}{\partial n} & = & 0 & \text{for} \quad (x,y) \in \Gamma_2 \end{array} \quad .$$

The solution is computed and displayed with the help of three commands.

- Divide the x and y axis in subintervalls of length 0.1 and generate the resulting rectangular mesh using the command `CreateMeshRect()`. Use the options `...,-1,-2,-1,-2)` to indicate the boundary conditions at the four edges in order lower, upper, left and right. In this example use the order Dirichlet, Neumann, Dirichlet, Neumann.
- Use `BVP2Dsym()` with constant coefficients to generate and solve the system of linear equation by the FEM.
- Use `FEMtrimesh()` to display the solution.

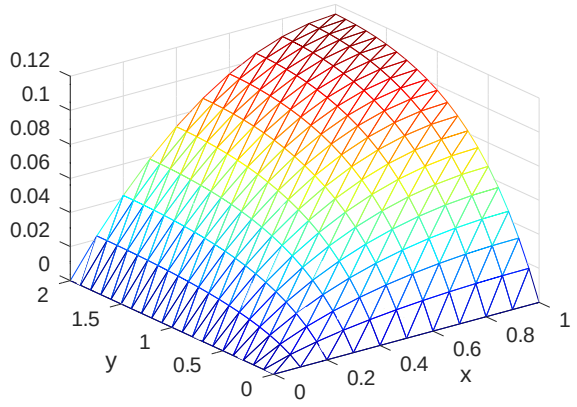


Figure 8: Solution of $-\Delta u = 0.25$ on a rectangle

LaplaceRectangle.m

```
FEMmesh = CreateMeshRect([0:0.1:1],[0:0.1:2],-1,-2,-1,-2);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment to use quadratic elements
%%FEMmesh = MeshUpgrade(FEMmesh,'cubic');      %% uncomment to use cubic elements

u = BVP2Dsym(FEMmesh,1,0,0.25,0,0,0);
figure(1); FEMtrimesh(FEMmesh,u);  xlabel('x'); ylabel('y');
```

Find the result in Figure 8. The above code is using linear elements. To use quadratic or cubic elements uncomment one of the lines with `MeshUpgrade()`.

3.1.2 Laplace equation in cylindrical coordinates

The Laplace operator in cylindrical coordinates is given by

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho \frac{\partial u}{\partial \rho}\right) + \frac{1}{\rho^2}\frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2}\,.$$

Assuming that the solution is independent on the angle θ, then the Laplace equation $-\Delta u(\rho, z) + b_0(\rho, z) = f(\rho, z)$ is given by

$$-\frac{\partial}{\partial \rho}\left(\rho \frac{\partial u}{\partial \rho}\right) - \frac{\partial}{\partial z}\left(\rho \frac{\partial u}{\partial z}\right) + \rho\, b_0(\rho, z) = \rho\, f(\rho, z)\,.$$

Thus it is in the form of equation (2), with $x = \rho$ and $y = z$. As an example consider $b_0(\rho, z) = 10$ and $f(\rho, z) = 2\,z$. The domain Ω to be examined is given by $0 \le \rho \le 2$ and $-1 \le z \le 2$ and the boundary conditions are

$$\begin{aligned} \frac{\partial u(0,z)}{\partial \rho} &= 0 \quad \text{symmetry for } -1 < z < 2 \\ \rho \frac{\partial u(2,z)}{\partial \rho} &= -1 \quad \text{flux out of domain for } -1 < z < 2 \\ u(\rho,-1) = u(\rho,2) &= 0 \quad \text{given value for } 0 < \rho < 2 \quad . \end{aligned}$$

Since the coefficient functions in (2) are not constants define these functions in *Octave* and then use `BVP2Dsym()` to solve the problem. Observe that both Neumann boundary conditions are described by the same function $g_2(\rho, z) = \frac{-\rho}{2}$, since $g_2(0, z) = 0$ and $g_2(2, z) = -1$. The code is shown below and find the result in Figure 9.

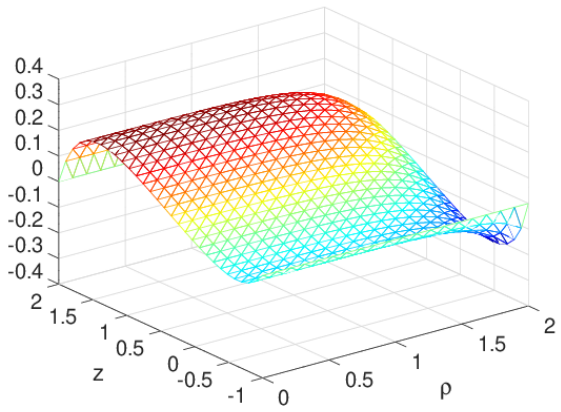


Figure 9: Solution of the Laplace equation in cylindrical coordinates

LaplaceCylindrical.m

```
FEMmesh = CreateMeshRect(linspace(0,2,20),linspace(-1,2,30),-1,-1,-2,-2);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment to use quadratic elements
function res = f(rz,dummy)  res = rz(:,1)*2.*rz(:,2); endfunction
function res = b0(rz,dummy) res = 10*rz(:,1);         endfunction
```

```
function res = a(rz,dummy)   res = rz(:,1);            endfunction
function res = g2(rz)        res = -1*rz(:,1)/2;       endfunction

u = BVP2Dsym(FEMmesh,'a','b0','f',0,'g2',0);
FEMtrimesh(FEMmesh,u); xlabel('\rho'); ylabel('z');
```

3.1.3 Diffusion on an L-shaped domain

Examine a BVP on an L-shaped domain, as created in Section 4.1. The equation to be solved is

$$\begin{aligned} -\Delta u &= 1 \qquad \text{for} \quad (x,y) \in \Omega \\ \tfrac{\partial u}{\partial n} &= -2\,u \qquad \text{for} \quad (x,y) \in \Gamma \end{aligned} \quad .$$

For this problem there is no Dirichlet condition and it is solved in three steps.

- Generate the L-shaped domain with the help of `CreateMeshTriangle()`.
- Solve the equations with `BVP2Dsym()`.
- Display the result with `FEMtrimesh()` and `FEMtricontour()`.
- The code below uses linear elements. Uncommenting the line with `MeshUpgrade()` will solve the same problem using second or third order elements.

Find the code below and the result in Figure 10.

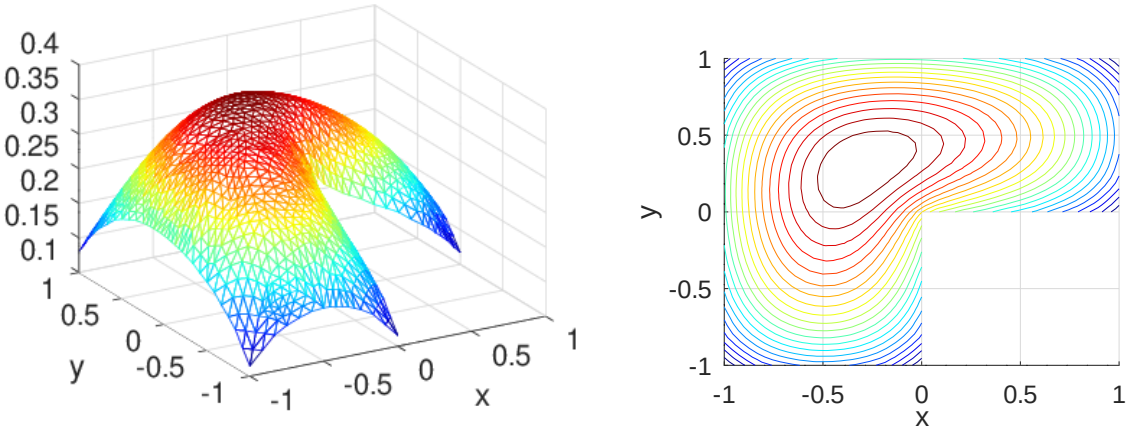


Figure 10: Solution of a diffusion problem on a L-shaped domain

DiffusionLshape.m

```
nodes = [0,0,-2;1,0,-2;1,1,-2;-1,1,-2;-1,-1,-2;0,-1,-2];
FEMmesh = CreateMeshTriangle('Ldomain',nodes,0.02);
FEMmesh = MeshUpgrade(FEMmesh,'cubic');  %% uncomment to use cubic elements

u = BVP2Dsym(FEMmesh,1,0,1,0,0,-2);
figure(1); FEMtrimesh(FEMmesh,u);    xlabel('x'); ylabel('y'); view(-30,30)
figure(2); clf; FEMtricontour(FEMmesh,u); xlabel('x'); ylabel('y');
```

3.1.4 A diffusion convection problem

Examine a steady state heat problem on the square $\Omega = [0,2] \times [0,2]$ with constant heating $f(x,y) = +0.1$ and a strong convection in x–direction ($b_x(x,y) = 10$) and a weaker convection in y–direction ($b_y(x,y) = 5$). This leads to the PDE

$$-\Delta u + 10\,\frac{\partial u}{\partial x} + 5\,\frac{\partial u}{\partial y} = 0.1\,.$$

The temperature on all of the boundary vanishes. This is a problem of type (1). Solve the BVP with the code below and find the resulting level curves of the temperature in Figure 11.

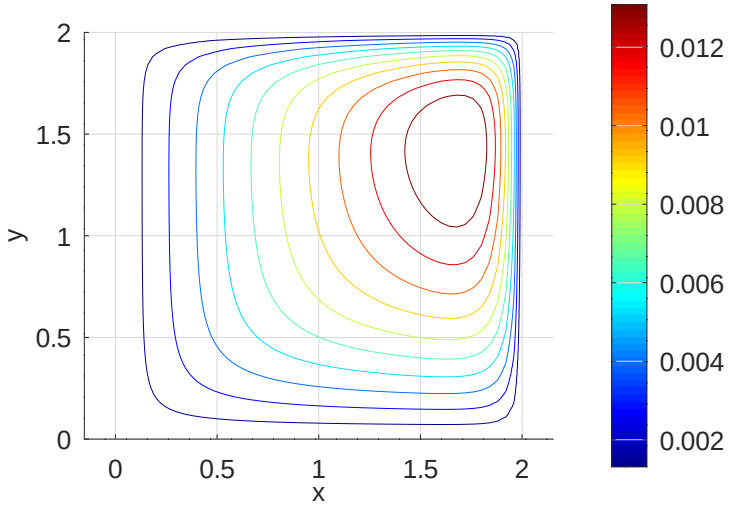


Figure 11: Solution of a diffusion convection problem

DiffusionConvection.m

```
FEMmesh = CreateMeshRect(linspace(0,2,51),linspace(0,2,51),-1,-1,-1,-1);
u = BVP2D(FEMmesh,1,0,10,5,0.1,0,0,0);

figure(1); clf; FEMtricontour(FEMmesh,u,10);
           colorbar(); xlabel('x'); ylabel('y'); grid on
```

The above code uses elements of order 1. To use elements of order 2 on a similar mesh one can first generate a mesh with linear elements and then use `MeshUpgrade()` to generate a mesh with elements of order 2. Convert the mesh back to linear elements, but with the identical nodes, i.e. use `MeshQuad2Linear()` and then display.

DiffusionConvection.m

```
FEMmesh = CreateMeshRect(linspace(0,2,26),linspace(0,2,26),-1,-1,-1,-1);
FEMmesh = MeshUpgrade(FEMmesh, 'quadratic');  %% make a mesh with elements of order 2
u = BVP2D(FEMmesh,1,0,10,5,0.1,0,0,0);
FEMmesh = MeshQuad2Linear(FEMmesh);  %% convert to identical mesh with linear elements

figure(1); clf; FEMtricontour(FEMmesh,u,10);
           colorbar(); xlabel('x'); ylabel('y'); grid on
```

3.1.5 Solving nonisotropic problems

A static heat equation of the form

$$-\nabla\left(\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\nabla u\right) = f(x,y) = \begin{cases} 10 & \text{for } x^2+y^2 < 0.2^2 \\ 0 & \text{for } x^2+y^2 > 0.2^2 \end{cases}$$

describes a situation with with a larger conduction coefficient in the $+45°$ (or $-135°$) direction. The eigenvalues of the matrix are $+1$ (eigenvector in $-45°$ or $+135°$ direction) and $+3$ (eigenvector in the $+45°$ direction). Figure 12 illustrates the effect with the resulting temperature u.

HeatNonisotropic.m

```
R = 2; N = 50; alpha = linspace(0,2*pi*N/(N-1),N)';
FEMmesh = CreateMeshTriangle('circle', ...
      [R*cos(alpha),R*sin(alpha),-ones(size(alpha))],0.1/4 );
```

```
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = f(xy)
  x = xy(:,1); y = xy(:,2); res = 10*((x.^2+y.^2)<0.2^2);;
endfunction

a = [2 2 1]; %% eigenvalues 1 and 3
u = BVP2D(FEMmesh,a,0,0,0,'f',0,0,0);

figure(1); FEMtrimesh(FEMmesh,u);           xlabel('x') ; ylabel('y'); zlabel('u')
figure(2); FEMtricontour(FEMmesh,u);   xlabel('x') ; ylabel('y'); axis equal
```

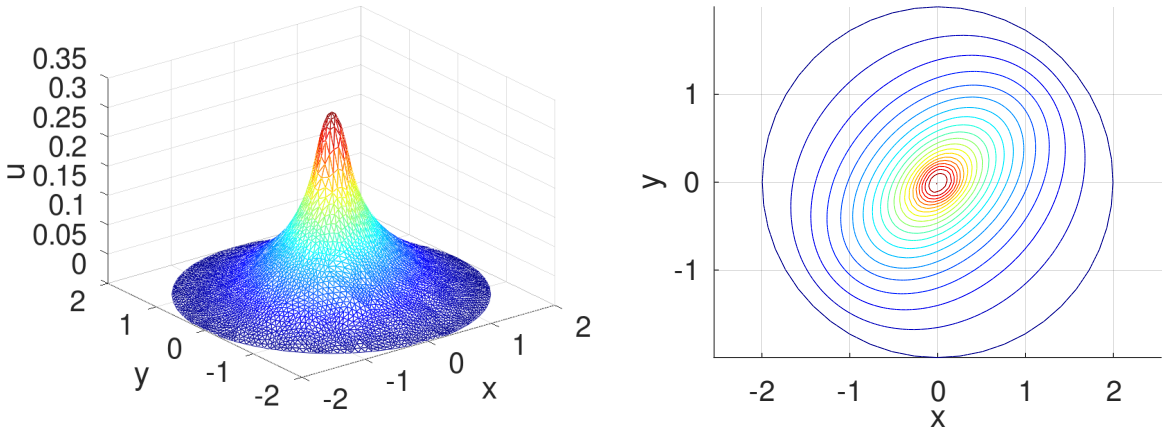


Figure 12: The temperature u for a steady state heat equation with direction dependent conduction coefficients

The energy flux in this case is given by the vector field

$$\vec{F} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \nabla u = \begin{pmatrix} 2\,\frac{\partial u}{\partial x} + 1\,\frac{\partial u}{\partial y} \\ 1\,\frac{\partial u}{\partial x} + 2\,\frac{\partial u}{\partial y} \end{pmatrix} .$$

With `FEMgriddata()` evaluate the gradient ∇u on a fine grid and multiply by the matrix. Then the *Octave* command `streamline()` will display the flow lines for the energy flow. Find the result in Figure 13(a). By evaluating the flux

$$\text{flux}(\phi) = \langle \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \end{pmatrix}, \vec{F} \rangle$$

along a circle very close to the boundary determine the energy flux through the boundary as function of the angle. Find the result in Figure 13(b). It is clearly visible that the higher thermal conductivity in the $+45°$ direction leads to the energy mainly flowing this these directions.

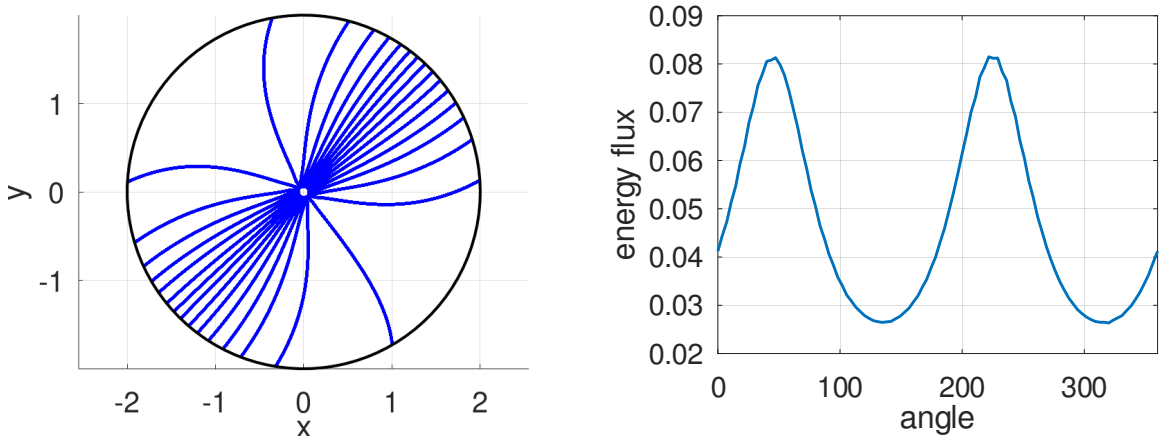
HeatNonisotropic.m

```
%%% generate flow lines
N = 501;
[x,y] = meshgrid(linspace(-2,2,N));
[ui,uix,uiy] = FEMgriddata(FEMmesh,u,x,y);
%%% determine energy flux
uix2 = 2*uix + 1*uiy; uiy2 = 2*uiy + 1*uix;
uix = uix2; uiy = uiy2;

r0 = 0.05; alpha = linspace(0,2*pi,31); alpha = alpha(1:end-1);
sx = r0*cos(alpha); sy = r0*sin(alpha);
beta = linspace(0,2*pi); xR = R*cos(beta); yR = R*sin(beta);
figure(3);clf; streamline(x,y,-uix,-uiy,sx,sy,[0.01,30000]);
          hold on; plot (xR,yR,'k'); hold off
          xlabel('x'); ylabel('y'); axis equal
```

```
%% evaluate flow through boundary
beta = linspace(0,2*pi); xR = 0.95*R*cos(beta); yR = 0.95*R*sin(beta);
[uR,uRx,uRy] = FEMgriddata(FEMmesh,u,xR,yR);

figure(4); plot(beta*180/pi,-uRx.*cos(beta)-uRy.*sin(beta))
           xlabel('angle'); ylabel('energy flux'); xlim tight
```



(a) energy flow lines

(b) energy flux through the boundary

Figure 13: The flow lines and the energy flux through the boundary

3.1.6 Solving a nonlinear problem

On a triangular domain $\Omega \subset \mathbb{R}^2$ examine the (purely academic) nonlinear boundary value problem

$$\begin{array}{rcll} -\Delta u & = & \sin(u) & \text{in } \Omega \\ u & = & 1 & \text{on } \Gamma_1 \\ \frac{\partial}{\partial \vec{n}}\, u & = & 1-u & \text{on } \Gamma_2 \end{array} \quad ,$$

where Γ_2 is the edge connecting the points $(1,0)$ and $(0.5,1)$. The code below uses the command `BVP2DNL()` and requires the function $f(x,y,u) = \sin(u)$ and its derivative $\frac{\partial}{\partial u} f(x,y,u) = \cos(u)$. Observing the iterations confirms a quadratic convergence to the solution displayed in Figure 14.

DemoBVP2DNL.m

```
Mesh = CreateMeshTriangle('triangle',[-1,0,-1;1,0,-2;0.5,1,-1],0.01);
Mesh = MeshUpgrade(Mesh,'quadratic');
a = 1; b0 = 0; bx = 0; by = 0; gD = 1; u0 = 1; c = 1;
f = {@(xy,u)c*sin(u);   @(xy,u)c*cos(u)};

u = BVP2DNL(Mesh,a,b0,bx,by,f,gD,1,-1,u0,'tol',1e-10,'Display','iter');
figure(1); FEMtrimesh(Mesh,u);
           xlabel('x'); ylabel('y'); zlabel('u'); view([-80,50])
figure(2); FEMtricontour(Mesh,u); xlabel('x'); ylabel('y'); colorbar
-->
iteration 1, RMS(correction) = 1.165640e-03
iteration 2, RMS(correction) = 5.710658e-08
iteration 3, RMS(correction) = 6.267914e-16
```

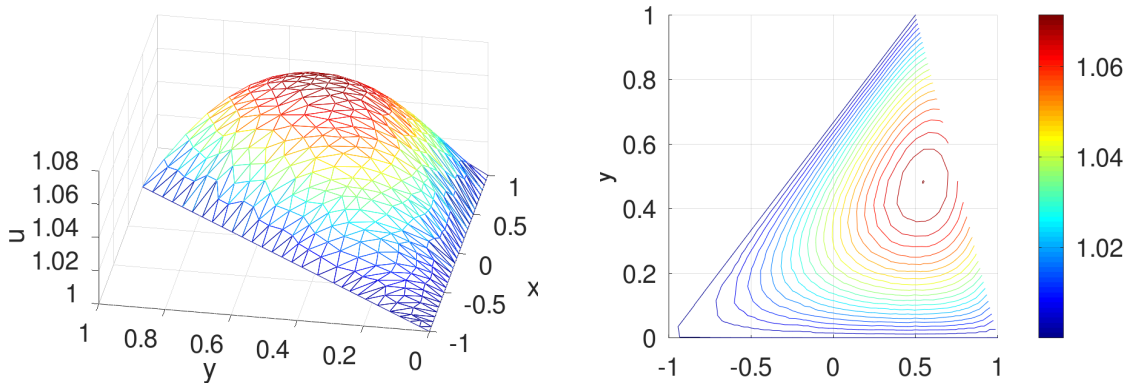


Figure 14: The solution of a nonlinear 2D boundary value problem

3.2 Solving eigenvalue problems

As a first eigenvalue problem compute the eigenvalues and eigenfunctions of the Laplace operator on the unit disc with Dirichlet boundary conditions, i.e. determine a scalar λ and nontrivial function u such that

$$-\Delta u = \lambda u \quad \text{on unit disc and} \quad u = 0 \quad \text{on the boundary.}$$

The goal is to compute the four smallest eigenvalues and display the fourth eigenfunction. Proceed in three steps.

- Use `CreateTriangleMesh()` to generate the mesh on the unit disc.
- Use `BVP2Deig()` with constant coefficients to generate and solve the eigensystem.
- Use `FEMtrimesh()` to display the fourth eigenfunction. Find the result in Figure 15.
- To use second order element, use `MeshUpgrade()`.

The computed eigenvalues are $\lambda_1 \approx 5.7857$, $\lambda_2 = \lambda_3 \approx 14.6959$ and $\lambda_4 \approx 26.4169$. These values coincide nicely with the squares of the first zeros of the Bessel functions $J_0(r)$, $J_1(r)$ and $J_2(r)$, the values of the exact problem.

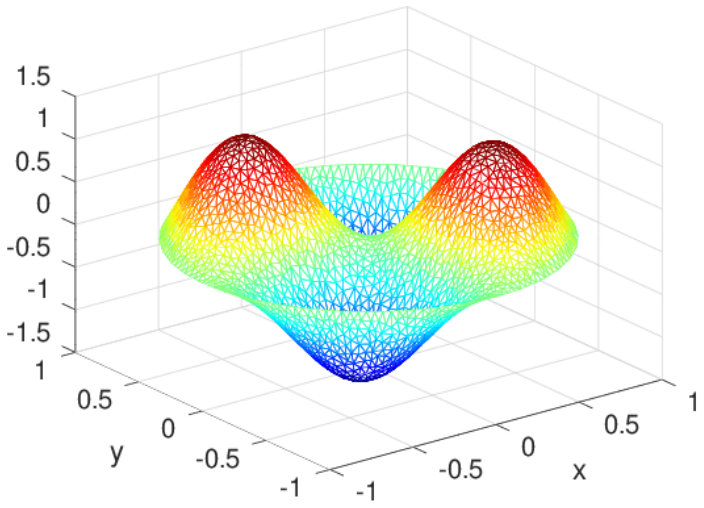


Figure 15: The fourth eigenfunction of $\Delta u = \lambda\, u$ on a disc

EigenvaluesDisc.m

```
xM = 0; yM = 0; R = 1; N = 160; alpha = linspace(0,N/(N+1)*2*pi,N)';
xy = [xM+R*cos(alpha),yM+R*sin(alpha),-ones(size(alpha))];

FEMmesh = CreateMeshTriangle('circle',xy,0.0005);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
%%%%%%% solve the eigenvalue problem, show the eigenvalues
%%[la,ve]           = BVP2Deig(FEMmesh,1,0,1,0,4);
[la,ve,errorbound] = BVP2Deig(FEMmesh,1,0,1,0,4);
eigenvalues = la
errorbound
exact_values = [fsolve(@(x)besselj(0,x),2.3), fsolve(@(x)besselj(1,x),3.8),...
                fsolve(@(x)besselj(2,x),5)].^2
figure(1); FEMtrimesh(FEMmesh,ve(:,4));  xlabel('x'); ylabel('y');
```

The result shows the first 4 eigenvalues and their corresponding error bounds. The error bounds of 10^{-28} for the first eigenvalue is not to be taken too seriously, it just means *accurate up to machine precision* as eigenvalue of the global stiffness matrix. Observe that these are the eigenvalues of the FEM approximation to the boundary value problem. They are close to the eigenvalues of the continuous problem, i.e. the squares of the zeros of the Bessel functions.

```
eigenvalues =    5.7857
                14.6959
                14.6961
                26.4169

errorbound =   2.5479e-12   1.6604e-28
               2.9179e-12   7.0763e-16
               3.2020e-12   7.2782e-15
               3.5589e-12   2.3726e-28

exact_values =  5.7832   14.6820   26.3746
```

In cylindrical coordinates the Laplace operator is given by

$$\Delta u(r,\theta) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\,\frac{\partial\, u(r,\theta)}{\partial r}\right) + \frac{1}{r^2}\,\frac{\partial\, u(r,\theta)}{\partial \theta^2}\ .$$

For purely radial solutions the eigenvalue problem on a disk of radius R leads to

$$\begin{aligned} -(r\,u'(r))' &= \lambda\, r\, u(r) \qquad \text{for} \quad 0 < r < R \\ u'(0) &= 0 \\ u(R) &= 0 \end{aligned} \quad .$$

This equation can be solved with the help of `BVP1Deig()`. Find the graphical result for the first four radial eigenfunctions in Figure 16(a). The exact values are the squares of the zeros[7] of the Bessel function $J_0(r)$. Obviously the first eigenvalue coincides with the above 2D approach. The other eigenvalues differ, since the above code takes angular dependence into account, while `BVP1Deig()` examines radial dependence exclusively.

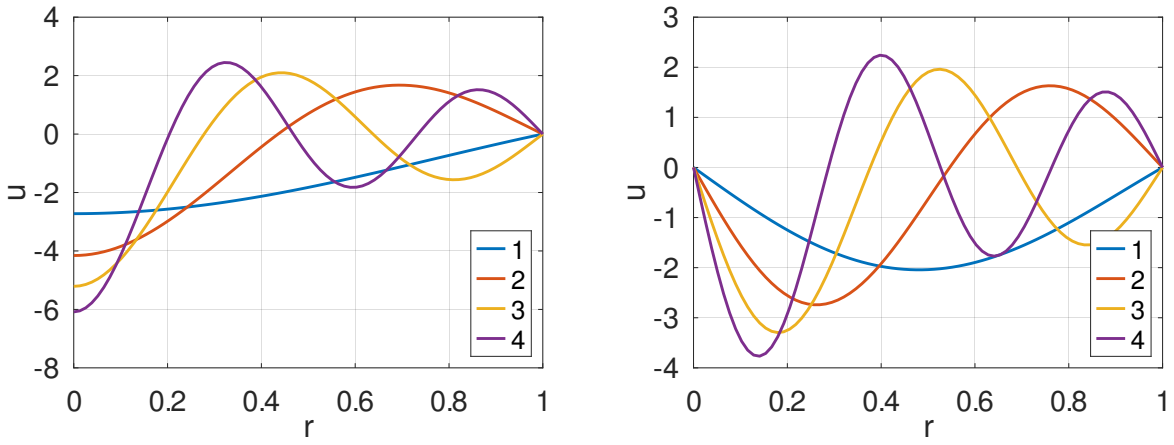
EigenvaluesDisk1D.m

```
R = 1; N = 40;  interval = linspace(0,R,N)';
f_r = @(r)r;
[r,eVal,eVec] = BVP1Deig(interval,f_r,0,0,f_r,[0,0],0,4);

figure(1); plot(r,eVec); xlabel('r'); ylabel('u');
           legend('1','2','3','4','location','southeast')
eVal_FEM     = eVal'
exact_values = [fsolve(@(x)besselj(0,x),2.3),fsolve(@(x)besselj(0,x),5.4),...
                fsolve(@(x)besselj(0,x),9),fsolve(@(x)besselj(0,x),12)].^2
```

[7] The function `fsolve()` is used to determine the zeros. To find approximate values use a plot of the function `besselj(0,x)`.

```
-->
eVal_FEM =        5.7832    30.4713    74.8872   139.0416
exact_values =    5.7832    30.4713    74.8866   139.0403
```



(a) pure radial dependence $u(r)$ (b) with angular dependence $u(r)\,\sin(\theta)$

Figure 16: The first four radial eigenmodes of the Laplace operator on a disk of radius 1, for $u(r)$ and $u(r)\,\sin(\theta)$

To obtain the eigenfunctions with angular dependence use the tool separation of variables. For a function $f(r,\theta) = u(r)\,\sin(n\,\theta+\delta)$ obtain

$$\begin{aligned}\Delta f(r,\theta) &= \frac{1}{r}\frac{\partial}{\partial r}\,(r\,\frac{\partial\, f(r,\theta)}{\partial r}) + \frac{1}{r^2}\,\frac{\partial\, f(r,\theta)}{\partial\theta^2} \\ &= \left(\frac{1}{r}\,\frac{\partial}{\partial r}\,(r\,\frac{\partial\, u(r}{\partial r}) - n^2\,\frac{1}{r^2}\,u(r)\right)\,\sin(n\,\theta+\delta)\;.\end{aligned}$$

Thus the corresponding eigenvalue problem is

$$\begin{aligned}-(r\,u'(r))' + \tfrac{1}{r}\,n^2\,u(r) &= \lambda\,r\,u(r) \qquad \text{for} \quad 0<r<R \\ u'(0) &= 0 \\ u(R) &= 0\end{aligned}\quad .$$

Find the result for $n=1$ of `BVP1Deig()` in Figure 16(b). The exact values are the squares of the zeros of the Bessel function $J_n(r)$.

EigenvaluesDisk1D.m

```
n = 1; n2_r = @(r)n^2./r;
[r,eVal2,eVec2] = BVP1Deig(interval,f_r,0,n2_r,f_r,[0,0],0,4);

figure(2); plot(r,eVec2); xlabel('r'); ylabel('u');
           legend('1','2','3','4','location','southeast')
eVal2_FEM = eVal2'
exact_values2 = [fsolve(@(x)besselj(n,x),4),fsolve(@(x)besselj(n,x),7),...
                 fsolve(@(x)besselj(n,x),10),fsolve(@(x)besselj(n,x),13)].^2
-->
eVal2_FEM     =  14.682    49.219   103.500   177.523
exact_values2 =  14.682    49.218   103.499   177.521
```

The eigenvalues $\lambda_{n,m}$ of the complex boundary value problem

$$\begin{aligned}-\Delta\,u(x,y) + i\,u(x,y) &= \lambda\,u(x,y) \qquad \text{for } 0<x,y<\pi \\ u(x,y) &= 0 \qquad\qquad\qquad\quad \text{on the boundary}\end{aligned}$$

are given by $\lambda_{n,m} = n^2 + m^2 + i$ for $n, m \in \mathbb{N}$ with eigenfunctions $u_{n,m}(x, y) = \sin(x)\,\sin(y)$. The code below and the resulting Figure 17 confirm this result. Observe that with $u(x, y)$ any function $c\,u(x, y)$ for a complex constant $c \in \mathbb{C}$ is an eigenfunction too. As consequence observe in Figure 17 that $\mathrm{Im}(u) = \alpha\ \mathrm{Re}(u)$ for some constant factor $\alpha \in \mathbb{R}$.

EigenvaluesComplex.m

```
FEMmesh = CreateMeshRect(linspace(0,pi,21),linspace(0,pi,21),-1,-1,-1,-1);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

%%%%%%% solve the eigenvalue problem, show the eigenvalues
[la,ve] = BVP2Deig(FEMmesh,1,i,1,0,7,"type","complex");
eigenvalues = la
figure(1); FEMtrimesh(FEMmesh,real(ve(:,3)));  xlabel("x"); ylabel("y"); zlabel("real(u)")
figure(2); FEMtrimesh(FEMmesh,imag(ve(:,3)));  xlabel("x"); ylabel("y"); zlabel("imag(u)")
-->
eigenvalues =     2.0000 +  1.0000i
                  5.0001 +  1.0000i
                  5.0002 +  1.0000i
                  8.0007 +  1.0000i
                 10.0011 +  1.0000i
                 10.0011 +  1.0000i
                 13.0020 +  1.0000i
```

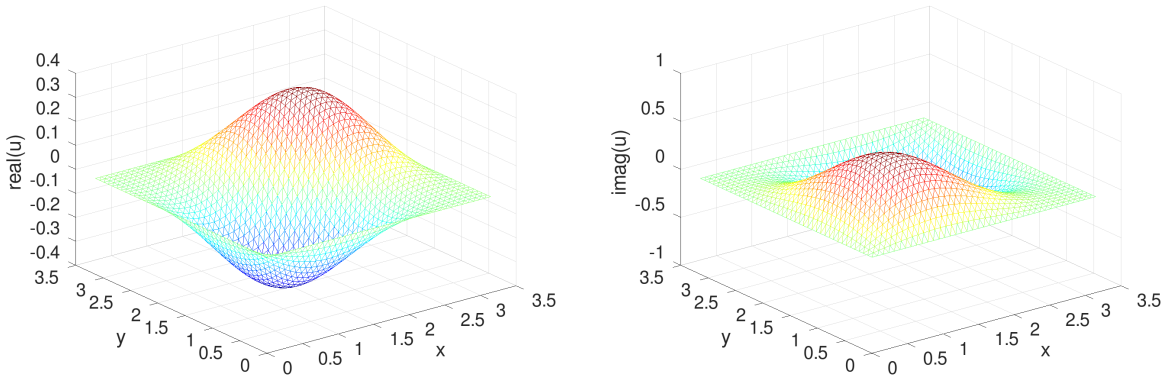


Figure 17: An eigen function of a complex eigenvalue problem

3.3 Solving parabolic problems, dynamic heat equations

As an example solve the dynamic heat equation

$$\frac{\partial\, u}{\partial t} - \Delta u + 10\,\frac{\partial\, u}{\partial x} + 5\,\frac{\partial\, u}{\partial y} = 0.1 \quad \text{for} \quad 0 < x, y < 2$$

with zero Dirichlet boundary conditions and the initial temperature

$$u(0, x, y) = u_0(x, y) = 0.005\, x\,(2 - x)^2\, y\,(2 - y)\,.$$

The solution is computed at 7 equally spaced times t_i between 0 and 0.1. In-between 10 steps are taken, but the solution is not returned. Find the result of the code below in Figure 18. At time 0 the maximal value is attained at $(x, y) = (\frac{2}{3}, 1)$. The convection term $+10\,\frac{\partial\, u}{\partial x} + 5\,\frac{\partial\, u}{\partial y}$ then moves the point of maximal temperature to the upper right section of the square. For large times t the solution will converge to the steady state solution shown in Figure 11 in Section 3.1.4.

HeatDynamic.m

```
%% a dynamic heat problem
N = 25; %%grid size 2N+1 by 2N+1, generate the mesh
if 0  %% first order elements
  FEMmesh = CreateMeshRect(linspace(0,2,2*N+1),linspace(0,2,2*N+1),-1,-1,-1,-1);
else  %% second order elements
  FEMmesh = CreateMeshRect(linspace(0,2,N+1),linspace(0,2,N+1),-1,-1,-1,-1);
  FEMmesh = MeshUpgrade(FEMmesh);
endif
x = FEMmesh.nodes(:,1); y = FEMmesh.nodes(:,2);
%% setup and solve the initial boundary value problem
m = 1; a = 1; b0 = 0; bx = 10; by = 5; f = 0.1; gD = 0; gN1 = 0; gN2 = 0;
t0 = 0; tend = 0.1 ; steps = [6,10];
u0 = zeros(length(FEMmesh.nodes),1);
u0 = 0.005*(2-x).^2.*x.*y.*(2-y);
tic()
[u_dyn,t] = IBVP2D(FEMmesh,m,a,b0,bx,by,f,gD,gN1,gN2,u0,t0,tend,steps,'solver','RK');
toc()
figure(1); FEMtrimesh(FEMmesh,u_dyn(:,end));  xlabel('x'); ylabel('y')

%% show the animation on screen
u_max = max(u_dyn(:));
for t_ii = 1:length(t)
  figure(2); FEMtrimesh(FEMmesh,u_dyn(:,t_ii));  xlabel('x'); ylabel('y')
             caxis([0,u_max]);  axis([0 2 0 2 0 u_max])
  drawnow();
  figure(3); FEMtricontour(FEMmesh,u_dyn(:,t_ii),linspace(0,0.99*u_max,11))
             xlabel('x'); ylabel('y');  caxis([0,u_max]);
  drawnow();
  pause(0.4)
endfor
```

3.4 Solving semilinear parabolic problems, dynamic heat equations

The right hand side of a heat equation might depend on the solution u too, leading to a semilinear problem. As example consider

$$\begin{aligned} \tfrac{\partial}{\partial t}\, u - \Delta u &= u\,(1-u) + \exp(-2\,t)\,\sin^2(x)\,\sin^2(y) \quad \text{for } 0 < x, y < \pi, \quad t > 0 \\ u &= 0 \qquad \text{on the boundary} \\ u(x,y,0) &= \sin(x)\,\sin(y) \end{aligned}$$

This initial boundary value problem can be solved by the command `IBVP2DNL()`, as shown in the code below. The same problem is used in Section 5.10 to illustrate the converge on the time stepping algorithms.

HeatSemilinear.m

```
h  = 0.25; %%typical radius of triangles
nodes = [0 0 -1; pi 0, -1; pi pi -1; 0 pi -1];
FEMmesh = CreateMeshTriangle("test",nodes,h^2/2);

f = @(xy,t,u) u.*(1-u) + exp(-2*t)*sin(xy(:,1)).^2 .* sin(xy(:,2)).^2;
t_end = 1;  Steps = [5,10];
u0 = @(xy)sin(xy(:,1)) .* sin(xy(:,2));
[u_dyn,t] = IBVP2DNL(FEMmesh,1,1,0,0,0,f,0,0,0,u0,0,t_end,Steps);

figure(1); FEMtrimesh(FEMmesh,u_dyn(:,end)); xlabel('x'); ylabel('y'); title('solution')
```

3.5 Solving hyperbolic problems, wave equations

As an example solve the wave equation

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = 0 \quad \text{for} \quad x^2 + y^2 < 6$$

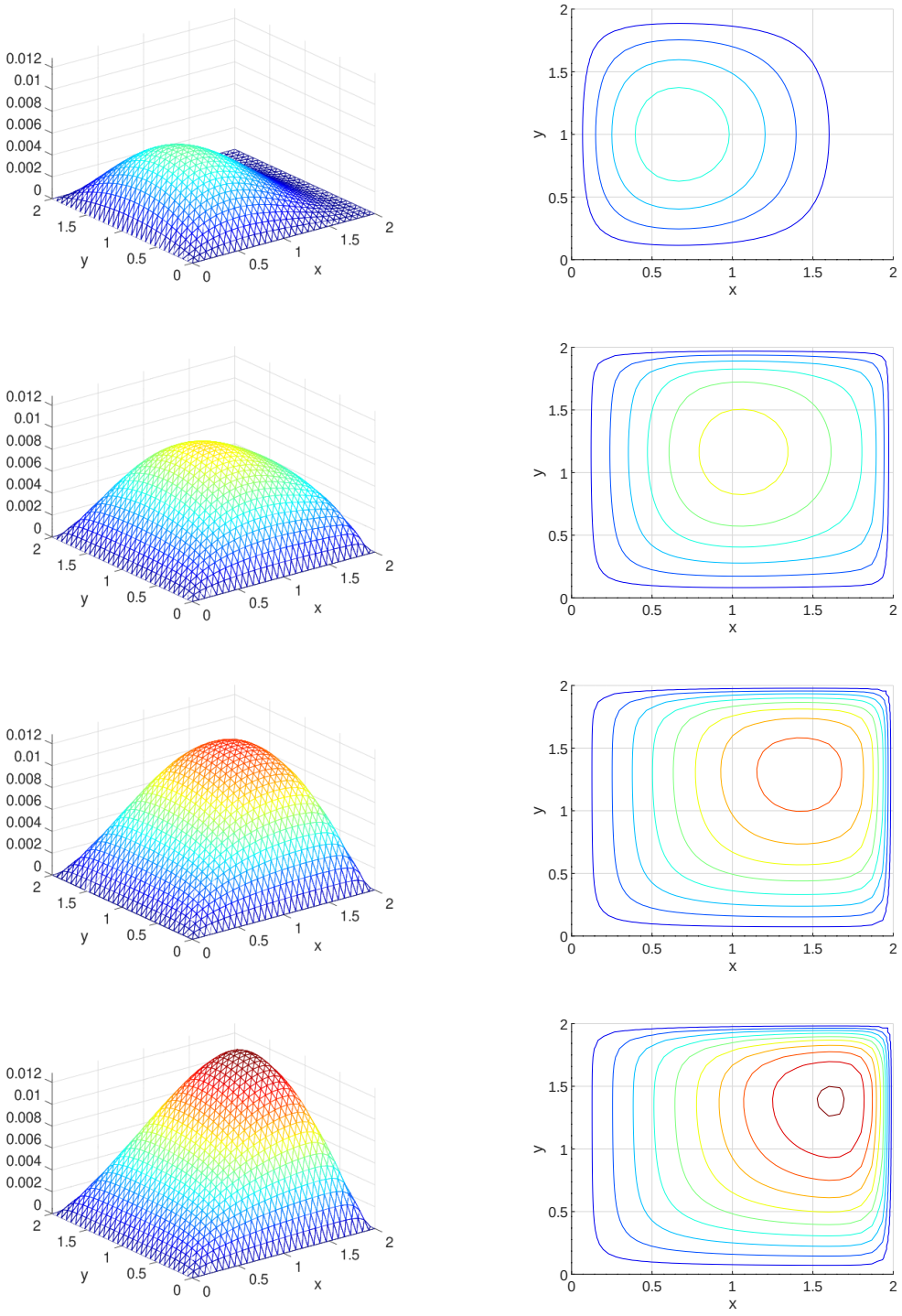


Figure 18: Solution of a dynamic heat equation

with zero Dirichlet boundary conditions, the initial displacement

$$u(0,x,y) = u_0(x,y) = 0.1\,\exp(-(x-1)^2 - y^2)\,(R^2 - x^2 - y^2)/R^2$$

and zero initial velocity $v_0 = 0$. This assures compatible initial values, i.e. the boundary condition is satisfied at time $t = 0$. The solution is computed at 15 equally spaced times t_i between 0 and 7. In-between 30 steps are taken, but the solution is not returned. The solution is returned at 15 times, leading to Figure 19. The initial hump is traveling towards the boundary of the circle with speed 1, where it is reflected. More examples are shown in Sections 9.2 and 9.18.

WaveDynamic.m

```
%% generate a circle
alpha = linspace(0,2*pi,101)'; alpha = alpha(1:end-1); R = 6;
xy = [R*cos(alpha),R*sin(alpha),-ones(size(alpha))];
if 1  %% linear elements
  FEMmesh = CreateMeshTriangle('Circle',xy,0.03);
else  %% quadratic elements
  FEMmesh = CreateMeshTriangle('Circle',xy,4*0.03);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
endif

x = FEMmesh.nodes(:,1); y = FEMmesh.nodes(:,2);
v0 = zeros(size(x));
u0 = 0.1*exp(-1*((x-1).^2+y.^2)); u0 = u0.*(R^2-x.^2-y.^2)/R^2;
%% setup and solve the initial boundary value problem
m=1; d=0; a=1; b0=0; bx=0; by=0; f=0; gD=0; gN1=0; gN2=0;
t0=0; tend=7 ; steps=[14,30];
tic();
[u_dyn,t] = I2BVP2D(FEMmesh,m,d,a,b0,bx,by,f,gD,gN1,gN2,u0,v0,t0,tend,steps);
toc()

figure(1) %% show the animation on screen
for t_ii = 1:length(t)
  FEMtrimesh(FEMmesh,u_dyn(:,t_ii))
  xlabel('x'); ylabel('y'); axis([-R R -R R -0.05 0.05])
  caxis([-0.05 0.05]);      text(4,-2,0.04,sprintf('t=%2.1f',t(t_ii)))
  drawnow();  pause(0.3)
endfor
-->
Elapsed time is 0.93231 seconds.
```

3.6 Solving semilinear hyperbolic problems, wave equations

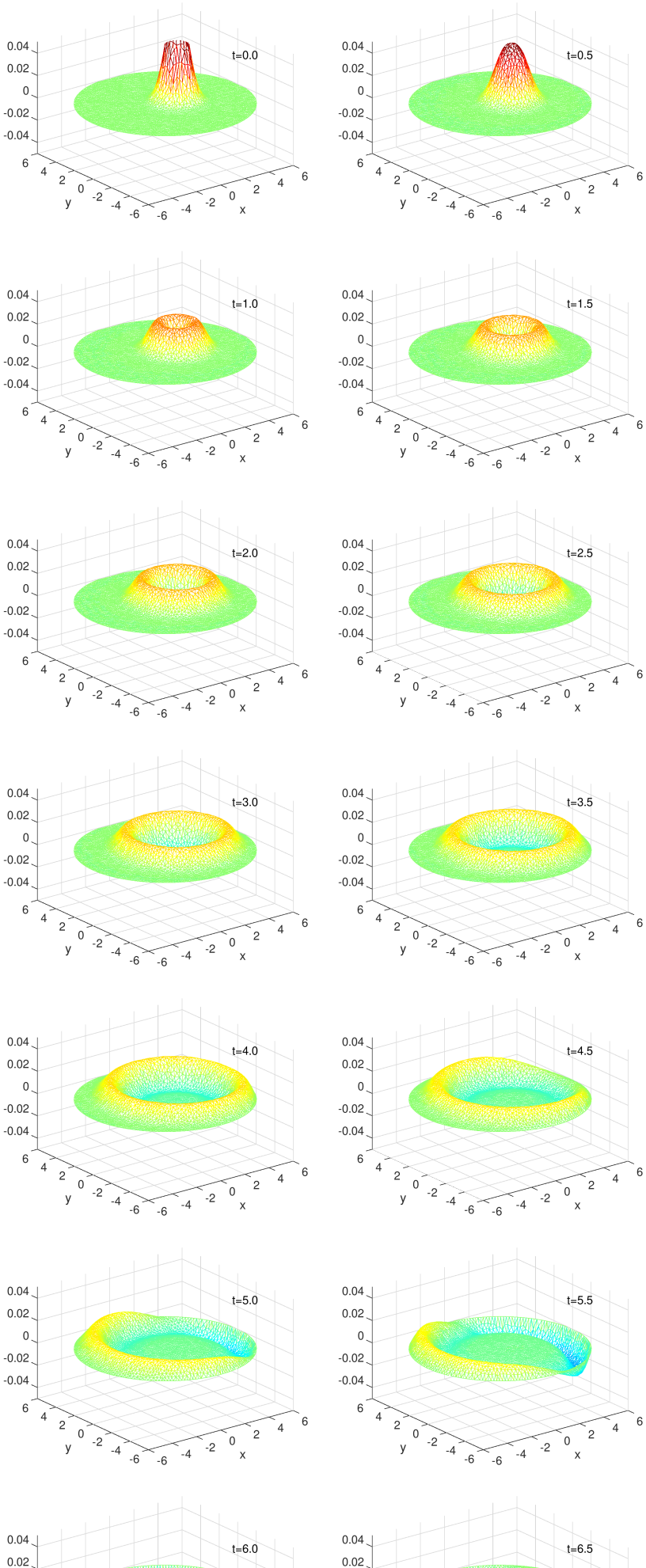
As an example for a semilinear wave equation of type (10) examine

$$\begin{aligned}
\tfrac{\partial^2}{\partial t^2}\,u + 2\,\tfrac{\partial}{\partial t}\,u - \Delta u &= \tfrac{1}{10}\,\cos(u) + 5\,x - \tfrac{1}{4}\,y && \text{for} \quad 0<x<1,\ 0<y<2,\ t>0\\
u &= 1 && \text{for} \quad 0<x<1,\ y=0 \text{ and } y=2 \text{ and } t>0\\
\tfrac{\partial}{\partial x}\,u &= 0 && \text{for} \quad 0<y<2,\ x=0 \text{ and } x=1 \text{ and } t>0\\
u(x,y,0) &= \exp(-2\,y^2) + \exp(-2\,(2-y)^2) && \text{on} \quad 0<x<1,\ 0<y<2\\
\tfrac{\partial}{\partial t}\,u(x,y,0) &= 0 && \text{on} \quad 0<x<1,\ 0<y<2
\end{aligned} \tag{41}$$

In Figure 20 fit the results of the code `DemoI2BVP2DNL.m` of the solution at time $t = 0.75$. The code allows to generate a short animation too.

DemoI2BVP2DNL.m

```
Wx = 1; Wy = 2; Nx = 21; Ny = 31;
Mesh = CreateMeshRect(linspace(0,Wx,Nx),linspace(0,Wy,Ny),-1,-1,-2,-2);
Mesh = MeshUpgrade(Mesh,'quadratic');
u0 = @(xy)exp(-2*xy(:,2).^2) + exp(-2*(Wy-xy(:,2)).^2);
Tend = 0.75;
```


t=0.0
t=0.5
t=1.0
t=1.5
t=2.0
t=2.5
t=3.0
t=3.5
t=4.0
t=4.5
t=5.0
t=5.5
t=6.0
t=6.5
0.04
0.02
0
-0.02
-0.04
x
y

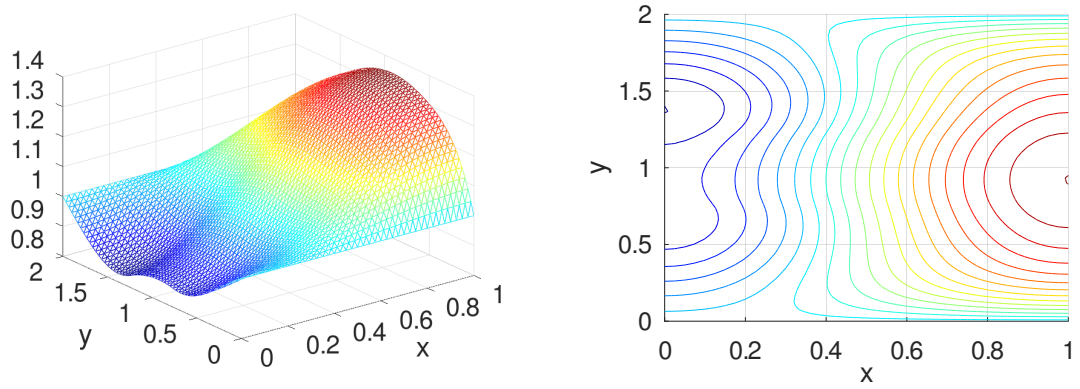


Figure 20: The final solution and its contour lines for a semilinear wave equation

```
f = @(xy,t,u)0.1*cos(u)+5*xy(:,1)-0.25*xy(:,2);
[u,t] = I2BVP2DNL(Mesh,1,1,1,0 ,0 ,0 ,f,1 ,0  ,0  ,u0,0 ,0 ,Tend,[26,50]);

figure(1); FEMtrimesh(Mesh,u(:,1))
           xlabel('x'); ylabel('y')
figure(2); FEMtrimesh(Mesh,u(:,end))
           xlabel('x'); ylabel('y')
figure(3); FEMtricontour(Mesh,u(:,end))
           xlabel('x'); ylabel('y')

if 0  %% set to 1 for an animation
  for ii = 1:length(t)
    figure(99); FEMtrimesh(Mesh,u(:,ii));
    xlabel('x'); ylabel('y'); zlim([0,1.5])
    drawnow(); pause(0.2)
  endfor
endif
```

3.7 Solving 1D steady state boundary value problems

To solve the BVP

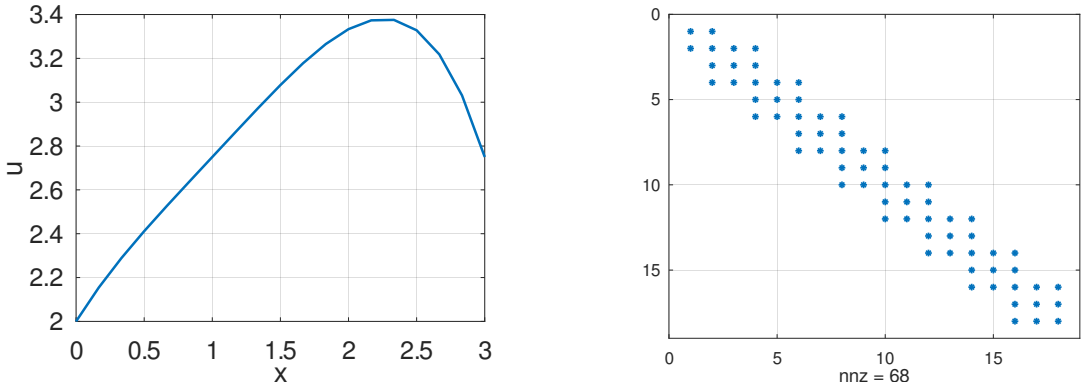
$$\begin{aligned} -u''(x) &= (1-x)^2 \qquad \text{for} \quad 0 \le x \le 3 \\ u(0) &= 2 \\ u'(3) &= -2 \end{aligned}$$

on an interval with 19 nodes use the code in `ODE1.m`, leading to Figure 21(a).

- The initial grid consists on $N - 1 = 9$ subintervals. The algorithm adds the midpoints and thus there are $2 \cdot 9 + 1 = 19$ nodes.
- The Dirichlet condition $u(0) = 2$ allows to remove one equation and consequently $\mathbf{A}$ will be a 18×18 matrix.
- Of the possible $18 \cdot 18 = 324$ entries in the stiffnes matrix $\mathbf{A}$ only 68 are different from zero. The nonzero entries are in a band of width 5 around the diagonal of $\mathbf{A}$, visible in Figure 21(b). This figure was generated by (temporarily) adding a command `spy(A)` in the code `BVP1D.m`.

ODE1.m

```
N = 10; x = linspace(0,3,N);
[xn,u] = BVP1D(x,1,0,0,1,@(x)(1-x).^2,2,[-2,0]);
figure(1); plot(xn,u);  xlabel('x'); ylabel('u')
```



(a) the solution $u(x)$ (b) the sparsity of $\mathbf{A}$

Figure 21: The solution for a simple 1D boundary value problem and the sparsity of the matrix $\mathbf{A}$

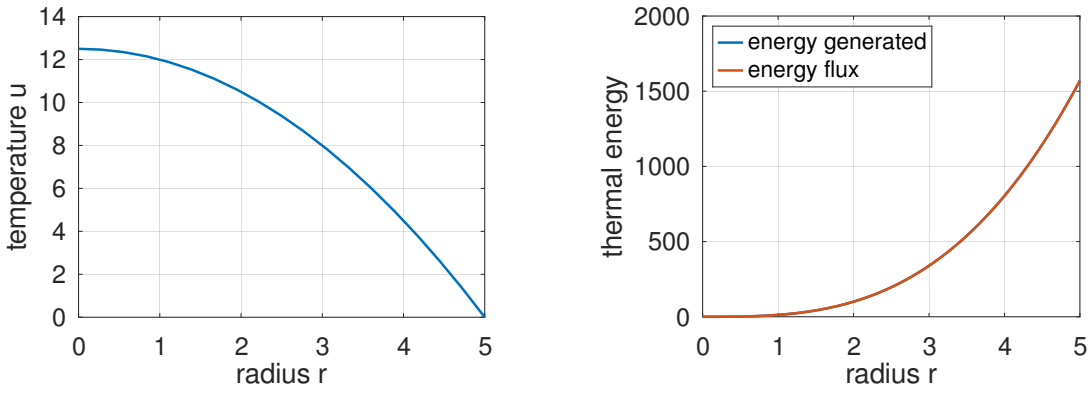
The constant and uniform heating of a ball of radius R can be described by a boundary value problem. The constant f indicates how much thermal energy is added to the ball per volume and time.

$$\begin{aligned} -\tfrac{\partial}{\partial r}\left(r^2\,\tfrac{\partial\,u(r)}{\partial r}\right) &= r^2\,f \qquad \text{for} \quad 0 \le r \le R \\ \tfrac{\partial}{\partial r}\,u(0) &= 0 \\ u(R) &= 0 \end{aligned}$$

This BVP determines the steady state solution. The code below and the resulting Figure 22(a) confirms that the maximal temperature is attained at the center of the ball.

HeatBall1D.m

```
R = 5;  f = 3; N = 10; r = linspace(0,R,N);
[r,u] = BVP1D(r,@(r)r.^2,0,0,@(r)r.^2,f,[0,0],0);
figure(1); plot(r,u); xlabel('radius r'); ylabel('temperature u')
```



(a) steady state solution (b) heat flux

Figure 22: The solution of a steady state heat problem and the heat flux across spheres of radius r

The thermal energy generated inside the radius r is given by $f\,\frac{4\,\pi}{3}\,r^3$ and should be equal to the flux through the sphere with radius r, i.e. $4\,\pi\,r^2\,\frac{\partial\,u(r)}{\partial r}$. The code below and the resulting Figure 22(b) confirms this observation.

HeatBall1D.m

```
r_fine = linspace(0,R,1001);
[u_fine,du_fine] = pwquadinterp(r,u,r_fine);
```

```
figure(2); plot(r_fine,f*4/3*pi*r_fine.^3,r_fine,-4*pi*r_fine.^2.*du_fine)
           xlabel('radius r'); ylabel('thermal energy')
           legend('energy generated','energy flux','location','northwest')
```

3.8 Solving 1D dynamic initial boundary value problems of order 1, a heat equation

The initial boundary value problem describing the dynamics of heating a ball with radius R and initial temperature $u(r,0) = u_0(r) = 0$ is given by

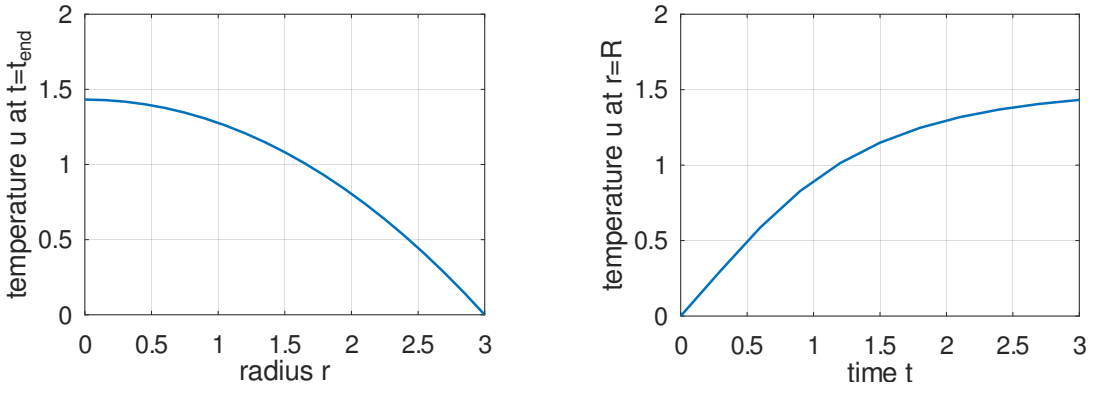
$$\begin{aligned} r^2 \frac{\partial}{\partial t} u(x,t) - \frac{\partial}{\partial r}\left(r^2 \frac{\partial}{dr} u(r,t)\right) &= r^2 f(r,t) && \text{for} \quad 0 < r < R \quad \text{and} \quad t > 0 \\ \frac{\partial}{\partial r} u(0,t) &= 0 && \text{for} \quad t > 0 \\ u(R,t) &= 0 && \text{for} \quad t > 0 \\ u(r,0) &= 0 && \text{for} \quad 0 < r < R \end{aligned} \quad .$$

This assumes that the temperature $u(r,t)$ depends on time t and radius r only and the temperature on the boundary $r = R$ is kept at 0 . Find the graphical output of the code below in Figures 23 and 24.

HeatingBallRadial.m

```
R = 3; BCleft = [0,0]; BCright = 0;  f = 1;
u0 = 0;  t0 = 0; t_end = 3;
steps = [10,10];  interval = linspace(0,R,11);
r_square = @(r) r.^2;
[r,u,t] = IBVP1D(interval,r_square,r_square,0,0,r_square,f,BCleft,BCright,...
                 u0,t0,t_end,steps);

figure(1); plot(r,u(:,end))
           xlabel('radius r'); ylabel('temperature u at t=t_{end}')
figure(2); plot(t,u(1,:))
           xlabel('time t'); ylabel('temperature u at r=R')
figure(3); mesh(t,r,u)
           xlabel('time t'); ylabel('radius r'); zlabel('temperature u')
figure(4); contour(t,r,u,[0.25:0.25:1.5])
           xlabel('time t'); ylabel('radius r');
```



(a) at final time $t = t_{end}$ (b) at border $r = R$

Figure 23: The solution of dynamic heating of a ball, u as function of r or t

3.9 Solving 1D dynamic initial boundary value problems of order 2, a wave equation

The initial boundary value problem describing the dynamics of a vibrating string with initial displacement

$$u(x,0) = u_0(x) = \begin{cases} \sin(x) & \text{for} \quad 0 \le x \le \pi \\ 0 & \text{for} \quad \pi \le x \le 3\,\pi \end{cases}$$

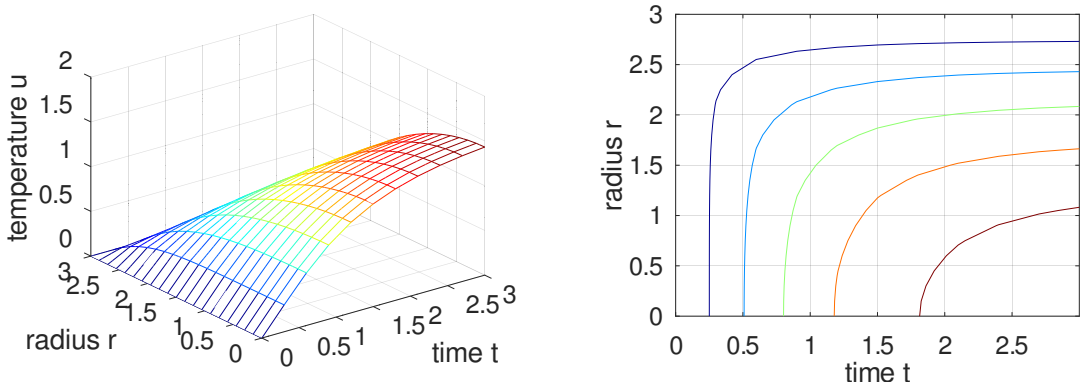


Figure 24: The solution of dynamic heating of a ball, u as function of r and t. The contours are at levels $u = 0.25$, 0.50, 0.75, 1.00 and 1.25.

and initial velocity

$$\frac{\partial}{\partial t}\,u(x,0) = u_1(x) = \begin{cases} -\cos(x) & \text{for} \quad 0 \le x \le \pi \\ 0 & \text{for} \quad \pi \le x \le 3\,\pi \end{cases}$$

is given by

$$\begin{aligned} \tfrac{\partial^2}{\partial t^2}\,u(x,t) - \tfrac{\partial^2}{\partial x^2}\,u(x,t)) &= 0 \qquad \text{for} \quad 0 < x < 3\,\pi \quad \text{and} \quad t > 0 \\ u(0,t) &= 0 \qquad \text{for} \quad t > 0 \\ \tfrac{\partial}{\partial x}\,u(3\,\pi,t) &= 0 \qquad \text{for} \quad t > 0 \\ u(x,0) &= u_0(x) \qquad \text{for} \quad 0 < x < 3\,\pi \\ \tfrac{\partial}{\partial t}\,u(x,0) &= u_1(x) \qquad \text{for} \quad 0 < x < 3\,\pi \end{aligned} \quad .$$

Find the graphical output of the code below in Figures 25 and 26.

Wave1D.m

```
a = 1; b = 0; c = 0; d = 1; f = 0;
w2 = 1; w1 = 0; BCleft = 0; BCright = [0,0];
t0 = 0; tend = 25; steps = [100,20];
interval = linspace(0,3*pi,51)';
u0 = @(x)sin(x).*(x<=pi); u1 = @(x)-cos(x).*(x<=pi);

[x,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,steps);

figure(1); mesh(t,x,u); xlabel('time t'); ylabel('position x'); zlabel('u')
           xlim([min(t),max(t)]); ylim([min(x),max(x)])
figure(2); contour(t,x,u,21); xlabel('time t'); ylabel('position x');
           colorbar
```

The initial hump of the form $\sin(x)$ for $0 \le x \le \pi$ is moving to the right with speed $c = 1$. At the border at $x = 3\,\pi$ the pulse is reflected, caused by the Neumann boundary condition $\frac{\partial}{\partial x}\,u(ß\,\pi,t) = 0$. Then is is moving back toward the border at $x = 0$. There it is reflected again, but with the negative of the amplitude, caused by the Dirichlet boundary condition $u(0,t) = 0$.

3.10 Solving nonlinear 1D boundary value problems

3.10.1 A nonlinear 1D BVP solved by `BVP1DNL()`

As example consider the boundary value problem

$$\begin{aligned} -\tfrac{d^2}{dx^2}\,u(x) &= \tfrac{1}{2} + \alpha\,(x\,u(x) + \sin(u(x)) \qquad \text{for } -1 < x < 2 \\ u(-1) &= 1 \\ u(+2) &= 4 \end{aligned}$$

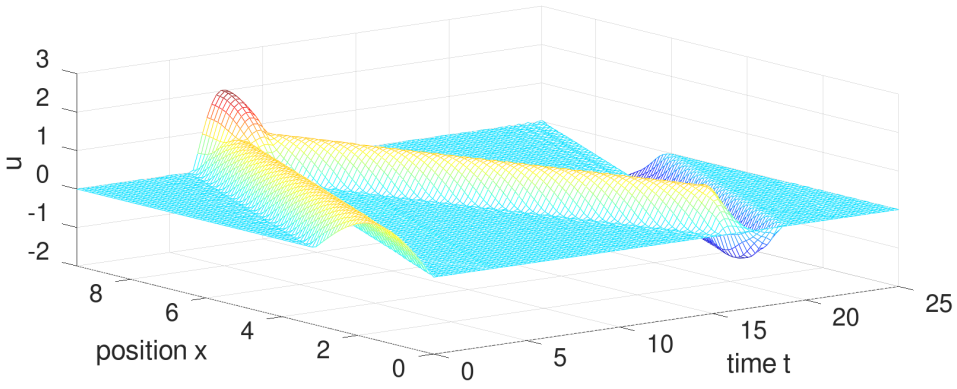


Figure 25: The amplitude of a vibrating string, u as function of x and t, the surface

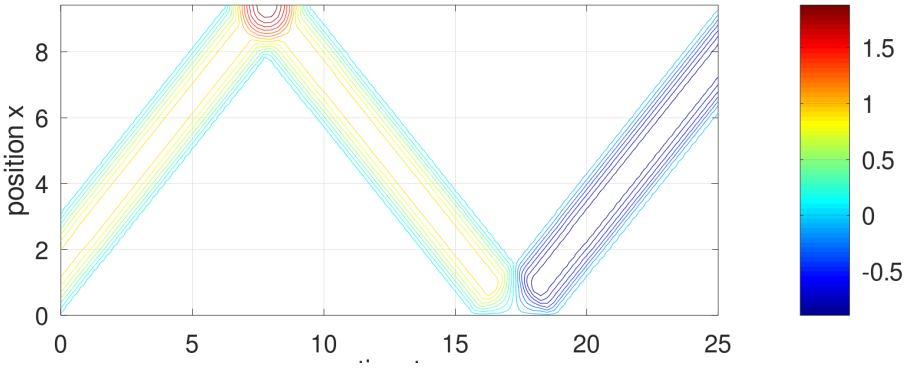


Figure 26: The amplitude of a vibrating string, u as function of x and t, the contour lines

with the parameter $\alpha = 0.1$. The right hand side and its partial derivative are given by

$$\begin{aligned} f(x,u) &= \frac{1}{2} + \alpha\,(x\,u(x) + \sin(u(x))) \\ \frac{\partial}{\partial u} f(x,u) &= 0 + \alpha\,(x + \cos(u(x))) \end{aligned}$$

and as initial guess for the solution use $u_0(x) = 1 + x$, which satisfies the boundary conditions. This nonlinear problem is then solved by the code below in `NL_Newton.m` . The iteration stopped after 2 steps. The progress of the iteration is displayed and the size of the last correction applied is $\approx 1 \cdot 10^{-7}$.

NL_Newton.m

```
interval = linspace(-1,2,40)';
al = +0.1;
f = {@(x,u)0.5+al*(x.*u+sin(u)), @(x,u)+al*(x+cos(u))};
BCleft = 1; BCright = 4;
u0 = @(x)2+x;
[x,u,inform] = BVP1DNL(interval,1,0,0,1,f,BCleft,BCright,u0,'Display','iter');
inform
figure(1); plot(x,u); xlabel('x'); ylabel('u')
-->
iteration 1, RMS(correction) = 1.465119e-02, RMS(phi) = 1.464906e-02
iteration 2, RMS(correction) = 1.065692e-07, RMS(phi) = 1.065692e-07

inform =  scalar structure containing the fields:
          info = 1
          iter = 2
          AbsError = 1.0657e-07
```

3.10.2 A nonlinear 1D BVP solved by successive substitution

To find the shortest connection of the form $y = u(x)$ between two points at $x = a$ and $x = b$ the length functional

$$L(u) = \int_a^b \sqrt{1 + (u'(x))^2}\; dx$$

has to be minimized. The corresponding Euler–Lagrange equation is

$$\frac{d}{dx}\left(\frac{1}{\sqrt{1 + (u'(x))^2}}\,\frac{d\,u(x)}{du}\right) = 0\,. \tag{42}$$

As example search the connection between $(x, y) = (0, 1)$ and $(1, 2)$, i.e. with the two boundary conditions $u(0) = 1$ and $u(1) = 2$. The straight line is easily determined by `BVP1DNL()` .

ShortestConnection.m

```
interval = linspace(0,1,21)';
a = @(x,u,du) 1./sqrt(1+du.^2);
u0 = @(x)2-cos(2*pi*x);
[x,u,inform] = BVP1DNL(interval,a,0,0,1,0,1,2,u0,'Tol',1e-4);
figure(1); plot(x,u); xlabel('x'); ylabel('u')
inform
-->
inform = scalar structure containing the fields:
    info = 1
    iter = 8
    AbsError = 1.1468e-04
```

Since the coefficient

$$a(x, u, u') = \frac{1}{\sqrt{1 + (u')^2}}$$

depends on u' a successive substitution is used and it takes many iteration to converge.

As second problem search for the shortest connection with $u(0) = 1$ and $a(u'(1))\,u'(1) = \alpha$, i.e.

$$\frac{1}{\sqrt{1+(u'(1))^2}}\,u'(1) = \alpha\,.$$

For $\alpha = 0.5$ the code below generates the result with 7 iterations. At the end of the code the values of $u(1)$, $u'(1)$ and the boundary condition are computed.

ShortestConnection.m

```
interval = linspace(0,1,21)';
a = @(x,u,du) 1./sqrt(1+du.^2);  alpha = 0.5;
u0 = @(x)1+x-0.3*(1-cos(2*pi*x));
[x,u,inform] = BVP1DNL(interval,a,0,0,1,0,1,[alpha,0],u0,
                       'tol',1e-6,'MaxIter',50,'tol',1e-4,'Display','iter');
figure(1); plot(x,u0(x),x,u); xlabel('x'); ylabel('u')
           legend('u_0','u','location','northwest')
inform
du = FEM1DEvaluateDu(x,u);
u_end_du_end = [u(end) du(end) du(end)*a(0,0,du(end))]
-->
iteration 1, RMS of correction = 2.424606e-01
iteration 2, RMS of correction = 3.402180e-02
iteration 3, RMS of correction = 1.014111e-02
iteration 4, RMS of correction = 2.689617e-03
iteration 5, RMS of correction = 6.834600e-04
iteration 6, RMS of correction = 1.715824e-04
iteration 7, RMS of correction = 4.294086e-05

inform =  scalar structure containing the fields:
    info = 1
    iter = 7
    AbsError = 4.2941e-05
u_end_du_end =  1.5774   0.5774   0.5000
```

It is easy to see[8] that for $\alpha > 1$ there is no solution. The above code for $\alpha = 0.9$ will converge only after 28 iterations. For $\alpha > 1$ it will not converge at all!

3.11 A dynamic nonlinear initial boundary value problem

As a first (academic) example of (18) examine a nonlinear dynamic heat equation.

$$\begin{aligned} \tfrac{\partial}{\partial t}\,u(x,t) - \tfrac{1}{10}\,\tfrac{\partial^2}{\partial x^2}\,u(x,t) &= x^3 + \sin(u(x,t)) && \text{for } 0 \le x \le 1 \text{ and } t \ge 0 \\ u(0,t) = u(1,t) &= 0 && \text{for } t \ge 0 \\ u(x,0) &= 0 && \text{for } 0 \le x \le 1 \end{aligned}$$

With the functions

$$f(x,t,u) = x^3 + \sin(u) \quad \text{use} \quad \frac{\partial}{\partial u}\,f(x,t,u) = \cos(u)$$

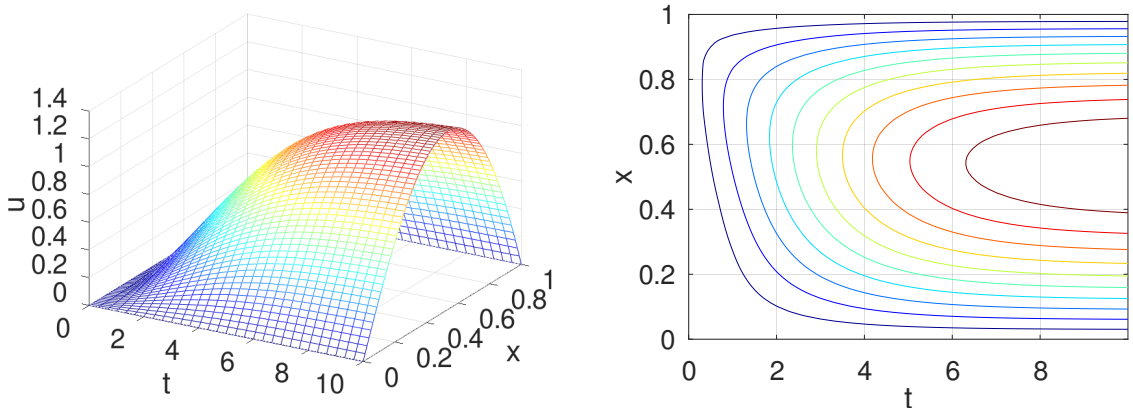
the code `HeatDynamic.m` below leads to Figure 27.

- The solution starts out at $t = 0$ with $u(x,0) = 0$ and then is moving up, caused by the heating term $x^3 + \sin(u)$.
- For large times t the solution $u(x,t)$ converges to the solution of the static problem

  $$-\frac{1}{10}\,\frac{\partial^2}{\partial x^2}\,u(x) = x^3 + \sin(u(x))$$

  with zero Dirichlet boundary conditions. This can be verified by using the command `BVP1DNL()`, as done in the code below but the figure is not shown in these notes.

[8] The Euler–Lagrange equation implies that $u'(x) = \beta$ is a constant. Then the equation $\frac{\beta}{\sqrt{1+\beta^2}} = \alpha$ has to be solved, leading to $\beta = \frac{\alpha}{\sqrt{1-\alpha^2}}$. The monotonous increasing function $f(z) = \frac{z}{\sqrt{1+z^2}}$ with the limits $\lim_{z\to\pm\infty} f(z) = \pm 1$ shows that the BVP has a solution for $-1 < \alpha < +1$.



(a) the solution $u(x,t)$ (b) the contour

Figure 27: The solution and its contour for a nonlinear dynamic heat problem

HeatNonlinear.m

```
L = 1; N = 41;  Interval = linspace(0,L,N)';
w = 1; a = 0.1 ; b = 0; c = 0; d = 1;
f = {@(x,t,u)x.^3 + sin(u),@(x,t,u)cos(u)};
t0 = 0; tend = 10; steps = [30,50];
BCleft = 0; BCright = 0;  u0 = 0;
[x,u_all,t] = IBVP1DNL(Interval,w,a,b,c,d,f,BCleft,BCright,u0,t0,tend,steps);
figure(1); mesh(t,x,u_all); xlabel('t'); xlim([t0,tend])
           ylabel('x'); zlabel('u'); view([30,30])
figure(2); contour(t,x,u_all); xlabel('t'); ylabel('x');

[x,u] = BVP1DNL(Interval,a,b,c,d,{@(x,u)x.^3+sin(u),@(x,u)cos(u)},BCleft,BCright,u0);
figure(3); plot(x,u,x,u_all(:,end)); xlabel('x'); ylabel('u');
           legend('static','dynamic', 'location','south')
```

3.12 A dynamic nonlinear wave equation

An academic example of a nonlinear wave equation of the type (19) is given by

$$\begin{aligned} \tfrac{\partial^2}{\partial t^2}\,u(x,t) + \tfrac{\partial}{\partial t}\,u(x,t) - \tfrac{\partial^2}{\partial x^2}\,u(x,t) &= \tfrac{1}{2}\,(\exp(\tfrac{-x}{L}) - \tfrac{1}{2}\,\sin(u(x,t))) && \text{for } 0 \le x \le L = 10 \text{ and } t \ge 0 \\ u(0,t) = u(L,t) &= 0 && \text{for } t \ge 0 \\ u(x,0) &= 0 && \text{for } 0 \le x \le L \\ \tfrac{\partial}{\partial t}\,u(x,0) &= \tfrac{x}{10}\,\sin(\tfrac{\pi\,x}{L}) && \text{for } 0 \le x \le L \end{aligned}$$

With the function

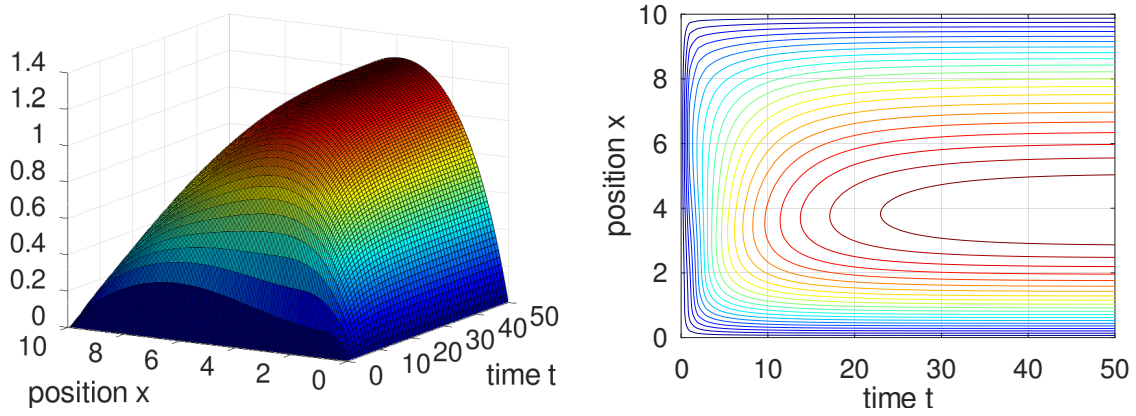
$$f(x,t,u) = \frac{1}{2}\,(\exp(\frac{-x}{L}) - \frac{1}{2}\,\sin(u(x,t)))$$

the code `DemoWaveNL.m` below leads to Figure 28.

Caused by the damping term $-\frac{\partial\,u}{\partial t}$ the solution of the dynamic problem will converge to the solution of the static problem

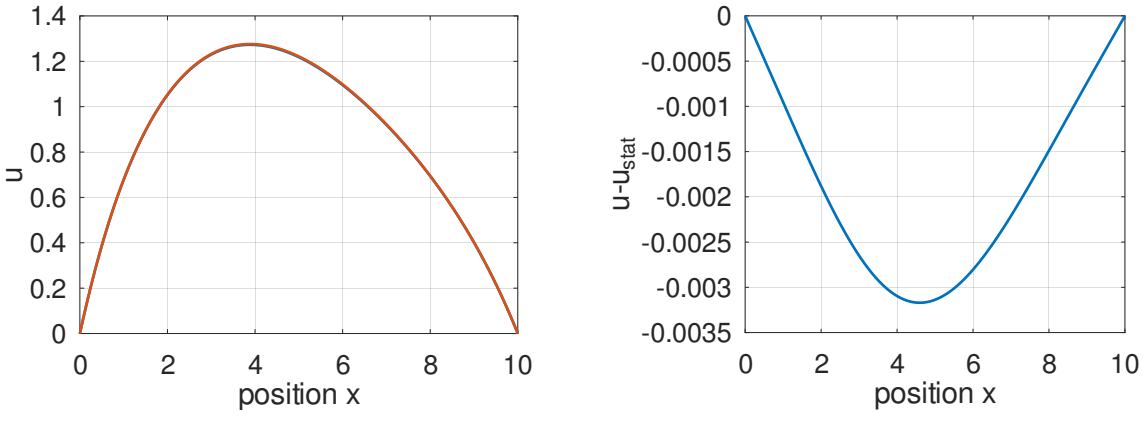
$$-\frac{1}{10}\,\frac{\partial^2}{\partial x^2}\,u(x,t) = \frac{1}{2}\,(\exp(\frac{-x}{L}) - \frac{1}{2}\,\sin(u(x,t))) \quad \text{for } 0 \le x \le L \text{ with } u(0) = u(L) = 0\,.$$

This nonlinear steady state equation is solved with the help of `BVP1DNL()`. In Figure 29 find graphs of this steady state solution and the final state of the dynamic problem. The remaining difference $u(x,T_{end}) - u_{static}(x)$ will be smaller for larger final times T_{end}.



(a) the solution $u(x,t)$ (b) the contour

Figure 28: The solution and its contour for a nonlinear dynamic wave problem



(a) final state $u(x,T_{end})$ and steady state $u_{static}(x)$ (b) difference to steady state

Figure 29: The final state $u(x,T_{end})$ and the steady state solution $u_{static}(x)$

DemoWaveNL.m

```
L = 10; Interval = linspace(0,L,101); Tend = 50; D = 1;
f  = @(x,t,u)0.5*(exp(-x/L)-0.5*sin(u)); v0 = @(x)0.1*x.*sin(pi*x/L);
[x,u,t] = I2BVP1DNL(Interval,1,D,1,0,0,1,f,0,0,0,v0,0,Tend,[51,100] ,'Solver','implicit');
figure(1); surf(t,x,u); view(-60,15)
           xlabel('time t'); ylabel('position x')
figure(2); contour(t,x,u,21)
           xlabel('time t'); ylabel('position x')

u0 = @(x)sin(pi*x/L);
f_stab  = {@(x,u)0.5*(exp(-x/L)-0.5*sin(u)), @(x,u)-0.5*0.5*cos(u)};
[x,u_stat] = BVP1DNL(Interval,1,0,0,1,f_stab,0,0,u0);
figure(3); plot(x,u(:,end),x,u_stat); xlabel('position x'); ylabel('u')
figure(4); plot(x,u(:,end)-u_stat);   xlabel('position x'); ylabel('u-u_{stat}')
```

3.13 Plane elasticity

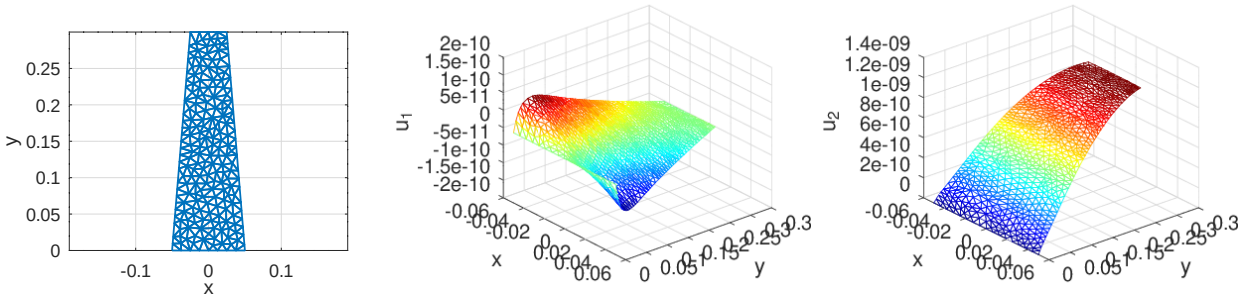
In this section a typical plane stress situation is examined to illustrate the related commands. This is followed by a similar plane strain situation. Then a plane stress eigenvalue problem and a dynamic situation are examined.

3.13.1 A plane stress example

On a trapezoidal domain visible in Figure 30(a) a plane stress problem is set up.

- The material parameters E and ν describe copper.
- At the lower edge at $y = 0$ the displacements are zero, i.e. $u_1(x, 0) = u_2(x, 0) = 0$ for $-0.05 \le x \le +0.05$.
- The other edges are force free.
- On all of the domain a force density of $\vec{f} = (0\,,\, \frac{100}{0.3 \cdot 0.1}) \approx (0\,,\, 3333)$ is given. Thus the solid is pushed in y direction.
- An initial mesh is generated with the help of `triangle` and then upgraded to a mesh with second order elements.

With a call of `PlaneStress()` the displacements $\vec{u}_1$ and $\vec{u}_2$ are computed and then displayed, leading to Figure 30.



(a) the domain (b) the displacement u_1 (c) the displacement u_2

Figure 30: The computational domain and the two displacements u_1 and u_2 for the plain stress example.

PlaneStressExample.m

```
W = 0.1; H = 0.3; Load = 100; E = 110e9; nu = 0.35;  %% copper
FEMmesh = CreateMeshTriangle('Example1',...
             [-W/2 0, -11; +W/2 0 -22; W/4 H -22; -W/4 H -22],0.0001);
figure(1); FEMtrimesh(FEMmesh)
           xlabel('x'); ylabel('y'); axis equal
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment for second order elements
```

```
f = {0,Load/(H*W)}; gD = {0,0}; gN = {0,0};
[u1,u2] = PlaneStress(FEMmesh,E,nu,f,gD,gN);
figure(2); FEMtrimesh(FEMmesh,u1)
           xlabel('x'); ylabel('y'); zlabel('u_1');  view([50,30])
figure(3); FEMtrimesh(FEMmesh,u2)
           xlabel('x'); ylabel('y'); zlabel('u_2');  view([50,30])
```

With `EvaluateStrain()` the three strains ε_{xx}, ε_{yy} and ε_{xy} are determined at the nodes and displayed, leading to Figure 31. The Saint–Venant's principle at the lower edge $y = 0$ is clearly visible.

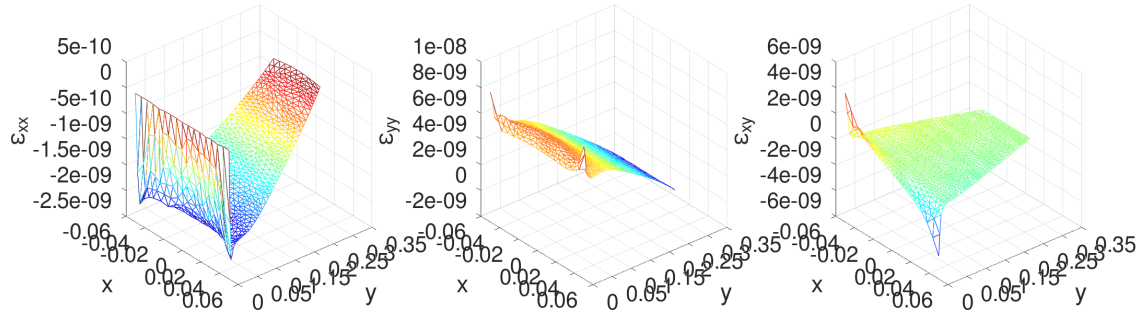


Figure 31: The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy} for the plain stress example.

PlaneStressExample.m

```
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(FEMmesh,u1,u2);
figure(4);
subplot(1,3,1); FEMtrimesh(FEMmesh,eps_xx)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}'); view([50,30])
subplot(1,3,2); FEMtrimesh(FEMmesh,eps_yy)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}'); view([50,30])
subplot(1,3,3); FEMtrimesh(FEMmesh,eps_xy)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{xy}'); view([50,30])
```

With `EvaluateStress()` the three stresses σ_x, σ_y and τ_{xy} are determined at the nodes and displayed, leading to Figure 32. The Saint–Venant's principle at the lower edge $y = 0$ is again clearly visible.

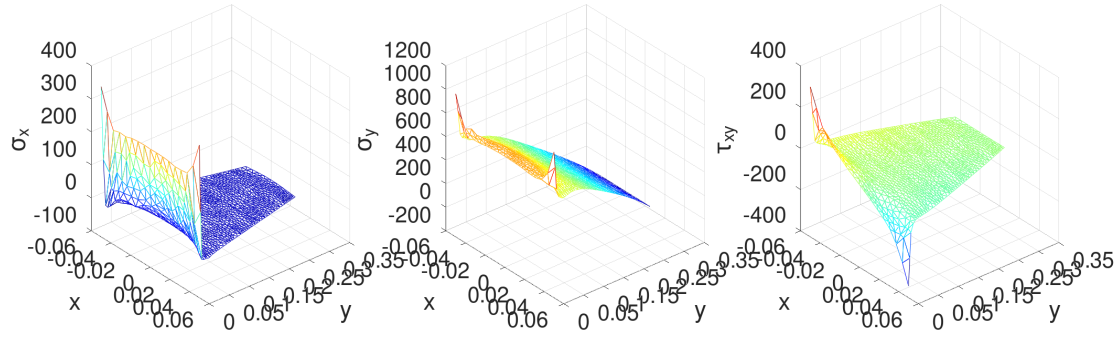


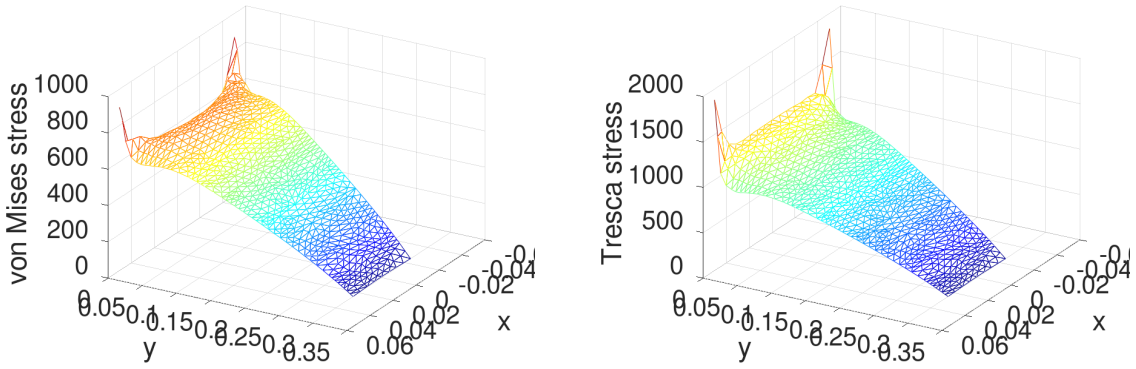
Figure 32: The normal stresses σ_x and σ_y and the shearing stress τ_{xy} for the plain stress example.

PlaneStressExample.m

```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(FEMmesh,u1,u2,E,nu);
figure(5);
subplot(1,3,1); FEMtrimesh(FEMmesh,sigma_x)
                xlabel('x'); ylabel('y'); zlabel('\sigma_x');  view([50,30])
```

```
subplot(1,3,2); FEMtrimesh(FEMmesh,sigma_y)
                xlabel('x'); ylabel('y'); zlabel('\sigma_y');  view([50,30])
subplot(1,3,3); FEMtrimesh(FEMmesh,tau_xy)
                xlabel('x'); ylabel('y'); zlabel('\tau_{xy}'); view([50,30])
```

With the two commands `EvaluateVonMises()` and `EvaluateTresca()` the von Mises stress and the Tresca stress are computed and displayed, leading to Figure 33. At the end of the code the two principal stresses σ_1 and σ_2 are computed, but not displayed.



(a) von Mises stress (b) Tresca stress

Figure 33: The von Mises and Tresca stress for the plain stress example.

PlaneStressExample.m

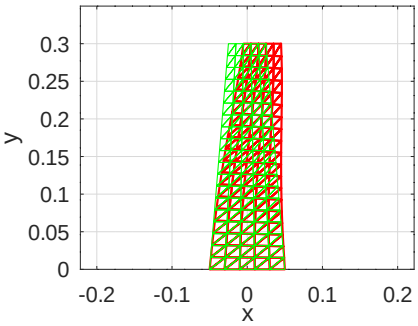
```
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);
figure(6); FEMtrimesh(FEMmesh,vonMises)
           xlabel('x'); ylabel('y'); zlabel('von Mises stress'); view([120,30])
Tresca = EvaluateTresca(sigma_x,sigma_y,tau_xy);
figure(7); FEMtrimesh(FEMmesh,Tresca)
           xlabel('x'); ylabel('y'); zlabel('Tresca stress'); view([120,30])
[s1,s2] = EvaluatePrincipalStress(sigma_x,sigma_y,tau_xy);
```

3.13.2 A plane strain example

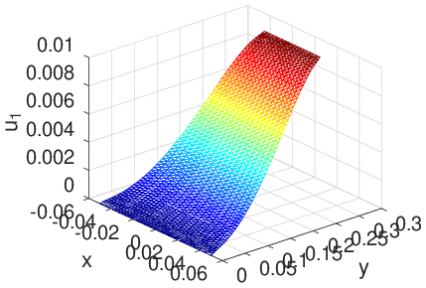
On the trapezoidal domain visible in Figure 30(a) a plane strain problem is set up.

- The material parameters E and ν describe copper.
- At the lower edge at $y = 0$ the displacements are zero, i.e. $u_1(x, 0) = u_2(x, 0) = 0$ for $-0.05 \leq x \leq +0.05$.
- At the upper edge at $y = 0.3$ the horizontal displacements is set to $+0.01$ and the vertical displacement is zero.
- The edges on the side are force free.
- There is no volume force applied to the domain, i.e. $\vec{f} = \vec{0}$.
- An initial mesh is generated by deforminag a regular, rectangular mesh, and then upgraded to a mesh with second order elements.

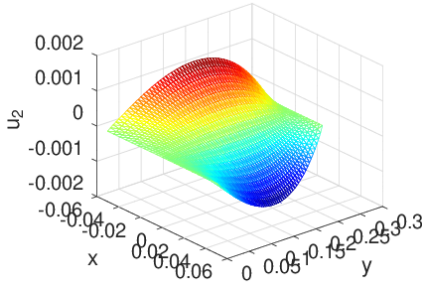
With a call of `PlaneStrain()` the displacements $\vec{u}_1$ and $\vec{u}_2$ are computed and then displayed, leading to Figure 34. A corser mesh on the same domain is generated and then used to display the original and deformed domain. Find the result in Figure 34(a) with the original domain in green and the deformed domain in red.



(a) the original and deformed domain



(b) the displacement u_1



(c) the displacement u_2

Figure 34: The computational domain and the two displacements u_1 and u_2 for the plain strain example.

PlaneStrainExample.m

```
W = 0.1; H = 0.3; E = 110e9; nu = 0.35;  %% copper
FEMmesh = CreateMeshRect(linspace(-W/2,W/2,10),linspace(0,H,30),-11,-11,-22,-22);
function xy_new = Deform(xy)
  xy_new = [xy(:,1).*(1-0.5/0.3*xy(:,2)) , xy(:,2)];
endfunction
FEMmesh = MeshDeform(FEMmesh,'Deform');
CMesh = CreateMeshRect(linspace(-W/2,W/2,6),linspace(0,H,20),-11,-11,-22,-22);
CMesh = MeshDeform(CMesh,'Deform'); %% create a course mesh on the same domain
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

f = {0,0}; gN = {0,0};
function res = gD(xy)
  res = +(xy(:,2)>0.1)*0.01;
endfunction

[u1,u2] = PlaneStrain(FEMmesh,E,nu,f,{'gD',0},gN);
u1i = FEMgriddata(FEMmesh,u1,CMesh.nodes(:,1),CMesh.nodes(:,2));
u2i = FEMgriddata(FEMmesh,u2,CMesh.nodes(:,1),CMesh.nodes(:,2));
figure(1); ShowDeformation(CMesh,u1i,u2i,2)
           axis equal; xlabel('x'); ylabel('y'); ylim([0,0.35])
figure(2); FEMtrimesh(FEMmesh,u1)
           xlabel('x'); ylabel('y'); zlabel('u_1'); view([50,30])
figure(3); FEMtrimesh(FEMmesh,u2)
           xlabel('x'); ylabel('y'); zlabel('u_2'); view([50,30])
```

With `EvaluateStrain()` the three strains ε_{xx}, ε_{yy} and ε_{xy} are determined at the nodes and displayed, leading to Figure 35.

PlaneStrainExample.m

```
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(FEMmesh,u1,u2);
```

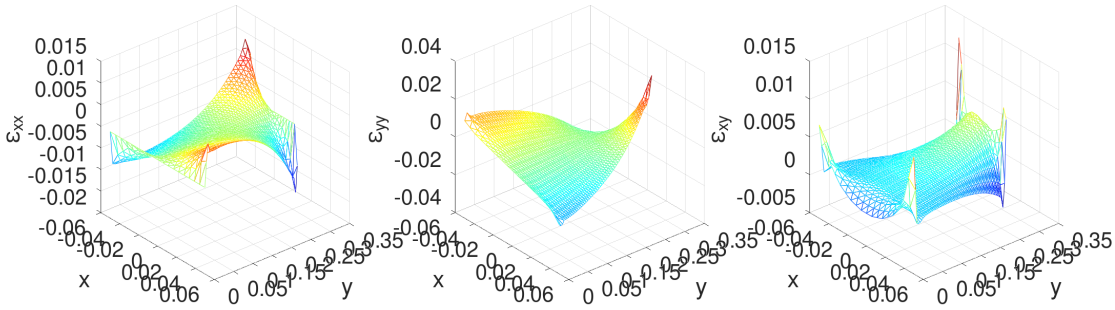


Figure 35: The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy} for the plain strain example.

```
figure(4);
subplot(1,3,1); FEMtrimesh(FEMmesh,eps_xx)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}'); view([50,30])
subplot(1,3,2); FEMtrimesh(FEMmesh,eps_yy)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}'); view([50,30])
subplot(1,3,3); FEMtrimesh(FEMmesh,eps_xy)
                xlabel('x'); ylabel('y'); zlabel('\epsilon_{xy}'); view([50,30])
```

With `EvaluateStress()` the three stresses σ_x, σ_y and τ_{xy} are determined at the nodes and displayed, leading to Figure 36. Observe that the function `EvaluateStress()` is called with for four return arguments, including σ_z. This assures that the plane strain expressions are used for the computations.

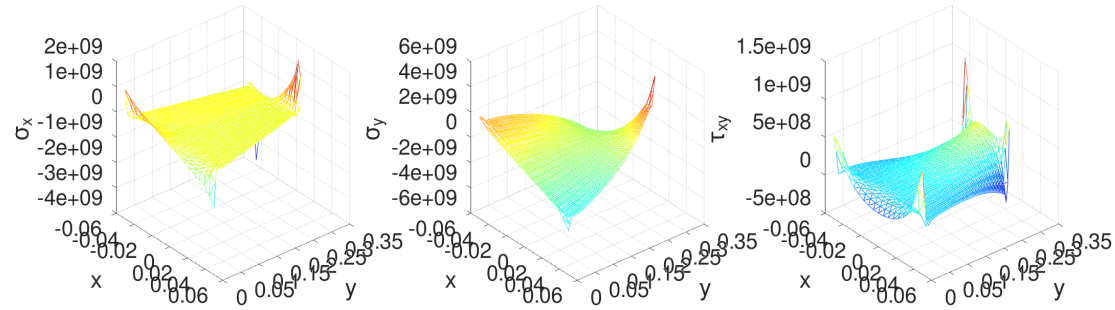


Figure 36: The normal stresses σ_x and σ_y and the shearing stress τ_{xy} for the plain strain example.

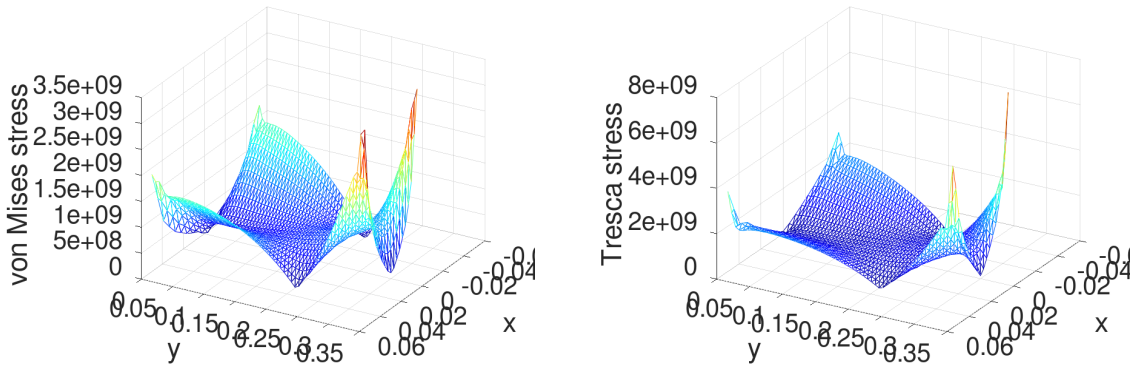
PlaneStrainExample.m

```
[sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(FEMmesh,u1,u2,E,nu);
figure(5); title('stress')
subplot(1,3,1); FEMtrimesh(FEMmesh,sigma_x)
                xlabel('x'); ylabel('y'); zlabel('\sigma_x'); view([50,30])
subplot(1,3,2); FEMtrimesh(FEMmesh,sigma_y)
                xlabel('x'); ylabel('y'); zlabel('\sigma_y'); view([50,30])
subplot(1,3,3); FEMtrimesh(FEMmesh,tau_xy)
                xlabel('x'); ylabel('y'); zlabel('\tau_{xy}'); view([50,30])
```

With the two commands `EvaluateVonMises()` and`EvaluateTresca()` the von Mises stress and the Tresca stress are computed and displayed, leading to Figure 33. Observe that four input arguments are given for the functions `EvaluateVonMises()` and `EvaluateTresca()`, including σ_z. This assures that the plane strain expressions are used for the computations. At the end of the code the two unknown principal stresses σ_1 and σ_2 are computed, but not displayed.

PlaneStrainExample.m

```
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy,sigma_z);
figure(6); FEMtrimesh(FEMmesh,vonMises)
           xlabel('x'); ylabel('y'); zlabel("von Mises stress"); view([120,30])
Tresca = EvaluateTresca(sigma_x,sigma_y,tau_xy,sigma_z);
figure(7); FEMtrimesh(FEMmesh,Tresca)
           xlabel('x'); ylabel('y'); zlabel("Tresca stress"); view([120,30])
[s1,s2] = EvaluatePrincipalStress(sigma_x,sigma_y,tau_xy);
```

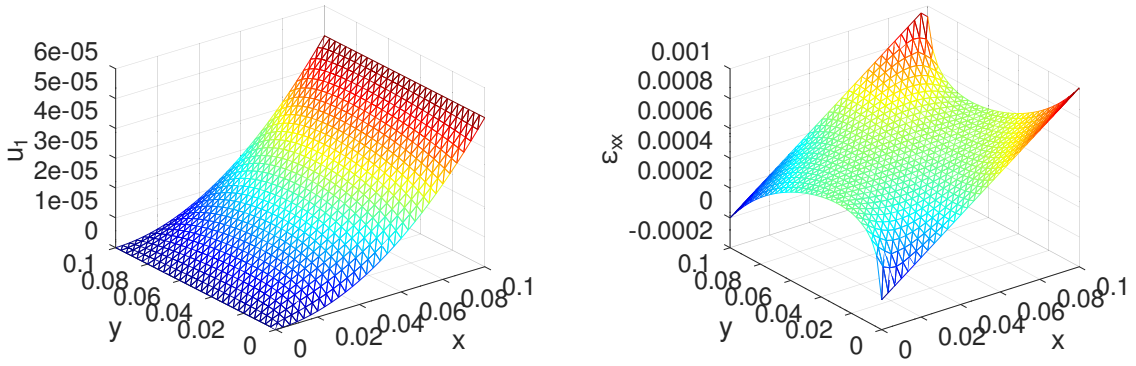


(a) von Mises stress (b) Tresca stress

Figure 37: The von Mises and Tresca stress for the plain strain example.

3.13.3 A plane stress problem with thermal expansion

Examine a copper plate with a clamped boundary by (rather academic) boundary conditions. Due to these boundary conditions the plane stress plate is stretched, i.e. all normal stresses are positive. Then the plates temperature is incleased by 30°. This will expand the plate even more, but the clamped boundary conditions stay in place. Thus there will be sections with a compressive stress. This is modeled with the code `Demo_PlaneStressThermal.m` shown below. Find the displacement u_1 and the corresponding strain ε_{xx} in Figure 38. The resulting normal stress is shown in Figure 39.



(a) displacement u_1 (b) strain ε_{xx}

Figure 38: Displacement and strain for a plane stress plate with thermal expansion

Demo_PlaneStressThermal.m

```
E = 110e3; nu = 0.35; alpha = 1.7e-5; %% copper, length in mm
```

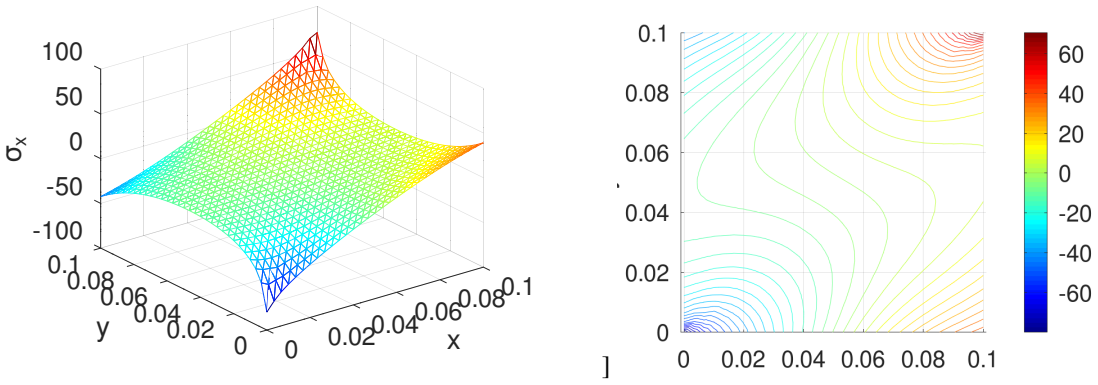


Figure 39: The normal stress σ_x for a plane stress plate with thermal expansion

```
L = 0.1;  N = 11;

Mesh = CreateMeshRect(linspace(0,L,N),linspace(0,L,N),-11,-11,-11,-11);
figure(1); FEMtrimesh(Mesh)
% Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

g1 = @(xy)(0.005*xy(:,1).^2); g2 = @(xy)(0.005*xy(:,2).^2);
alphaDeltaT = alpha*30;  %% approx 0.0005

[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{g1,g2},{0,0},'thermal',alphaDeltaT);
figure(2); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(3); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')

[eps_xx,eps_yy,eps_xy] = EvaluateStrain(Mesh,u1,u2);
figure(4); FEMtrimesh(Mesh,eps_xx); xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(5); FEMtrimesh(Mesh,eps_yy); xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')

[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',alphaDeltaT);
figure(6); FEMtrimesh(Mesh,sigma_x); xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(7); FEMtrimesh(Mesh,sigma_y); xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(8); FEMtricontour(Mesh,sigma_x,41); xlabel('x'); ylabel('y');
           colorbar(); axis equal
figure(9); FEMtricontour(Mesh,sigma_y,41); xlabel('x'); ylabel('y');
           colorbar(); axis equal

MaxSigma_x  = max(abs(sigma_x))
MeanSigma_x = mean(sigma_x)
```

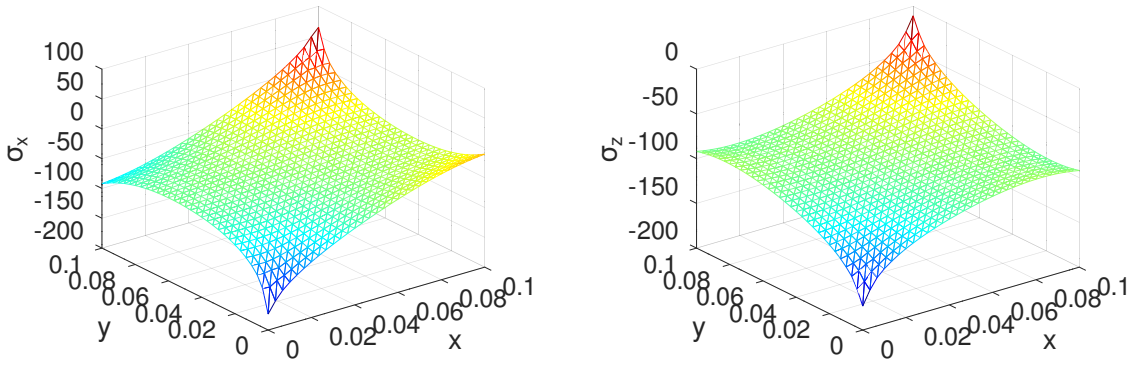
When a plane strain assumption is used the above code can be modified slightly. Observe that the normal stress σ_z has to be evaluated with a call of `EvaluateStress()` with four return arguments.

Demo_PlaneStrainThermal.m

```
E = 110e3; nu = 0.35; alpha = 1.7e-5; %% copper, length in mm
L = 0.1;  N = 11;

Mesh = CreateMeshRect(linspace(0,L,N),linspace(0,L,N),-11,-11,-11,-11);
figure(1); FEMtrimesh(Mesh)
% Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

g1 = @(xy)(0.005*xy(:,1).^2); g2 = @(xy)(0.005*xy(:,2).^2);
alphaDeltaT = alpha*30;  %% approx 0.0005
```



(a) normal stress σ_x (b) normal stress σ_z

Figure 40: Two normal stresses for a plane strain plate with thermal expansion

```
[u1,u2] = PlaneStrain(Mesh,E,nu,{0,0},{g1,g2},{0,0},'thermal',alphaDeltaT);
figure(2); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(3); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')

[eps_xx,eps_yy,eps_xy] = EvaluateStrain(Mesh,u1,u2);
figure(4); FEMtrimesh(Mesh,eps_xx); xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(5); FEMtrimesh(Mesh,eps_yy); xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')

[sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',alphaDeltaT);
figure(6); FEMtrimesh(Mesh,sigma_x); xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(7); FEMtrimesh(Mesh,sigma_y); xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(8); FEMtricontour(Mesh,sigma_x,41); xlabel('x'); ylabel('y');
           colorbar(); axis equal
figure(9); FEMtricontour(Mesh,sigma_y,41); xlabel('x'); ylabel('y');
           colorbar(); axis equal
figure(10); FEMtrimesh(Mesh,sigma_z); xlabel('x'); ylabel('y'); zlabel('\sigma_z')
figure(11); FEMtricontour(Mesh,sigma_z,41); xlabel('x'); ylabel('y');
            colorbar(); axis equal

MaxSigma_x  = max(abs(sigma_x))
MeanSigma_x = mean(sigma_x)
```

3.13.4 A plane stress eigenvalue problem and a dynamic problem

Examine an aluminum beam of length $L = 0.2$, height $H = 0.01$ and width $W = 0.01$. The beam is clamped on the left at $x = 0$ and the other boundaries are free. According to the Euler beam theory[9] the frequency of the first eigenmode is given by

$$\text{freq} = \frac{z_0^2\sqrt{E\,I}}{2\,\pi\,\sqrt{\rho\,H\,W}\,L^2} \approx 205.63 \text{ Hz}\,.$$

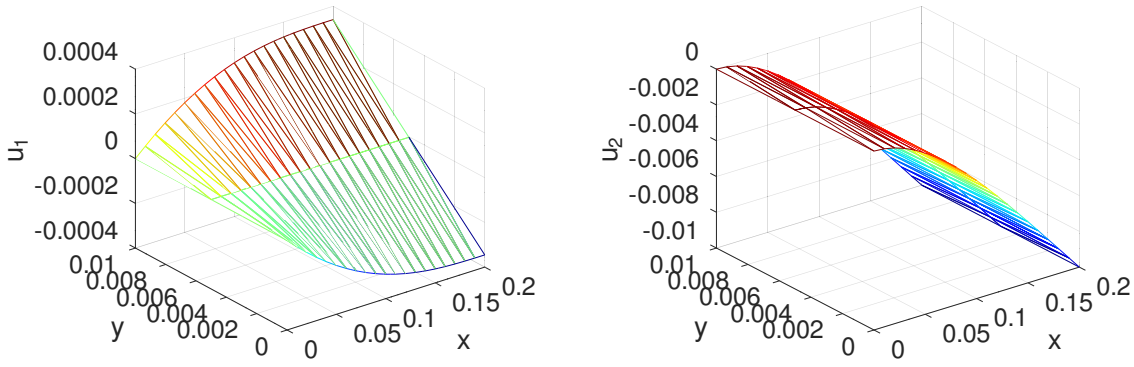
With `FEMoctave` use the command `PlaneStressEig()` to determine the first eigenvalue and the corresponding eigenmode. A mesh with 80 second order elements is used. The code determines the frequency and the horizontal and vertical displacements, see Figure 41.

EulerBeamMode1.m

```
clear *
L = 0.20; H = 0.01; W = 0.01; rho = 2.7e3;
E = 70e9; nu = 0.33; %% Aluminum
I2 = 1/12*H^3*W;
```

[9] The coefficient $z_0 \approx 1.8751$ is the first zero of the function $f(z) = 1 - \cos(z)\,\cosh(z)$ and $I = \frac{1}{12}\,W\,H^3$ is the second moment of the cross section.

```
f = @(z) 1+cos(z).*cosh(z);  %% clamped at x=0, free at x=L
z0 = fsolve(f,pi/2);
freqEuler = z0^2*sqrt(E*I2/(rho*H*W))/(2*pi*L^2)
Nx = 20; Ny = 2;
Mesh = CreateMeshRect(linspace(0,L,Nx+1),linspace(0,+H,Ny+1),-22,-22,-11,-22);
Mesh = MeshUpgrade(Mesh,'quadratic');
[la,u1,u2] = PlaneStressEig(Mesh,E,nu,rho,1);
freqFEM = sqrt(la)/(2*pi)
u1 = u1/max(abs(u2))/100; u2 = u2/max(abs(u2))/100;
figure(1);FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(2);FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')
-->
freqEuler = 205.63
freqFEM   = 205.69
```



(a) displacement u_1 in x–direction

(b) displacement u_2 in y–direction

Figure 41: The first eigenmode of a bending beam

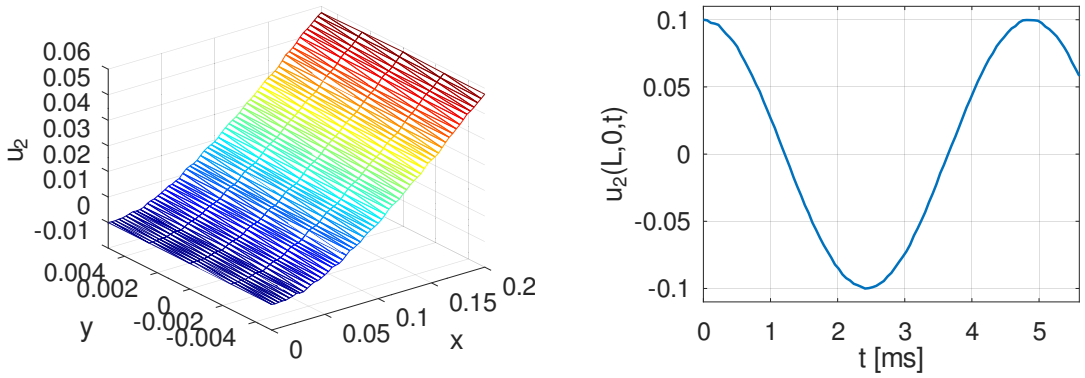
The same setup can be examined as a dynamic problem. As initial displacement use the shape of the first eigenmode and zero initial velocity. Then the command `PlaneStressDynamic()` can construct a solution, leading to Figure 42. The above frequency of 205.6 Hz is confirmed by the period $\frac{1}{205.6} \approx 4.86$ ms, visible in Figure 42.

EulerBeamDynamic.m

```
E = 70e9; nu = 0.33; rho = 2.7e3; L = 0.2; H = 0.01;
f = {0,0}; gD = {0,0}; gN = {0,0};
function res = u0Func(xy)
  z = 1.8751; L = max(xy(:,1));  C = -(cos(z)+cosh(z))/(sin(z)+sinh(z));
  x = z*xy(:,1)/L;
  res = cos(x)-cosh(x) + C*(sin(x)-sinh(x));
  res = -0.1*res./max(abs(res));
endfunction
u0 = {0,'u0Func'}; v0 = {0,0};
t0 = 0; tend = 0.0056; steps = [100,20];

Mesh = CreateMeshRect(linspace(0,L,31),linspace(-H/2,+H/2,3),-22,-22,-11,-22);
Mesh = MeshUpgrade(Mesh,'quadratic');  solver = 'implicit';

[u1_all,u2_all,t] = PlaneStressDynamic(Mesh,E,nu,rho,f,gD,gN,u0,v0,t0,...
                                       tend,steps,'solver',solver);
u1 = u1_all(:,end); u2 = u2_all(:,end);
figure(1); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2');
           ylim([-H/2,H/2])
ind = find((Mesh.nodes(:,1)==L).*(Mesh.nodes(:,2)==0)); u2_t = u2_all(ind,:);
figure(2); plot(t*1e3,u2_t); xlabel('t [ms]'); ylabel('u_2(L,0,t)');
           xlim([0,max(t)*1e2]); ylim(0.11*[-1,+1])
```



(a) u_2 at final time

(b) movement of u_2 at end point

Figure 42: The first eigenmode of a bending beam as dynamic problem

3.14 An axially symmetric elasticity example

A rectangular domain $0 \le r = x \le R = 0.1$ and $-2\,R \le z \le 2\,R$ is rotated about the z–axis and on the middle section $-R \le z \le R$ of the surface an external pressure of the form

$$p(z) = \begin{cases} P\,(R^2 - z^2) & \text{for } |z| \le R \\ 0 & \text{for } |z| > R \end{cases}$$

is applied. The aim is to determine the radial displacement u_r and the z–displacement u_z, as function of $x = r$ and z.

- Due to the symmetry only the upper half of the cylinder has to be examined, with the boundary condition $u_z = 0$ in the plane $z = 0$.
- Along the z–axis the boundary condition is $u_r = 0$.
- The upper edge is force free.
- Along the right edge at $r = R$ the external pressure is applied.

As a first step create the mesh, define the pressure funcation and the matreial parameters. Then solve the problem using the function `AxiStress()`.

AxiSymmetricExample.m

```
R = 0.1;
if 0  %% nonuniform mesh
  Mesh = CreateMeshTriangle('AxiSymm',[0 0 -21; R 0 -32; R 2*R -22; 0 2*R -12],1e-4);
else
  Mesh = CreateMeshRect(linspace(0,R,10),linspace(0,2*R,20),-21,-22,-12,-32);
endif
Mesh = MeshUpgrade(Mesh,'quadratic');

function res = force(rz)
  R = 0.1; P = 1e5;    res = -P*max(R^2-rz(:,2).^2,0);
endfunction

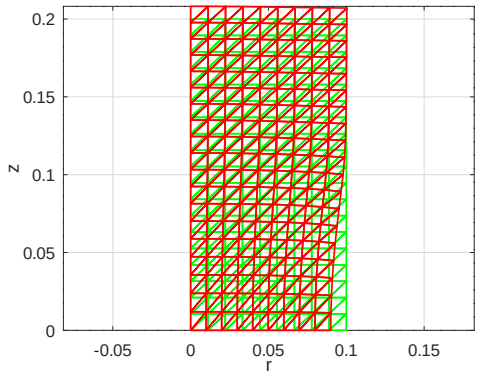
E = 1e9; nu = 0.3; f = {0,0}; gD = {0,0}; gN = {'force',0};
[ur,uz] = AxiStress(Mesh,E,nu,f,gD,gN);
```

With this solution the original and deformed mesh can be displayed, leading to the left part of Figure 43.

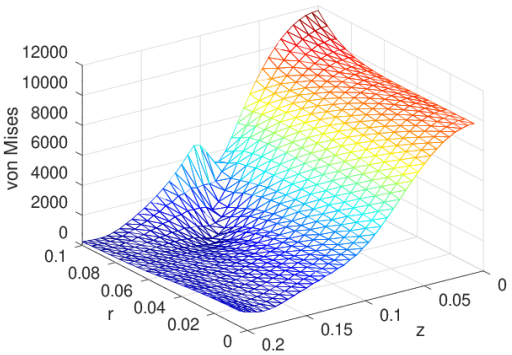
AxiSymmetricExample.m

```
factor = 0.1*R/max(sqrt(ur.^2+uz.^2));
figure(1); ShowDeformation(Mesh,ur,uz,factor);  xlabel('r'); ylabel('z'); axis equal;
```

With the displacements u_r and u_z evaluate stresses by using the functions `EvaluateStressAxi()` and `EvaluateVonMisesA`



(a) the original and deformed domain



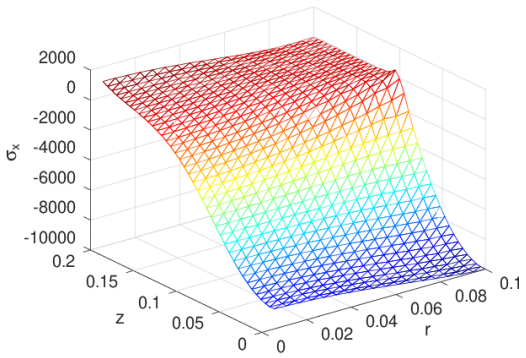
(b) the von Mises stress

Figure 43: The original and deformed domain and the von Mises stress for an axially symmetric setup

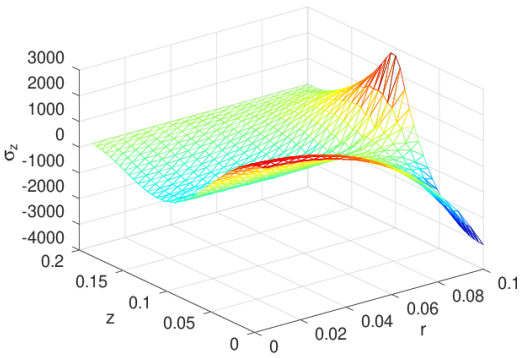
AxiSymmetricExample.m

```
[sigma_x,sigma_y,sigma_z,tau_xz] = EvaluateStressAxi(Mesh,ur,uz,E,nu);
figure(12); FEMtrimesh(Mesh,sigma_x)
            xlabel('r'); ylabel('z'); zlabel('\sigma_x')
figure(13); FEMtrimesh(Mesh,sigma_y)
            xlabel('r'); ylabel('z'); zlabel('\sigma_y')
figure(14); FEMtrimesh(Mesh,sigma_z)
            xlabel('r'); ylabel('z'); zlabel('\sigma_z')

vonMises = EvaluateVonMises(sigma_x,sigma_y,sigma_z,tau_xz);
figure(15); FEMtrimesh(Mesh,vonMises)
            xlabel('r'); ylabel('z'); zlabel('von Mises'); view(-125,30])
```



(a) the normal stress σ_x



(b) the nomal stress σ_z

Figure 44: Stresses for an axially symmetric setup

4 The Commands of FEMoctave

In this section find the documentation for the commands provided by FEMoctave. A considerable part of this documentation is also available by using the command `help` within *Octave*, e.g. `help BVP2D` .

4.1 Commands for 2D meshes: creation and modification

`FEMoctave` provides a few commands to generate and modify meshes, see Table 4. In this section the commands are briefly explained and elementary examples provided.

command	purpose
`CreateMeshRect()`	create a rectangular mesh
`CreateMeshTriangle()`	create a mesh by `Triangle`
`ReadMeshTriangle()`	read a mesh created by `Triangle`
`MeshAddConstraint()`	add a constraint to one node in a mesh
`MeshUpgrade()`	transform a mesh of order 1 to order 2, 3 or Hermite
`MeshQuad2Linear()`	transform a mesh of order 2 to order 1
`MeshCubic2Linear()`	transform a mesh of order 3 to order 1
`MeshHermite2Linear()`	transform a Hermite mesh and solution to a mesh of order 1
`Delaunay2Mesh()`	translate a Delaunay grid to a mesh of order 1
`MeshDeform()`	deform a given mesh

Table 4: Commands to create and modify meshes

4.1.1 Structure of a mesh

The main information of a mesh, as shown in Section 6.1 is given by the position of the nodes (points), the corresponding triangles and the boundary edges. A mesh consists of

Nn	nodes, with their (x, y) coordinates,
Ne	elements, with 3 (or 6, or 10) nodes forming one triangle,
Nb	boundary edges, with 2 (or 3, or 4) nodes forming one edge.

In FEMoctave this information is stored as a structure with an arbitrary name, but the elements of the structure require specific names, as shown in Table 5. The first 6 of these elements can be modified by the user and contain all the necessary information on the mesh to be used.

- `type`: a string indicating the order of the element, currently `linear`, `quadratic` or `cubic`. The algorithms will read this information and construct element stiffness matrices of the correct order.
- `nodes`: this $Nn \times 2$ matrix contains the coordinates (x_i, y_i) of the nodes numbered by $1 \leq i \leq Nn$. The entries are real numbers.
- `nodesT`: this Nn vector of integers contains the information of the type of nodes. If the entry in row i equals 0 then node i is a DOF, i.e. the value of the solution is not prescribed. If the entry in row i equals 1 then node i is a Dirichlet node and the value of the solution is determined by the given function. For elasticity problems this is a $Nn \times 2$ matrix with the information on both components.
- `elem`: for first order meshes this $Ne \times 3$ matrix of integers contains in each row the numbers of three nodes forming one linear element (triangle). The triangles have a positive orientation. For second order elements it is a $Ne \times 6$ matrix of integers. For third order elements it is a $Ne \times 10$ matrix of integers.
- `elemT`: type of elements is not used yet.
- `edges`: this $Nb \times 2$, $Nb \times 3$ or $Nb \times 4$ matrix of integers contains in each row the numbers of two, three or four nodes forming a boundary edge.

- `edgesT`: this Nb vector of integers contains the information of the type of edges. If the entry in row i equals -1 then edge i is part of the Dirichlet boundary, i.e. the value of the solution is prescribed. If the entry in row i equals -2 then edge i is part of the Neumann boundary, i.e. the value of the solution is not yet known. For elasticity problems this is a $Nb \times 2$ matrix with the information on both components. See Table 6 for the codes.

Name	Size	Information
`type`	string	type of element, "linear", "quadratic" or "cubic"
`nodes`	$Nn \times 2$	coordinates of nodes
`nodesT`	$Nn \times \{1, 2\}$	type of nodes, for scalar either 1(free) or 1 (fixed)
		for elasticity see Table 6
`elem`	$Ne \times \{3, 6, 10\}$	list of nodes that make up the triangles
	$Ne \times 3$	for first order elements
	$Ne \times 6$	for second order elements
	$Ne \times 10$	for third order elements
`elemT`	$Ne \times 1$	type of elements
`edges`	$Nb \times \{2, 3, 4\}$	list of nodes that make up the boundary edges
`edgesT`	$Nb \times \{1, 2\}$	type of boundary edge, Dirichlet, Neumann or elasticity
`elemArea`	$Ne \times 1$	area of the triangles
`GP`		coordinates of the Gauss integration points
	$3 \cdot Ne \times 2$	for first order elements
	$7 \cdot Ne \times 2$	for second and third order elements
`GPT`	$(3 \text{ or } 7) \cdot Ne \times 1$	type of the Gauss integration points
`nDOF`	$1 \times \{1, 2\}$	total number of DOF of the system
`node2DOF`	$Nn \times \{1, 2\}$	renumbering from nodes to DOF

Table 5: Elements of a mesh structure

All other elements of a mesh structure can be derived or computed from the above data.

- `elemArea`: this vector of real numbers contains the area of the individual triangles.
- `GP`: this matrix of reals contains the coordinates of all Gauss points for the numerical integration. There are 3 (or 7) Gauss points for each triangle.
- `GPT`: this vector of integer contains the type for each Gauss point. Currently not used.
- `nDOF`: this integer gives the total number of degrees of freedom (DOF) for the system to be solved.
- `node2DOF`: This vector (or matrix for elasticity problems) gives for each node the number of the corresponding DOF. If the number equals 0 then it is a Dirichlet node.

The commands `CreateMeshRect()` and `CreateMeshTriangle()` create meshes with this structure.

The codes for the boundary conditions in Table 6 for elasticity problems might ask for a few examples of boundary conditions.

-11 : at this node the displacements are given by $u_1(x, y) = gD1(x, y)$ and $u_2(x, y) = gD2(x, y)$.

-22 : at this node there are no surface forces, i.e. the node is on a free section of the boundary.

-12 : at this node the x–displacement $u_1(x, y) = gD1(x, y)$ is given and there is no surface force in y–direction.

-31 : at this node the y–displacement $u_2(x, y) = gD2(x, y)$ is given and surface force in x–direction is given by $gN1(x, y)$.

-23 : at this node there is no surface force in x–direction and the surface force in y–direction is given by $gN2(x, y)$.

code	for scalar problems	
-1	Dirichlet condition , $u = g_1$ given	
-2	Neumann condition , $a \frac{\partial}{\partial n} u = g_2 + g_2 u$	
	for elasticity problems	
code	in x–direction	in y-direction
-1*	displacement $u_1 = gD_1$ given	*
-*1	*	displacement $u_2 = gD_2$ given
-2*	force free section	*
-*2	*	force free section
-3*	force density gN_1 given	*
-*3	*	force density gN_2 given

Table 6: Codes for the boundary conditions

The structure of a mesh with Hermite elements is slightly different. The first three and the tenth entry in an element point to a node, entries four through 9 point to the other DOF, i.e. the partial derivatives at the three corners of the triangle. See Section 6.7.

4.1.2 Create a uniform mesh on a rectangle: `CreateMeshRect()`

With the command `CreateMeshRect(x,y,Blow,Bup,Bleft,Bright)` you can create a mesh on a rectangle. The function takes 6 input arguments.

- The ordered vectors `x` and `y` contain the x and y coordinates of the mesh to be generated.
- For scalar problems the variables `Blow`, `Bup`, `Bleft` and `Bright` indicate the boundary condition on the corresponding edges. If the index is -1 then the edge is part of the Dirichlet boundary Γ_1 and thus the value of the function is prescribed. If the index is -2 then the edge is part of the Neumann boundary Γ_2 and thus information about the outer normal derivative is known, but not the value of the solution.
- For elasticity problems the variables `Blow`, `Bup`, `Bleft` and `Bright` indicate the boundary condition according to the codes in Table 6.

Examples of the usage are given in Sections 3.1.1 and 3.1.2.

CreateMeshRect()

```
Mesh = CreateMeshRect(X,Y,BLOW,BUP,BLEFT,BRIGHT)

Create a rectangular mesh with nodes at (x_i,y_j)  with linear elements

parameters:
  * X,Y are the vectors containing the coodinates of the nodes to be generated.
  * BLOW, BUP, BLEFT, BRIGHT indicate the type of boundary condition at lower,
    upper, left and right edge of the rectangle
    * for scalar problems
      * B* = -1: Dirichlet boundary condition
      * B* = -2: Neumann or Robin boundary condition
    * for elasticity problems
      * bi = -xy : with two digits for x and y directions
      * x/y = 1 : given displacement
      * x/y = 2 : force free
      * x/y = 3 : given force density

return values
  * MESH is a a structure with the information about the mesh.
    The mesh consists of n_e elements, n_n nodes and n_ed edges.
    * MESH.TYPE a string with the type of triangle: linear
    * MESH.ELEM n_e by 3 matrix with the numbers of the nodes forming triangular elements
```

```
 * MESH.ELEMAREA n_e vector with the areas of the elements
 * MESH.ELEMT n_e vector with the type of elements (not  used)
 * MESH.NODES n_n by 2 matrix with the coordinates of the nodes
 * MESH.NODEST n_n vector with the type of nodes
 * MESH.EDGES n_ed by 2 matrix with the numbers of the nodes forming edges
 * MESH.EDGEST n_ed vector with the type of edge
 * MESH.GP n_e*3 by 2 matrix with the coordinates of the Gauss points
 * MESH.GPT n_e*3 vector of integers with the type of Gauss points
 * MESH.NDOF number of DOF, degrees of freedom
 * MESH.NODE2DOF n_n vector or n_n by 2 matrix of integers, mapping nodes to DOF

  Sample call:
  Mesh = CreateMeshRect(linspace(0,1,10),linspace(-1,2,20),-1,-1,-2,-2)
         will create a mesh with 200 nodes and 0<=x<=1, -1<=y<=+2
```

With `CreateMeshRect()` generate meshes with elements of order 1. With the help of `MeshUpgrade()` (Section 4.1.6) you can upgrade to the same mesh with elements of order 2 or 3, or with Hermite elements.

4.1.3 Using triangle: `CreateMeshTriangle()` and `ReadMeshTriangle()`

With the command `CreateMeshTriangle(name,xy,area)` you can create a mesh with the outer borders given in `xy`. The mesh will satisfy a minimal angle condition of $30°$ to avoid distorted triangles. The function takes 3 or 4 input arguments.

- The string `'name'` is the file name to be used to store the information.
- The matrix `xy` contains the edge points of the domain and the information on the boundary conditions.
- `area` is the typical are of the triangles to be used.
- The optional argument `options` can specify more flags to the external call of the program `triangle`.

The mesh can then be read by calling `Mesh = ReadMeshTriangle('name.1')`. Examples of the usage are given in Sections 3.1.3 and 3.2 and in many of the examples in Section 9 starting on page 243.

CreateMeshTriangle()

```
MESH = CreateMeshTriangle(NAME,XY,AREA,OPTIONS)

Generate files with a mesh with linear elements using the external code triangle

parameters:
  * NAME the base filename: the file NAME.poly will be generated then
    triangle will generate files NAME.1.* with the mesh
  * XY vector containing the coordinates of the nodes forming the outer boundary.
    The last given node will be connected to the first given node to create a closed curve.
    The format for XY is [x1,y1,b1;x2,y2,b2;...;xn,yn,bn] where
      * xi x-coordinate of node i
      * yi y-coordinate of node i
      * bi boundary marker for segment from node i to node i+1
        * for scalar problems
          * bi = -1 Dirichlet boundary condition
          * bi = -2 Neumann or Robin boundary condition
        * for elasticity problems
          * B* = -xy : with two digits for x and y directions
            * x/y = 1 : given displacement
            * x/y = 2 : force free
            * x/y = 3 : given force density
   * AREA the typical area of the individual triangles to be used
   * OPTIONS additional options to be used when calling triangle.
     The options "pa" and the area will be added automatically.
     Default options are "q", resp. "qpa". To suppress the verbose information use "Q"

     More options are available to adapt mesh sizes and create holes.
```

```
        See the documentation in FEMdoc.pdf

  The information on the mesh generated is written to files and returned
  in the structure MESH, if the return argument is provided.

    * The information can then be read and used by Mesh = ReadMeshTriangle('NAME.1');
    * MESH is a structure with the information about the mesh.
      The mesh consists of n_e elements, n_n nodes and n_ed edges.
      * MESH.TYPE a string with the type of triangle: linear, quadratic or cubic
      * MESH.ELEM n_e by 3 (or 6/10) matrix with the numbers of the nodes
        forming triangular elements
      * MESH.ELEMAREA n_e vector with the areas of the elements
      * MESH.ELEMT n_e vector with the type of elements (not used)
      * MESH.NODES n_n by 2 matrix with the coordinates of the  nodes
      * MESH.NODEST n_n vector with the type of nodes (not used)
      * MESH.EDGES n_ed by 2 (or 3/4) matrix with the numbers of the nodes forming edges
      * MESH.EDGEST n_ed vector with the type of edge
      * MESH.GP n_e*(3/7) by 2 matrix with the coordinates of the Gauss points
      * MESH.GPT n_e*(3/7) vector of integers with the type of Gauss points
      * MESH.NDOF number of DOF, degrees of freedom
      * MESH.NODE2DOF n_n vector of integer, mapping nodes to DOF

  Sample call:
  Mesh = CreateMeshTriangle('Test',[0,-1,-1;1,-1,-2;1,2,-1;0,2,-2],0.01)
  will create a mesh with 0<=x<=1, -1<=y<=+2 and a typical area of 0.01 for each triangle
  Could be read by Mesh = ReadMeshTriangle('Test.1')
```

- With `CreateMeshTriangle()` generate meshes with elements of order 1. With the help of the command `MeshUpgrade()` (Section 4.1.6) you can upgrade to the same mesh with elements of order 2 or 3, or with Hermite elements.
- If a return argument for `CreateMeshTriangle()` is provided, the mesh is returned.
- If no return argument is provided, the information is written to files. The generated mesh is then read by calling the function `ReadMeshTriangle()`.

This function can also be used to read meshes generated by direct call of the external program `triangle`. This allows to use all features of `triangle` and not only the very restricted setup used by `CreateMeshTriangle()`. In Section 4.1.4 find the options to generate meshes with holes and adapted mesh sizes. To find more about the features of `triangle` use the web page www.cs.cmu.edu/∼quake/triangle.html or compile and install the code and then run `triangle -h` to examine the built-in help.

ReadMeshTriangle()

```
  FEMMESH = ReadMeshTriangle(NAME.1)
    read a mesh generated by CreateMeshTriangle(NAME)
    parameter: NAME.1 the filename
    return value: FEMMESH the mesh stored in NAME

    Sample call:
    CreateMeshTriangle('Test',[0,-1,-1;1,-1,-2;1,2,-1;0,2,-2],0.01)
    Mesh = ReadMeshTriangle('Test.1');
           will create a mesh with 0<=x<=1, -1<=y<=+2
           and a typical area of 0.01 for each triangle
```

Find an example in Section 9.14.

With `CreateMeshTriange()` and `ReadMeshTriangle()` one can only generate meshes with elements of order 1. With the help of `MeshUpgrade()` (see Section 4.1.6) you can upgrade to the same mesh with elements of order 2 or 3, or with Hermite elements.

4.1.4 Adapting meshes and creating holes by using options of `CreateMeshTriangle()`

Give `CreateMeshTriangle()` more arguments to use some of the features of `Triangle` to locally create finer meshes or generate domains with holes. If more than 4 arguments are provided, then line segments, point with mesh sizes or holes can be generated. Each of the additional arguments is a structure with a name as first entry. There are four types of options:

- **Segment**: to create additional line segments, used to modify mesh sizes. The mandatory entry `name='Segment'` has to be supplemented with one additional entry `border`. It lists the x and y coordinates of the points forming the segment, and the third entry `0` for each point.

```
Seg1.name = 'Segment';               %% mandatory name
Seg1.border = [0 0 0; 1 0 0; 1 2 0]  %% the points on the segment
```

- **MeshSize**: to specify the mesh size in one part of the domain. Besides the entry `name='MeshSize'` two additional entries have to be provided. `where` with the x and y coordinates of the points where the maximal mesh size is given and `area` with the desired mesh size, i.e. the maximal area of the triangles.

```
Point1.name = 'MeshSize';   %% mandatory name
Point1.where = [1.5 0.2];   %% the point at which the mesh size is applied
Point1.area  = 0.01;        %% maximal area in the selected area
```

- **Hole**: to create a hole in the domain. Besides the mandatory entry `name='Hole'` two additional entries `border` and `point` have to be provided. `border` lists the x and y coordinates of the points forming the hole, with the third entry indicating the type of boundary condition, according to Table 6. The entry `point` has to contain the coordinates of one point inside the hole.

```
Hole1.name   = 'Hole'                                %% mandatory name
Hole1.border = [1 1 -22; 3 2 -22; 3 4 -22; 1 2 -22] %% border of the hole
Hole1.point  = [1.1 1.1];                           %% one point in the hole
```

- **Option**: to give more options, not documented yet.

There are a few points to watch out for when using the above optional arguments to `CreateMeshTriangle()`:

- The lines created by `Segment` shall not interfere with the holes.
- If `Segment` is used to divide the domain into multiple sections, the endpoints of the segments have to be exactly on the borders of the domain, but you can (often) not use the points defining the borders of the domain. A possible way out is described in Example 4–2 below. Section 9.41 contains an example with a refined mesh in a critical area.

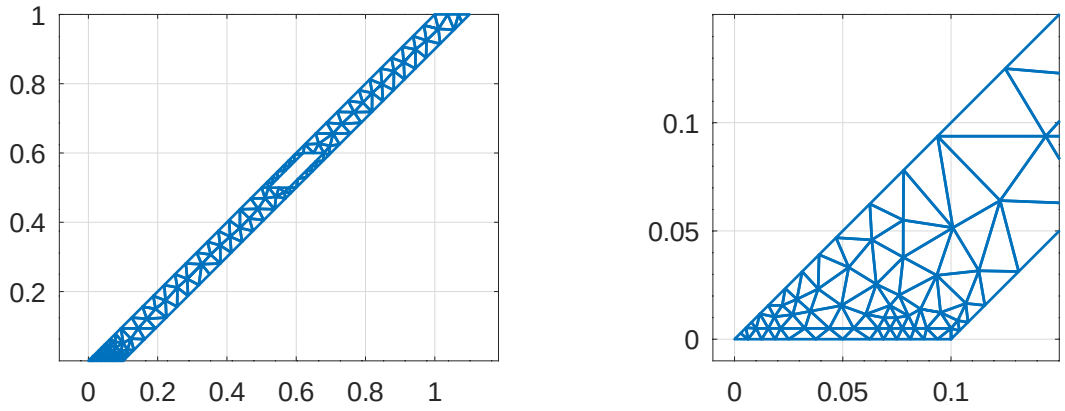
4–1 Example : As a first example for the additional options for the command `CreateMeshTriangle()` examine a domain with a hole in the middle section and a finer mesh at the lower edge. Find the result of the code below in Figure 45. In most parts of the domain the area of the triangles is approximately 0.001. Close to the hole the narrow section leads to smaller triangle and close to the lower edge the optional `Segment` leads to a finer mesh, visible in Figure 45(b).

```
MeshBorder = [0 0 -11; 0.1 0 -22; 1.1 1 -23; 1 1 -22]; %% outer boundary of the domain

Hole.name   = 'Hole';   %% a hole in the middle section
Hole.border = [0.5+0.02 0.5 -22; 0.5+0.08 0.5 -22;0.6+0.08 0.6 -22; 0.6+0.02 0.6 -22];
Hole.point  = [0.522 0.501];

Segment.name   = 'Segment';    %% close to the lower edge
Segment.border = [0.01 0.01 0; 0.09 0.01 0];

Mesh = CreateMeshTriangle('Mesh1',MeshBorder,0.01/9, Hole, Segment);
figure(1); FEMtrimesh(Mesh); axis equal
```



(a) the domain with a hole (b) the finer mesh at the lower edge

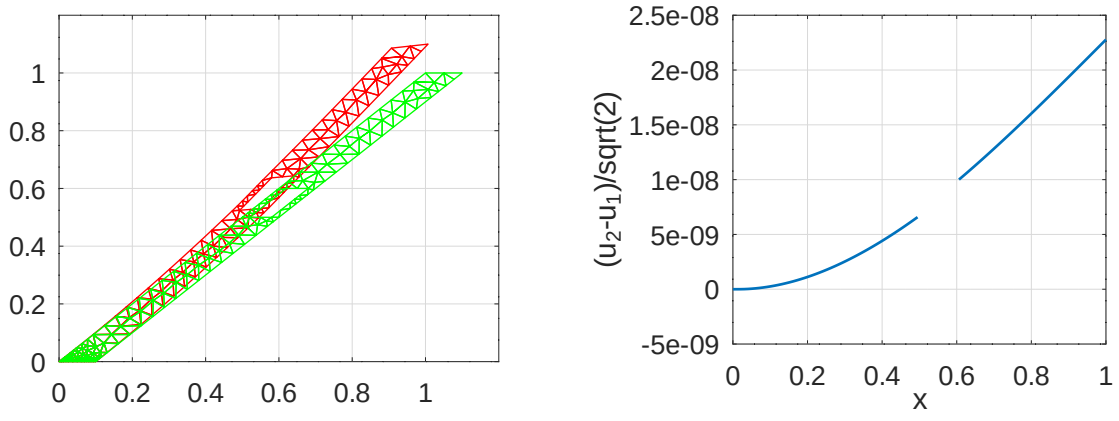
Figure 45: A domain with a hole and a finer mesh at the lower edge

The above domain can be used for an elasticity computation. The lower edge is fixed and at the upper edge a vertical force is applied. At the lower edge Saint–Venant's principle applies, i.e. shearing is expected and thus a finer mesh should be used. Find the graphical result in Figure 46. In addition the transversal deflection $(u_2 - u_1)/\sqrt{2}$ along the center line is displayed. The result shows that the lever is behaving like a bending beam.

```
Mesh = MeshUpgrade(Mesh,'quadratic');  E = 100e9; nu = 0.3; f = 1;
[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{0,0},{0,f});

figure(2);clf; scale = 0.1/max(u2);
ShowDeformation(Mesh,u1,u2,scale); axis([0 1.2 0 1.2])

yi = linspace(0,1); xi = yi+0.05;
u1i = FEMgriddata(Mesh,u1,xi,yi); u2i = FEMgriddata(Mesh,u2,xi,yi);
bend = (u2i-u1i)/sqrt(2);
figure(3); plot(yi,bend);  xlabel('x'); ylabel('(u_2-u_1)/sqrt(2)')
```



(a) the deformed lever (b) bending of the center line

Figure 46: The deformed lever and the bending of the center line

♢

4–2 Example : In this example a mesh with different size triangles is created on a unit square, see Figure 47. The first section of the code below generates the file `Mesh2.poly`, in which the two corners $(0,0)$ and $(1,1)$ are listed twice. This causes FEMoctave to issue a warning `matrix singular to machine precision` when solving the resulting linear system. To avoid this problem edit the file `Mesh2.poly`, remove the two "extra" points and modify the connection of points 1 and 3.

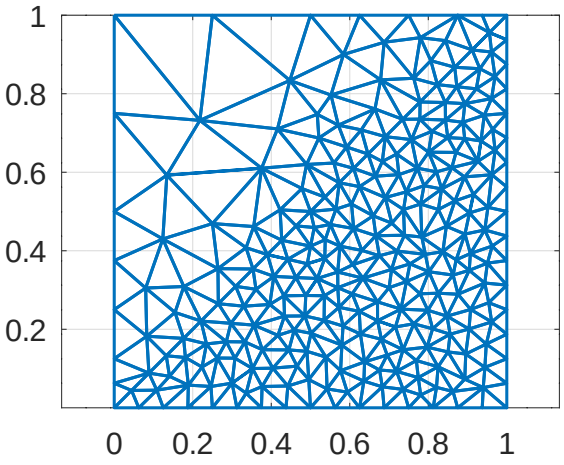


Figure 47: A domain with a two different mesh sizes

Mesh2.poly

```
# nodes
6 2 0 1
1   0.000000000000   0.000000000000 -1
2 1.000000000000000e+00 0.000000000000000e+00 -1
3 1.000000000000000e+00 1.000000000000000e+00 -2
4 0.000000000000000e+00 1.000000000000000e+00 -1
5 0.000000000000000e+00 0.000000000000000e+00 0
6 1.000000000000000e+00 1.000000000000000e+00 0
# segments
5 1
1 1 2 -1
2 2 3 -2
3 3 4 -2
4 4 1 -1
5 5 6 0
# holes
0
# area markers
2
1 0.100000 0.900000 0 0.100000
2 0.900000 0.100000 0 0.002000
# generate mesh by : triangle -Qpq30a Mesh2.poly
```

Save the new file as `Mesh2_mod.poly` and run triangle, either by `triangle -pq30a Mesh2_mod.poly` from a command line or by `system('triangle -Qpq30a Mesh2_mod.poly')` within *Octave*.

Mesh2_mod.poly

```
# nodes
4 2 0 1
1   0.000000000000   0.000000000000 -1
2 1.000000000000000e+00 0.000000000000000e+00 -1
3 1.000000000000000e+00 1.000000000000000e+00 -2
4 0.000000000000000e+00 1.000000000000000e+00 -1
# segments
5 1
```

```
1 1 2 -1
2 2 3 -2
3 3 4 -2
4 4 1 -1
5 1 3 0
# holes
0
# area markers
2
1 0.100000 0.900000 0 0.100000
2 0.900000 0.100000 0 0.002000
# generate mesh by : triangle -Qpq30a Mesh2_mod.poly
```

Then read the new mesh by `Mesh = ReadMeshTriangle('Mesh2_mod.1')`. The resulting mesh avoids the *Octave* warning. ♢

4.1.5 Adding constraints to a node in the mesh

On occasion it is convenient to add additional constraints at single points in the mesh. For this task the `FEMoctave` command `MeshAddConstraint()` can be used. The command

- finds the node in the mesh with the coordinates closest to the given position.
- adds the desired constraint at the selected node.

MeshAddConstraint()

```
MESH = MeshAddConstraint(MESH,POSITION,MODE)

apply an additional constraint

parameters:
  * MESH is the mesh describing the domain
  * POSITION coordinates of the node, may be approximate
  * MODE mode of the node with the additional constraint
    * MODE = -1: fixed value for scalar problems
    * MODE = [-1,-1]: fixed x and y displacements for elasticity
    * MODE = [-1,-2]: fixed x-displacement for elasticity
    * MODE = [-2,-1]: fixed y-displacement for elasticity

return values:
  * MESH the new mesh with the additional constraints
```

Find examples in Sections 5.17.1, 5.17.2 and 9.41.

4.1.6 Converting meshes: upgrading and downgrading

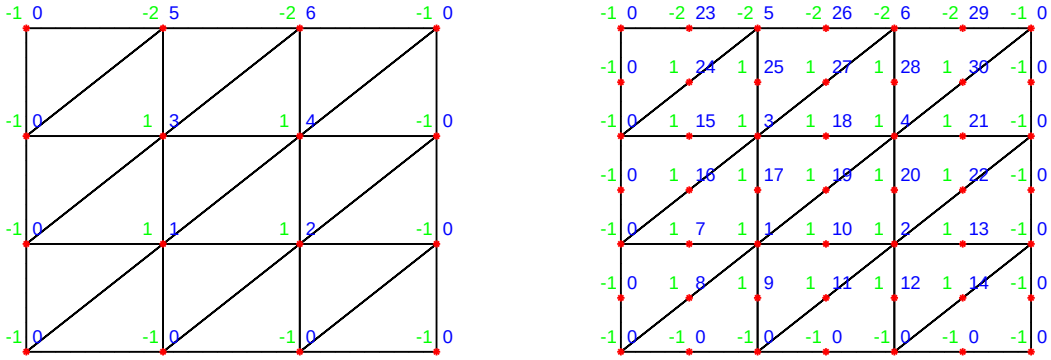
Given a mesh `MeshLin` with first order elements one can generate the same mesh with elements of order 2 by calling the command `MeshUpgrade(MeshLin,'quadratic')`. The numbering of the nodes of the linear elements is preserved in the mesh with the quadratic elements. The new nodes are placed at the midpoints of the edges of the triangles. With `MeshUpgrade(MeshLin,'cubic')` a mesh with 10 node cubic elements is generated. Examine Figure 91 on page 164 on how the nodes are placed within the triangles.

MeshUpgrade()

```
MESHNEW = MeshUpgrade(MESHLIN,TYPE)
     convert a mesh MESHLIN of order 1 to a mesh MESHNEW of order 2, 3 or Hermite
     parameters:
        * MESHLIN the input mesh of order 1
        * TYPE is a string, either 'quadratic', 'cubic' or 'Hermite'
          the default is 'quadratic'
     return value: MESHNEW the output mesh of order 2, 3 or Hermite
```

As example generate a mesh with elements of order 1 on the rectangle $0 \le x, y \le 2$ with Dirichlet conditions on three edges and a Neumann condition on the upper edge at $y = 2$. In Figure 48 find the mesh with the types of nodes indicated and the numbering of the resulting degrees of freedom.

```
N = 3;
FEMmesh1 = CreateMeshRect(linspace(0,2,N+1),linspace(0,2,N+1),-1,-2,-1,-1);
FEMmeshQ = MeshUpgrade(FEMmesh1,'quadratic');
```



(a) linear elements (b) quadratic elements

Figure 48: The same mesh with linear or quadratic elements. The types of the nodes are marked in green. Dirichlet nodes are marked by -1, Neumann nodes by -2 and interior nodes by $+1$. The numbering of the resulting degrees of freedom is shown in blue. For Dirichlet nodes a DOF of 0 is used.

Using `MeshQuad2Linear()` convert a mesh of order 2 to a mesh of order 1. The nodes will remain unchanged, but there will be a factor of 4 more elements. With this function one can compare results based on first or second order elements, using exactly the same degrees of freedom.

MeshQuad2Linear()

```
MESHLIN = MeshQuad2Linear(MESHQUAD)
     convert a mesh MESHQUAD of order 2 to a mesh MESHLIN of order 1
     parameter: MESHQUAD the input mesh of order 2
     return value: MESHLIN the output mesh of order 1
```

An example is shown in Section 3.1.4.

Using `MeshCubic2Linear()` convert a mesh of order 3 to a mesh of order 1. The nodes will remain unchanged, but there will be a factor of 9 more elements. With this function one can compare results based on first or third order elements, using exactly the same degrees of freedom.

MeshCubic2Linear()

```
MESHLIN = MeshCubic2Linear(MESHCUBIC)
     convert a mesh MESHCUBIC of order 3 to a mesh MESHLIN of order 1
     parameter:    MESHCUBIC the input mesh of order 3
     return value: MESHLIN the output mesh of order 1
```

Using `MeshHermite2Linear()` convert a mesh of Hermite elements and the DOF of a solution to a mesh of order 1 and the values of the solution at the new nodes.

MeshHermite2Linear()

```
[MeshLin,uLin]: = MeshHermite2Linear(MeshHermite,uHermite)

  convert a mesh MeshHermite to a mesh MeshLin of order 1

  parameters:
    * MeshHermite the input mesh with Hermite elements
    * uHermite the DOF of the solution of the BVP
```

```
  return values:
    * MeshLin the output mesh of order 1
    * uLin the values of the solution at the nodes
```

A combination of `MeshUpgrade()` and subsequent `MeshQuad2Liner()` or `MeshCubic2Liner()` can be used to refine meshes.

4.1.7 Use `delaunay()` to create a mesh: `Delaunay2Mesh()`

It is possible to use the *Octave* command `delaunay()` to generate a triangulation of a convex domain and then the command `Delaunay2Mesh()` to generate a mesh to be used by FEMoctave.

- The generated mesh consists of elements of order one. Use `MeshUpgrade()` to work with elements of order two or three.
- At first all boundary points are marked as Dirichlet points. Change the type description in the mesh if you want Neumann points.

Delaunay2Mesh()

```
FEMMESH = Delaunay2Mesh(TRI,X,Y)
   generate a mesh with elements of order 1, using a Delaunay triangulation
   parameters:
     * TRI the Delaunay triangulation
     * X,Y the coodinates of the points
   return value
     * FEMMESH is the mesh to be used by FEMoctave
```

Observe that the quality of the mesh might be very poor, e.g. triangles with very small angles. As example have a look at the upper edge on the right of the mesh in Figure 49. For almost all cases `triangle` will generate meshes of better quality. To generate the domain and the solution in Figure 49 use the code below.

TestDelaunay.m

```
[x,y] = meshgrid(linspace(-1,1,20));  x = x(:); y = y(:);
ind = find(y<1-0.5*x+0.001);  x = x(ind); y = y(ind);
ind = find(x+y>-0.001);       x = x(ind); y = y(ind);

tri = delaunay(x,y);
figure(1); triplot(tri,x,y); hold on; plot(x,y,'*'); hold off
           xlabel('x'); ylabel('y');
FEMmesh = Delaunay2Mesh(tri,x,y);  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

u = BVP2Dsym(FEMmesh,1,0,4,0,0,0);
figure(2); FEMtrimesh(FEMmesh,u)      xlabel('x'); ylabel('y'); view([100,45])
figure(3); FEMtricontour(FEMmesh,u);  xlabel('x'); ylabel('y');
```

4.1.8 Deforming meshes by `MeshDeform()`

With the function `MeshDeform()` the a linear mesh can be deformed by moving the nodes and recomputing the areas of the triangles.

MeshDeform()

```
MeshDeformed = MeshDeform(MESH,DEFORM)
  Deform the nodes of MESH by the transformation DEFORM
  parameters:
    * MESH the initial mesh with linear elements
      this has to be a mesh with linear elements
    * DEFORM the transformation formula
      the function DEFORM takes one argument XY, a n by 2 matrix with the
      x and y components in colums and returns the result in a n by 2 matrix.
 return value
    * DEFORMEDMESH the deformed mesh consistes of linear elements
      use MESHUPGRADE to generate quadratic or cubic elements
```

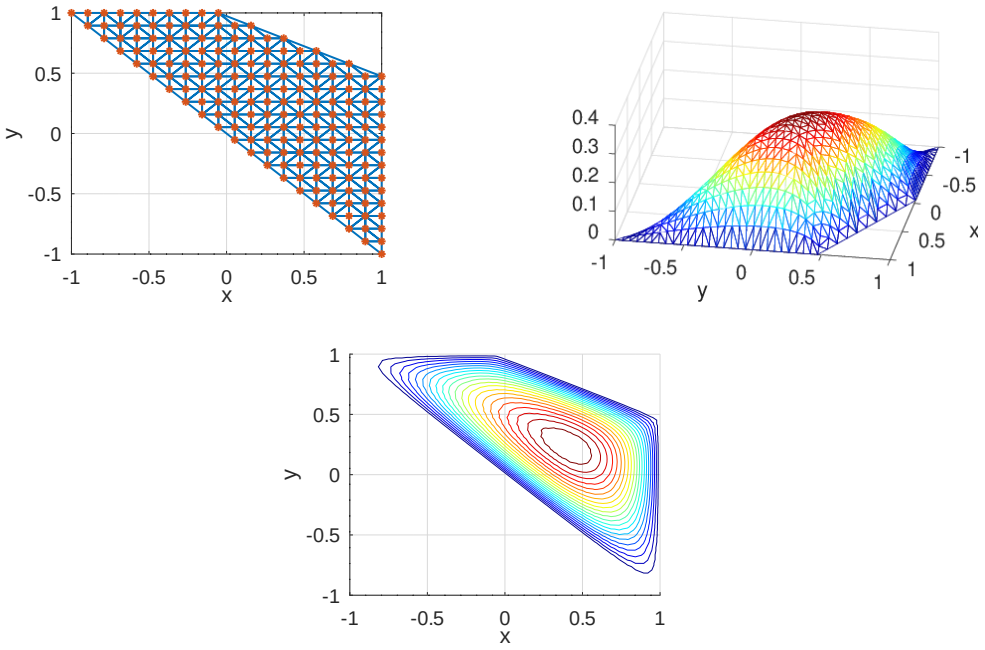


Figure 49: A mesh generated by a Delaunay triangulation and the solution of a BVP

One should pay attention to not deform the triangles in the mesh too badly by `MeshDeform()`, as this might decrease the accuracy of the solutions. The mesh generated by `MeshCreateTriangle()` will respect the condition of minimal 30° angles. After calling `MeshDeform()` this condition could be violated. Another option is to deform the borders of the mesh first, and then call `CreateMeshTriangle()`. In this case the minimal angle condition is respected.

To generate the quarter of a ring in Figure 64 on page 128 use polar coodinates

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \cdot \cos\varphi \\ r \cdot \sin\varphi \end{pmatrix} \quad \text{with} \quad 1 \le r \le 2 \quad \text{and} \quad 0 \le \varphi \le \frac{\pi}{2} .$$

```
FEMmesh = CreateMeshTriangle('Test',[1,0,-1;2,0,-1;2,pi/2,-2;1,pi/2,-1],0.1^2);
function xy_new = Deform(xy)  %% use polar coordinates
  xy_new = [xy(:,1).*cos(xy(:,2)), xy(:,1).*sin(xy(:,2))];
endfunction
FEMmesh = MeshDeform(FEMmesh,'Deform');
FEMtrimesh(FEMmesh)
```

Find examples in Sections 3.13.2, 9.1, 9.24.3, 9.43.1 and 9.54.3.

4.2 Evaluation and displaying results

Table 7 shows commands to display solutions and evaluate different expressions.

4.2.1 Display results on meshes, `FEMtrimesh()`, `FEMtrisurf()`, and `FEMtricontour()`

To display the results of the computations very elementary wrappers around `trimesh()`, `trisurf()` and `tricontour()` are provided.[10]

- With `FEMtrimesh()` display a function u as a 3D mesh. If no values for u are provided, the 2D mesh is displayed.

[10] It is obviously possible to improve the wrappers, as non of the advanced features of `trimesh()` or `trisurf()` is passed through. If you want to use those, have a look at the elementary code in the `FEMtri*` functions and copy the necessary lines into your code.

command	purpose
`FEMtrimesh()`	display a solution as mesh
`FEMtrisurf()`	display a solution as surface
`FEMtricontour()`	display the countour lines of a solution
`FEMEvaluateGradient()`	evaluate the gradient at the nodes
`FEMEvaluateGP()`	evaluate a function and the gradient at the Gauss points
`FEMIntegrate()`	integrate an expression over the domain
`FEMgriddata()`	evaluate at arbitrary points or along curves

Table 7: Commands to display and evaluate solutions

- With `FEMtrisurf()` display a function u as a 3D surface. The syntax is identical to `FEMtrimesh()`.
- With `FEMtricontour()` display level curves of a function u. The syntax similar to the above.

All functions accept meshes with linear, quadratic, cubic or Hermite elements.

- For quadratic elements the 6 nodes in each element are connected by straight lines, i.e. as if one second order triangle would be composed of 4 first order triangles.
- For cubic elements the 10 nodes in each element are connected by straight lines, i.e. as if one third order triangle would be composed of 9 first order triangles.
- For Hermite elements only the corners of the triangles are used. For better resolution first use the command `MeshHermite2Linear()` to convert the Hermite mesh and solution to a finer mesh with linear elements.

FEMtrimesh()

```
FEMtrimesh (MESH, U)
  display a solution U on a triangular mesh
  parameters:
    * MESH is the mesh
    * U values of the function to be displayed
      if U is not given, then the mesh is displayed in 2D
```

FEMtrisurf()

```
FEMtrisurf (MESH, U)
  display a solution U as surface on a triangular mesh
  parameters:
    * MESH is the mesh
    * U values of the function to be displayed
```

FEMtricontour()

```
FEMtricontour (MESH, U, V)
     display contours of a solution U on a triangular mesh
     parameters:
        * MESH is the mesh
        * U values of the function to be displayed
        * V contours to be used, default value is 21
          if V is scalar, it is the number of contours
          if V is a vector, it is the levels of the contours
```

4.2.2 Evaluate the gradient of a function at the nodes: `FEMEvaluateGradient()`

Given the values u of a function at the nodes, the two components of the gradient can be computed with the function `FEMEvaluateGradient()`.

FEMEvaluateGradient()

```
[UX,UY] = FEMEvaluateGradient(MESH,U)
     evaluate the gradient of the function u at the nodes
     parameters:
        * MESH is the mesh describing the domain and the boundary types
        * U vector with the values of the function at the node
     return value
        * UX x component of the gradient of u
        * UY y component of the gradient of u

     the values of the gradient are determined on each element
     at the nodes the average of the gradient of the elements is used
```

The gradient is determined on each of the elements, using either linear, quadratic or cubic interpolation. Then at each node the average of the values of the gradient of the neighboring triangles is returned. This is different from the results generated by `FEMgriddata()`. Examples are given in Sections 5.1, 9.3, 9.4, 9.9 and 9.15. Due to using broadcasting in the *Octave* code (`bsxfun()`) the code is fast! This function could be used (or is that abused?) to evaluate derivatives of functions given on an irregular grid! `FEMEvaluategradient()` can not be used for Hermite elements. Use `MeshHermite2Linear()` to convert to a linear mesh and solution.

4.2.3 Evaluate a function and its gradient at the Gauss points: `FEMEvaluateGP()`

Given the values u of a function at the nodes, the values of u and its gradient can be computed at the Gauss points by calling `FEMEvaluateGP()`. For first order elements a piecewise linear interpolation is used, thus the gradients will be constant on each triangular element. For second order elements a quadratic interpolation is used. For third order elements a cubic interpolation is used. For Hermite elements the corresponding cubic interpolation is used.

FEMEvaluateGP()

```
[UGP,GRADUGP] = FEMEvaluateGP(MESH,U)
     evaluate the function and gradient at the Gauss points
     parameters:
        * MESH is the mesh describing the domain and the boundary types
        * U vector with the values of at the nodes
     return values
        * UGP values of u at the Gauss points
        * GRADUGP matrix with the values of the gradients in the columns
```

Examples are given in Sections 9.13 and 9.15.

4.2.4 Integrate a function over the domain: `FEMIntegrate()`

Given a function name, the values of a function at the nodes or at the Gauss points one can integrate this function over the domain given by the mesh. There are different methods used, all based on the Gauss integration presented in Section 6.3.2.

- If a function name is specified, then this function will be evaluated at the Gauss points and then integrated.
- If a scalar value is given, then the function is assumed to be constant.
- If a column vector is given with as many components as nodes in the mesh, then an element wise interpolation is used to obtain the values at the Gauss points. The function `FEMEvaluateGP()` is used internally to find the values at the Gauss points.
- If a column vector is given with as many components as Gauss points in the mesh, then these are used as values at the Gauss points.
- For Hermite elements the values of the DOF can be given insted of the values at the nodes.

FEMIntegrate()

```
NUMINTEGRAL = FEMIntegrate(MESH,U)

  integrate a function u over the domain given in Mesh

  parameters:
   * MESH is the mesh describing the domain
   * U the function to be integrated, can be given as
     * function name to be evaluated
     * scalar value
     * vector with the values at the nodes or the DOF (Hermite)
     * vector with the values at the Gauss points.

  return value
     * NUMINTGERAL the numerical approximation of the integral
```

As a simple example integrate the function $u(x, y) = x\, y^3$ over the unit square $0 \le x, y \le 1$. The exact integral equals $\frac{1}{8}$, but you have to subtract the exact value to see the difference to the numerical evaluation with the Gauss points. This is not unusual, since the Gauss integration leads to very accurate approximations, if the function is smooth. Linear elements use 3 integration points in each triangle, quadratic and cubic meshes use 7 integration points in each triangle. Thus integrations using a linear mesh might not be as accurate.

```
N = 40; Mesh = CreateMeshRect(linspace(0,1,N),linspace(0,1,N),-2,-2,-2,-2);
function res = f_int(xy)  res = xy(:,1).*xy(:,2).^3; endfunction

integral1 = FEMIntegrate(Mesh,'f_int')  % using the function name
uGP = feval('f_int',Mesh.GP);
integral2 = FEMIntegrate(Mesh,uGP)      % using the values at the Gauss points
-->
integral1 =  0.12500      integral2 =  0.12500
```

To determine the area of a domain $\Omega \subset \mathbb{R}^2$ one can integrate the constant 1 over the domain. More examples are given in Sections 5.1, 9.1, 9.13 and 9.15.

4.2.5 Evaluation at arbitrary points or along curves, integration along curves: `FEMgriddata()`

Given a function by the values at the nodes of a mesh use the command `FEMgriddata()` to evaluate the function at arbitrary points.

- The value of the function and the partial derivatives can be evaluated.
- Depending on the mesh provided either a piecewise linear, quadratic or cubic interpolation is used.
- If a point (x_i, y_i) is on the edge of a triangle it is a matter of rounding which of the neighboring triangles is used for the interpolation. Since all elements used by FEMoctave are C^0 conforming, this has no influence on the value of the function. The elements are not C^1 conforming and thus the partial derivatives will jump across element boundaries. See also Section 5.3 starting on page 123.
- If a point (x_i, y_i) is not in a triangle, then `NaN` is returned.
- The evaluation is very fast, even for large numbers of elements and interpolation points.
- Evaluation along arbitrary curves is possible, and fast. Then use `trapz()` to integrate along curves. Find examples in Sections 9.9 and 9.25.

FEMgriddata()

```
[UI,UXI,UYI] = FEMgriddata(MESH,U,XI,YI)
  evaluate the function (and gradient) at given points by interpolation
  parameters:
    * MESH is the mesh describing the domain
      If MESH consists of linear elements, piecewise linear interpolation is used.
```

```
    If MESH consists of quadratic elements, piecewise quadratic interpolation is used.
    If MESH consists of cubic elements, piecewise cubic interpolation is used.
  * U vector with the values of the function at the nodes
  * XI, YI coordinates of the points where the function is evaluated
return values:
  * UI values of the interpolated function u
  * UXI x component of the gradient of u
  * UYI y component of the gradient of u

The values of the function and the gradient are determined on each element by
a piecewise linear, quadratic or cubic interpolation.
If a point is not inside the mesh NaN is returned.
```

This function is similar to `FEMEvaluateGradient()`, but allows to evaluate at arbitrary points. At the nodes the value of the gradient in **one of the triangles** is returned. As a consequence the results generated by the command `FEMEvaluateGradient()` look smoother on occasion. But if the derivatives of the function change severly at the edge of elements (e.g. change of material or domain of termal expansion) the results can be surprising or confusing. Thus the command `EvaluateStress()` and `EvaluateStrain()` have an optional use of `FEMgriddata()`. As application see Example 9.41, starting on page 370.

The code below evaluates a function on an L-shaped domain on a rectangular grid. Find the result in Figure 50.

```
nodes = [0,0,-2;1,0,-2;1,1,-2;-1,1,-2;-1,-1,-2;0,-1,-2];
Mesh = CreateMeshTriangle('Ldomain',nodes,0.002);
x = Mesh.nodes(:,1);   y = Mesh.nodes(:,2);

function res = f_int2(xy)  res = sin(pi*xy(:,1)).^2.*xy(:,2)+1; endfunction

u = feval('f_int2',Mesh.nodes);
N = 51;  [xi,yi] = meshgrid(linspace(-1,1,N));  %% generate the uniform grid
tic(); ui3 = FEMgriddata(Mesh,u,xi,yi); toc()

figure(1);  mesh(xi,yi,ui3)
            xlabel('x'); ylabel('y'); zlabel('u')
-->
Elapsed time is 0.0075829 seconds.
```

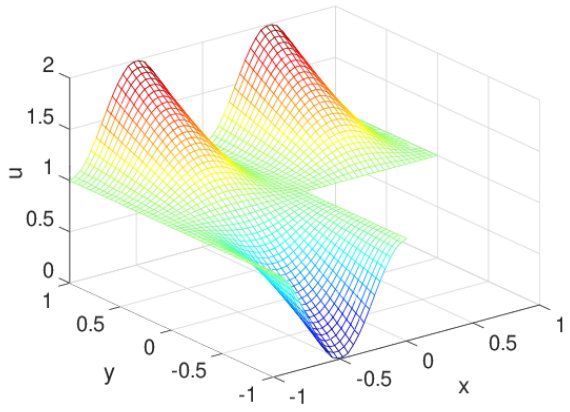


Figure 50: A function evaluated on a uniform grid

Examples are given in Sections 5.3, 9.3, 9.9, 9.14, 9.16 and 9.25.

4.3 How to define functions

There are three basic techniques to define functions in *Octave* to be used with FEMoctave.

- If the function is a constant you can simply use this scalar as input argument.
- You may provide the function name of the function to be called to compute the values of the function. Observe that the function **has to be vectorized**[11]. Due to a recent change in *Octave* the script versions should use a dummy second argument[12]. The function can be implemented as a `name.m` *Octave* function or as dynamically linked function `name.oct`, written in C++.
- You can provide a vector of the correct size with all the values of the function at the Gauss integration points of the mesh.

Section 9 contains many examples or you may examine the examples below.

4.3.1 Functions for static problems

The functions `BVP2D()`, `BVP2Dsym()` and `BVP2Deig()` accept the coefficient functions as input parameters. These functions accept (currently) one parameter, a matrix with two columns. The first (resp. second) column contains the x (resp. y) coordinates of the points at which the function is to be evaluated. For the coefficient a more options are available to work with nonisotropic material data (see Section 4.3.3 below) or complex coefficients.

As a first example consider the function $f(x, y) = 7$. There are three options:

1. Pass the constant 7 as scalar to the FEMoctave function. This is the preferred approach.
2. Define a function

   Octave
   ```
   function res = ff(xy,dummy)
     res = 7*ones(size(yz)(1),1);
   endfunction
   ```

   and then pass the string `'ff'` to the FEMoctave function.
3. Determine the vector of the correct size by

   Octave
   ```
   ffVec = 7 * ones(size(mesh.GP)(1),1);
   ```

   and then pass the vector `ffVec` to the FEMoctave function.

For the second example function

$$f(x, y) = 7 + 2\,x$$

the option *constant* is not applicable. There are two equally valid methods.

1. Define a function

   Octave
   ```
   function res = ff(xy,dummy)
     res = 7 + 2*xy(:,1);
   endfunction
   ```

   and then pass the string `'ff'` to the FEMoctave function.
2. Determine the vector of the correct size by

   Octave

[11] For scalar problems the functions on the boundary are actually called for one point at a time, but this might change. For elasticity problems the functions are called with multiple points. Thus it is advisable to write all functions vectorized.

[12] In the script files (`FEMEquation.m` and similar) the function is called with the node types as second argument, to be used for different sections in the domain. If you only use the compiled versions (`FEMEquation.oct` and similar) the dummy argument is not required. I might remove this "feature" in a next release.

```
ffVec = 7 + 2*xy(:,1);
```

and then pass the vector `ffVec` to the FEMoctave function.

To implement the function

$$f(x,y) = J_0(r) = J_0(\sqrt{x^2+y^2})$$

to be passed to the FEMoctave command use

Octave

```
function y = f(xy)
  y = besselj(0,sqrt(xy(:,1).^2+xy(:,2).^2));
endfunction
```

With this definition pass the string `'f'` to the FEMoctave function. Alternatively you can first compute the column vector `fVec` of this function at the Gauss points of the mesh by

Octave

```
fVec = f(mesh.GP);
```

and then pass the vector `fVec` to the FEMoctave function.

4.3.2 Functions for dynamic problems

The only change is the additional time t, to be passed as a second argument, i.e. `f(xy,t)=...` .

4.3.3 Functions for nonisotropic problems

The coefficient a in the differential expression can also be given as a symmetric matrix, i.e.

$$a \quad \longrightarrow \quad \begin{bmatrix} a_{xx} & a_{xy} \\ a_{xy} & a_{yy} \end{bmatrix}$$

Consequently there have to be more ways to provide this coefficient to the commands of `FEMoctave`. The parsing of this coefficient function is done in the commands `FEMEquation()`, `FEMEquationQuad()` and `FEMEquationCubic()`.

- For isotropic problems provide a scalar function.
  - given by a scalar $a \in \mathbb{R}$, e.g. by `a = 1` . Thus the coefficient function is a constant.
  - given by a string with the name of the function, e.g. `a = a_func`, where the function evaluates the coefficient function at the Gauss points.

    a_func.m

    ```
    function res = a_func(xy,dummy)
      x = xy(:,1); y = xy(:,2);
      res = x.^2 + y.^2;
    endfunction
    ```

  - given by a vector of the values of a at the Gauss points, e.g. `a = 2*ones(size(FEMmesh.GP,1),1)` .
- For non isotropic problems provide the three scalar functions for a_{xx}, a_{yy} and a_{xy}.
  - given by three scalar values, e.g. `a = [1 2 0.5]` leads to a constant coefficient matrix

    $$\mathbf{a} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 2 \end{bmatrix} .$$

  - given by a string with the name of the function, e.g. `a = a_func`, where the function evaluates the coefficient functions at the Gauss points. The coefficient matrix

    $$\mathbf{a} = \begin{bmatrix} 1+x^2 & x \cdot y \\ x \cdot y & 2+y^2 \end{bmatrix}$$

    is generated by

    a_func.m

```
function res = a_func(xy,dummy)
  x = xy(:,1); y = xy(:,2);
  res = [1+x.^2 2+y.^2 x.*y];
endfunction
```

– given by a vector of the values of a at the Gauss points as an $n \times 3$ matrix for n Gauss points in the given mesh, e.g. `a = ones(size(FEMmesh.GP,1),1)*[1 2 0.5]`.

4.4 Solving elliptic problems

The first few commands shown in Table 1 can be used to solve elliptic problem on a bounded domain $\Omega \subset \mathbb{R}^2$. In the next two sections the commands to solve a symmetric and a non-symmetric elliptic BVP are shown.

4.4.1 Symmetric elliptic problems: `BVP2Dsym()`

Equations given in the form (2)

$$\begin{aligned} -\nabla \cdot (a\,\nabla u) + b_0\,u &= f && \text{for} \quad (x,y) \in \Omega \\ u &= g_1 && \text{for} \quad (x,y) \in \Gamma_1 \\ a\,\frac{\partial u}{\partial n} &= g_2 + g_3\,u && \text{for} \quad (x,y) \in \Gamma_2 \end{aligned}$$

are solved by calling `BVP2Dsym()`

Octave

```
u = BVP2Dsym(mesh,a,b0,f,g1,g2,g3)
```

where the coefficient functions can be given as described in Section 4.3.1, as constants, strings or vectors. The return value `u` is a vector with the values of the solution at the nodes.

BVP2Dsym()

```
U = BVP2Dsym(MESH,A,B0,F,GD,GN1,GN2,OPTIONS)

Solve a symmetric, elliptic boundary value problem

    -div(a*grad u)+ b0*u = f            in domain
                      u = gD            on Dirichlet boundary
           n*(a*grad u) = gN1 + gN2*u   on Neumann boundary

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * The element type (linear, qudratic, cubic) is selected by MESH
  * A,B0,F,GD,GN1,GN2 are the coefficients and functions
    describing the PDE.
    Any constant function can be given by its scalar value.
    The functions A,B0 and F may also be given as vectors with the
    values of the function at the Gauss points.
  * The coefficient A can also be a symmetric matrix
    A=[axx,axy;axy,ayy] given by the row vector [axx,ayy,axy].  It
    can be given as row vector or as string with the function name
    or as nx3 matrix with the values at the Gauss points.
  * OPTIONS additional options, given as pairs name/value. Currently only real or complex
    coefficient problems can be selected by "TYPE" and the possible values are
    * "REAL": all coefficients are real (default)
    * "COMPLEX": some coefficients might be complex

return value
  * U is the vector with the values of the solution at the nodes
```

If at least one of the coefficients a, b_0, f or g_i leads to complex numbers, then it is required to use the option `'type'` with the value `'complex'`. Otherwise obtain wrong results.

Find examples in Sections 3.1.1, 3.1.2, 3.1.3, 9.4, 9.8, 9.9, 9.13 and 9.25. A Helmholtz equation with complex coefficients is solved in Section 9.21.

4.4.2 General elliptic problems: `BVP2D()`

Equations given in the form of (1)

$$\begin{aligned} -\nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y)\in\Omega \\ u &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned}$$

are solved by

Octave

```
u = BVP2D(mesh,a,b0,bx,by,f,g1,g2,g3)
```

where the coefficient functions can be given as described in Section 4.3.1, as constants, strings or vectors. The expressions `bx` and `by` denote the two components of the convection vector $\vec{b}$. The return value `u` is a vector with the values of the solution u at the nodes. Find an example in Section 3.1.4.

BVP2D()

```
U = BVP2D(MESH,A,B0,BX,BY,F,GD,GN1,GN2,OPTIONS)

Solve an elliptic boundary value problem

    -div(a*grad u - u*(bx,by))+ b0*u = f          in domain
                               u = gD          on Dirichlet boundary
          n*(a*grad u - u*(bx,by)) = gN1+gN2*u on Neumann boundary

parameters:
 * MESH is the mesh describing the domain and the boundary types
 * A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
   Any constant function can be given by its scalar value.
   The functions A,B0,BX,BY and F may also be given as vectors
   with the values of the function at the Gauss points.
 * The coefficient A can also be a symmetric matrix A=[axx,axy;axy,ayy]
   given by the row vector [axx,ayy,axy]. It can be given as row vector or as string
   with the function name or as nx3 matrix with the values at the Gauss points.
 * OPTIONS additional options, given as pairs name/value. Currently only real or complex
   coefficient problems can be selected by "TYPE" and the possible values are
   * "REAL": all coefficients are real (default)
   * "COMPLEX": some coefficients might be complex

return value
 * U is the vector with the values of the solution at the nodes
```

If at least one of the coefficients a, b_0, $\vec{b}$, f or g_i leads to complex numbers, then it is required to use the option `'type'` with the value `'complex'`. Otherwise obtain wrong results.

4.5 Solving 2D eigenvalue problems: `BVP2Deig()`

To solve an eigenvalue problem of the form (4)

$$\begin{aligned} -\nabla\cdot(a\,\nabla u) + b_0\,u &= \lambda\,f\,u && \text{for} \quad (x,y)\in\Omega \\ u &= 0 && \text{for} \quad (x,y)\in\Gamma_1 \\ a\,\frac{\partial u}{\partial n} &= g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned}$$

use

Octave

```
[Eval,Evec,errorbound] = BVP2Deig(mesh,a,b0,f,gN2,nVec);
```

where the coefficient functions can be given as described in Section 4.3.1, as constants, strings or vectors.

- The function can be called with one (`Eval`) or two (`[Eval,Evec]`) return arguments. A possible third return argument (`[Eval,Evec,errorbound]`) is of limited use, since with newer versions of FEMoctave `eigs()` is used, instead of an inverse power iteration.

  - The first return value `Eval` is a column vector containing the estimated values of the eigenvalues λ_i.
  - If the second return value `Evec` is asked for, then a matrix will be returned. Each column contains the values of a normalized eigenfunction at the nodes.
  - The third return argument `errorbound` will return a matrix with two columns, containing information on the error bound of the eigenvalues. Observe that the error of the eigenvalue computation is given, not the error of the overall FEM problem. The error of the FEM discretization has to be estimated by other tools. Some mathematical details are given in Section 6.11.
    * The first column contains a conservative error estimate. The actual error of the eigenvalue is guaranteed to be smaller.
    * The second column contains a more aggressive error estimate. Under most circumstances the estimate is valid. For highly clustered eigenvalues the error is overestimated.. There are circumstances when the error of the largest eigenvalues is underestimated. If the error is extremely small, the estimate might indicate an even smaller error. Keep in mind that the error is always larger than machine accuracy permits.

- The integer parameter `nVec` indicate the number of smallest eigenvalues to be be computed.
- There are three optional parameters, to be given as pairs name/value.

'tol' This parameter will lead to the iteration stopping if the relative change from one step to the next is smaller than `tol`. The default value of 'tol' is 10^{-5}.

'type' This parameter allows to select real or complex coefficients.
  * 'real': all coefficients are real valued. This is the default.
  * 'complex': some of the coefficients might be complex.

'Mode' Select the eigenvalues
  * 'sm': smallest magnitude, default
  * 'sa': smallest algebraic
  * 'lm': largest magnitude
  * `sigma`: closest to the scalar `sigma`
  * other selections are possible, see `help eigs`

Examples of eigenvalue problems are shown in Sections 3.2, 9.16.3, 9.23, 9.24.3 and 9.24.4.

BVP2Deig()

```
[EVAL,EVEC,ERRORBOUND] = BVP2Deig(MESH,A,B0,W,GN2,NVEC,OPTIONS)

determine eigenvalues EVAL and eigenfunctions EVEC for the BVP

      -div(a*grad u) + b0*u = Eval*w*u  in domain
                       u   = 0         on Dirichlet boundary
              n*(a*grad u)  = gN2*u on  Neumann boundary

parameters:
 * MESH the mesh describing the domain and the boundary types
 * A,B0,W,GN2 the coefficients and functions describing the PDE.
   Any constant function can be given by its scalar value.
   The functions A,B0 and W may also be given as vectors with the
   values of the function at the Gauss points.
 * The coefficient A can also be a symmetric matrix A=[axx,axy;axy,ayy]
   given by the row vector [axx,ayy,axy]. It can be given as row vector or as string
   with the function name or as nx3 matrix with the values at the Gauss points.
 * NVEC number of smallest eigenvalues to be computed
 * OPTIONS additional options, given as pairs name/value.
   * TOL tolerance for the eigenvalue iteration, default 1e-5
   * "TYPE" select real or complex coefficients
     * "REAL": all coefficients are real (default)
     * "COMPLEX": some coefficients might be complex
   * "MODE" select the eigenvalues
     * "SM": smallest magnitude, default
```

```
      * "SA": smallest algebraic
      * "LM": largest magnitude
      * SIGMA: closest to scalar sigma
      * see "help eigs" for more options

  return values:
   * EVAL is the vector with the eigenvalues
   * EVEC is the matrix with the eigenvectors as columns
   * ERRORBOUND is a matrix with error bounds of the eigenvalues
```

If at least one of the coefficients a, b_0 or w leads to complex numbers, then it is required to use the option `'type'` with the value `'complex'`. Otherwise obtain wrong results.

In Sections 6.10.2 and 6.10.5 find the consequences of the eigenvalues to solutions of dynamic heat and wave equations.

4.6 Solving nonlinear problems: `BVP2DNL()`

To solve a semilinear elliptic problem[13] of the form (5)

$$\begin{aligned} -\nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(xy,u) && \text{for} \quad (x,y)\in\Omega \\ u &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned}$$

use the command `BVP2DNL()`. Find Examples in Sections 3.1.6, 9.6 and 9.7.

BVP2DNL()

```
  [U,INFORM]  =  BVP2DNL(MESH,A,B0,BX,BY,F,GD,GN1,GN2,OPTIONS)

  solve a semilinear 2D boundary value problem

   -div(a*grad u - u*(bx,by))+ b0*u = f(xy,u)    in domain
                                  u = gD         on Dirichlet boundary
           n*(a*grad u - u*(bx,by)) = gN1+gN2*u  on Neumann boundary

  parameters:
   * MESH is the mesh describing the domain and the boundary types
   * A,B0,BX,BY,GD,GN1,GN2 are the coefficients and functions describing the PDE.
     Any constant function can be given by its scalar value.
     The functions A,B0,BX and BY and may also be given as vectors
     with the values of the function at the Gauss points.
   * F function handles to evaluate f(xy) or f(xy,u) and the partial derivative with respect to u.
     * F = @(XY) a function handle to evaluate f(xy) at the Gauss points.
     * F = {@(XY,U), @(XY,U)} assumes that the function f depends on x,y and u.
       The two function handles evaluate f(xy,u) and the partial derivative f_u(xy,u).
   * OPTIONS additional options, given as pairs name/value.
     * "TOL" the tolerance for the iteration to stop, given as pair [TOLREL,TOLABS] for the
       relative and absolute tolerance. The iteration stops if the absolute or relative error
       is smaller than the specified tolerance.  RMS (root mean square) values are used.
       If only TOLREL is specified TOLABS = TOLREL is used.
       The default values are TOLREL = TOLABS = 1E-5.
     * "MAXITER" the maximal number of iterations to be used. The default value is 10.
     * "DISPLAY" should information be displayed for the iterations
       * "OFF" no display, default
       * "ITER" display the number of the iteration and the RMS size of the update

  return values
    * U the values of the solution at the nodes
    * INFORM a structure with information on the performance of the algorithm
```

[13] Functions of the type $f(xy,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial y})$ are not implemented, but it is possible to do so. Complex coefficients are not implemented yet either. I have not run across an application. Let me know if you need this feature.

```
    * INFORM.INFO = 1 if the algorithm converged with the desired tolerance, -1 if not.
    * INFORM.ITER the number of iterations used.
    * INFORM.ABSERROR the RMS value of the last correction applied.
```

4.7 Solving parabolic problems: `IBVP2D()` and `IBVP2Dsym()`

To solve an initial boundary value problem (IBVP) of the form (6)

$$\begin{aligned} \rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \end{aligned}$$

use the command `IBVP2D()`. Find examples in Sections 3.3, 9.16 and a description of the algorithm in Section 6.10.1.

IBVP2D()

```
[U,T] =  IBVP2D(MESH,M,A,B0,BX,BY,F,GD,GN1,GN2,U0,T0,TEND,STEPS,OPTIONS)

solve an initial boundary value problem

  m*d/dt u - div(a*grad u-u*(bx,by)) + b0*u = f          in domain
                                          u = gD         on Dirichlet boundary
                    n*(a*grad u -u*(bx,by)) = gN1+gN2*u  on Neumann boundary
                                       u(t0) = u0        initial value

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * M,A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
    Any constant function can be given by its scalar value.
    The functions M,A,B0,BX and BY may also be given as vectors
    with the values of the function at the Gauss points.
  * F may be given as a string for a function depending on (x,y) and time t or
    as a vector with the values at nodes or as scalar.
    If F is given by a scalar or vector it is independent on time.
  * U0 is the initial value, can be given as a constant, function name or
    as vector with the values at the nodes
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n Crank Nicolson steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value.
    * The stepping algorithm can be selected as "SOLVER". The possible values are
      * "CN" the standard Crank-Nicolson (default)
      * "IMPLICIT" the standard implicit solver
      * "EXPLICIT" the standard explicit solver
      * "RK" an L-stable, implicit Runge-Kutta solver
    * Complex coefficients can be selected by TYPE. The possible values are
      * "REAL": all coefficients are real (default)
      * "COMPLEX": some coefficients might be complex
      If only the coefficient M is complex, there is no need to ask for complex coefficients.

return values
  * U is a matrix with n+1 columns with the values of the solution
    at the nodes at different times T
  * T is the vector with the values of the times at which the solutions are returned.
```

If there is no convection term $\vec{b}=\vec{0}$, then the resulting matrix $\mathbf{A}$ is symmetric and (most often) positive definite. Thus one can use a Cholesky factorization for the time stepper. This is (or should be) faster. If at least one of the coefficients a, $\vec{b}$, b_0, f or g_i leads to complex numbers, then it is required to use the option `'type'` with the value `'complex'`. Otherwise obtain wrong results.

The structure of `IBVP2Dsym()` is almost identical to `IBVP2D()`.

IBVP2Dsym()

```
IBVP2Dsym(MESH,M,A,B0,F,GD,GN1,GN2,U0,T0,TEND,STEPS,OPTIONS)
   Solve a symmetric initial boundary value problem
   m*d/dt u - div(a*grad u) + b0*u = f          in domain
                              u = gD            on Dirichlet boundary
                  n*(a*grad u) = gN1+gN2*u      on Neumann boundary
                         u(t0) = u0             initial value
...
```

4.8 Solving semilinear parabolic problems: `IBVP2DNL()`

To solve an semilinear initial boundary value problem (IBVP) of the form (8)

$$\begin{aligned} \rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t,u) && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0 \end{aligned}$$

use the command `IBVP2DNL()`. Do **not** use this command for linear problems, even if it is possible. The linear version `IBVP2D()` is suitable for linear problems and has more time steppers available, some of which are unconditionally stable. The time steppers in `IBVP2DNL()` are conditionally stable.

Find examples in Sections 3.4, 5.10 and 9.33.4.

IBVP2DNL()

```
[U,T]: =  IBVP2D(MESH,M,A,B0,BX,BY,F,GD,GN1,GN2,U0,T0,TEND,STEPS,OPTIONS)

Solve an initial boundary value problem

m*d/dt u - div(a*grad u-u*(bx,by)) + b0*u = f(xy,t,u) in domain
                                       u = gD        on Dirichlet boundary
                    n*(a*grad u -u*(bx,by)) = gN1+gN2*u on Neumann boundary
                                    u(t0) = u0       initial value

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * M,A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
    Any constant function can be given by its scalar value. The functions M,A,B0,BX and BY
    may also be given as vectors with the values of the function at the Gauss points.
  * F may be given as a string or handle for a function depending on (x,y), time t
    and u or as a vector with the values at nodes or as scalar.
    If F is given by a scalar or vector it is independent on time t and u.
  * U0 is the initial value, can be given as a constant, function name or as vector with
    the values at the nodes
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n Crank Nicolson steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value.
    * The stepping algorithm can be selected as "SOLVER". The possible values
      * "CNPC" standard Crank-Nicolson with predictor corrector (default)
      * "CNEXP" standard Crank-Nicolson with explicit nonlinearity
    * Complex coefficients can be selected by TYPE. The possible values are
      * "REAL": all coefficients are real (default)
      * "COMPLEX": some coefficients might be complex. If only the coefficient M is complex,
        there is no need to ask for complex coefficients.

return values
  * U is a matrix with n+1 columns with the values of the solution at the nodes at different times T
  * T is the vector with the values of the times at which the solutions are returned.
```

4.9 Solving hyperbolic problems: `I2BVP2D()`

Examine an IBVP (9) of hyperbolic type.

$$
\begin{aligned}
\rho\,\tfrac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T]\\
u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T]\\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T]\\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0\\
\tfrac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0
\end{aligned}
$$

To solve this wave type equation use the command `I2BVP2D()`. Find examples in Sections 9.2 and 9.18 and a description of the algorithm in Section 6.10.4.

I2BVP2D()

```
[U,T] =  I2BVP2D(MESH,M,D,A,B0,BX,BY,F,GD,GN1,GN2,U0,V0,T0,TEND,STEPS,OPTIONS)

Solve a second order initial boundary value problem

   m*d^2/dt^2 u + 2*d*d/dt u - div(a*grad u-u*(bx,by)) + b0*u = f  in domain
                          u = gD        on Dirichlet boundary
    n*(a*grad u -u*(bx,by)) = gN1+gN2*u on Neumann boundary
                    u(t0) = u0        initial value
                d/dt u(t0) = v0        initial velocity

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * M,D,A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
    Any constant function can be given by its scalar value.
    The functions M,D,A,B0,BX,BY and F may also be given as vectors with the
    values of the function at the Gauss points.
  * F may be given as a string for a function depending on (x,y) and time t or
    as a vector with the values at nodes or as scalar.
    If F is given by a scalar or vector it is independent on time.
  * U0,V0 are the initial value and velocity, can be given as a constant,
    function name or as vector with the values at the nodes
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value.
    * The stepping algorithm can be selected as "SOLVER". The possible values are
      * "IMPLICIT" the standard implicit solver (default)
      * "EXPLICIT" the standard explicit solver
    * Complex coefficients can be selected by TYPE. The possible values are
      * "REAL": all coefficients are real (default)
      * "COMPLEX": some coefficients might be complex
      If only the coefficients M and D are complex, there is no need to ask for complex coefficients.

return values
 * U is a matrix with n+1 columns with the values of the solution
   at the nodes at different times T
 * T is the vector with the values of the times at which the solutions are returned.
```

If at least one of the coefficients a, $\vec{b}$, b_0, f, g_i ot the initial values lead to complex numbers, then it is required to use the option `'type'` with the value `'complex'`. Otherwise obtain wrong results.

4.10 Solving semilinear hyperbolic problems: `I2BVP2DNL()`

Examine a semilinear IBVP (10) of hyperbolic type.

$$\begin{aligned}
\rho\,\tfrac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t,u) && \text{for} \quad (x,y,t)\in\Omega\times(t_0,T]\\
u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(t_0,T]\\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(t_0,T]\\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0\\
\tfrac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=t_0
\end{aligned}$$

To solve this wave type equation use the command `I2BVP2DNL()`. Most of the help for `I2BVP2DNL()` is very similar to `IBVP2D()` and thus only a few differences are shown below.

I2BVP2DNL()

```
[U,T]: = I2BVP2D(MESH,M,D,A,B0,BX,BY,F,GD,GN1,GN2,U0,V0,T0,TEND,STEPS,OPTIONS)

Solve a second order semilinear initial boundary value problem

m*d^2/dt^2 u + 2*d*d/dt u - div(a*grad u - u*(bx,by)) + b0*u = f(x,y,t,u) in domain
                          u = gD        on Dirichlet boundary
      n*(a*grad u -u*(bx,by)) = gN1+gN2*u on Neumann boundary
                     u(t0) = u0        initial value
                 d/dt u(t0) = v0        initial velocity

parameters:
  * MESH is the mesh describing the domain and the boundary types
  ...
  * F may be given as a string or handle for a function depending on (x,y), time t and u
    or as a vector with the values at nodes or as scalar.
    If F is given by a scalar or vector it is independent on time t and u.
  ...
```

4.11 Solving 1D steady state problems, `BVP1D()`

To solve a steady state boundary value problem (11), i.e.

$$-\frac{d}{dx}\left(a(x)\,\frac{d\,u(x)}{dx}\right) + b(x)\,\frac{d\,u(x)}{dx} + c(x)\,u(x) = d(x)\,f(x)$$

with boundary conditions (12) at $x = x_0$ and $x = x_n$

$$\begin{aligned}
u(x_i) &= g_D && \text{Dirichlet}\\
a(x_i)\,\tfrac{d\,u(x_i)}{dx} &= g_{N1} + g_{N2}\,u(x_i) && \text{Neumann}
\end{aligned}$$

use the command `BVP1D()`.

For a call of `BVP1D()`

[x, u] = BVP1D(interval, a, b, c, d, f, BCleft, BCright)

the following parameters are required:

- `interval`: the interval on which the BVP will be solved. It has the form interval $= [x_0, x_1, x_2, \ldots, x_n]$. On each subinterval $[x_i, x_{i+1}]$ the midpoint $\frac{x_i+x_{i+1}}{2}$ will be added and then quadratic functions on the subinterval will be used for the computations.
- `a`, `b`, `c` and `d`: these coefficient functions can be given either as constant scalar value, as vector or as function handle[14] to determine the values.
  - The scalar constant will be used as value at the Gauss points.

[14] This is different from the way to define functions for the 2D codes in FEMoctave.

command	purpose
`BVP1D()`	solve a static boundary value problem in 1D
`IBVP1D()`	solve a first order initial boundary value problem in 1D
`I2BVP1D()`	solve a second order initial boundary value problem in 1D
`BVP1Deig()`	solve an eigenvalue problem in 1D
`BVP1DNL()`	solve a nonlinear boundary value problem in 1D
`IBVP1DNL()`	solve a dynamic nonlinear boundary value problem in 1D
`IBVP1DNL()`	solve a second order dynamic nonlinear boundary value problem in 1D
`GenerateFEM1D()`	generate the matrices for `BVP1D()`
`pwquadinterp()`	piece–wise quadratic interpolation
`FEM1DEvaluateDu()`	evaluate the first derivative at the nodes
`FEM1DIntegrate()`	evaluate integral with Simpson's rule
`FEM1DGaussPoints()`	determine the coordinates of the Gauss points and interpolation matrices
`GenerateWeight1D()`	weight matrices for the dynamic 1D problems

Table 8: Commands to solve and examine 1D boundary value problems and initial boundary value problems

  - The vector has to contain the values at the Gauss points.
  - The function handle will be evaluated at the Gauss points.

  The for these functions code has to be vectorized, e.g. to describe the coefficient $a(x) = x^2$ use the vectorized function handle `a = @(x)x.^2`.

- `f`: this function can be given as constant scalar, as vector or as a vectorized function handle. `f` will determine the values of the function $f(x)$ at the nodes, i.e. the end- and mid-points of the subintervals.
  - The scalar constant will be used as value at all of the nodes.
  - The vector has to contain the values at the nodes.
  - The function handle will be evaluated at the nodes.
- There are two ways to define the inhomogeneous contribution $d(x)\,f(x)$ for a BVP. Except for quadratic functions the results will be slightly different. E.g. for the BVP $-u''(x) = \sin(x)$ the expression $\sin(x)$ can be given by $d(x) = \sin(x)$ and $f(x) = 1$ or $d(x) = 1$ and $f(x) = \sin(x)$.
  1. For $d(x) = \sin(x)$ and $f(x) = 1$ use the arguments `d=@(x)sin(x)` and `f=1`. With this description $\sin(x)$ will be evaluated at the Gauss points and then used for the RHS of the linear system. This is the approach preferred by this author.
  2. For $d(x) = 1$ and $f(x) = \sin(x)$ use the arguments `d=1` and `f=@(x)sin(x)`. With this description $\sin(x)$ will be evaluated at the nodes (end- and mid-points of the subintervals) and with a quadratic interpolation the values at the Gauss points will be determined and then used for the RHS of the linear system. This approach is be useful for iterative procedures for nonlinear problems.
- `BCleft` and `BCright`: with these parameters the boundary conditions are specified. If it is a single scalar Dirichlet conditions are used. If it is a vector of two scalars, Neumann conditions are used. E.g.
  - `BCleft = 3` describes the boundary condition $u(x_0) = 3$
  - `BCleft = [3, -2]` describes the boundary condition $a(x_0)\,u'(x_0) = 3 - 2\,u(x_0)$

There are two return arguments:

- `x` is a vector with the coordinates of the nodes, i.e. all end- and mid–points of the subintervals.
- `u` is a vector with the values of the (approximate) solution $u(x)$ at the nodes.

Internally `BVP1D()` will call the function `GenerateFEM1D()` to approximate the solution of the boundary value problem by a linear system $\mathbf{A}\vec{u} = \mathbf{M}\,\vec{f}$. The code in `BVP1D()` implements the different boundary conditions.

BVP1D()

```
[X,U] = BVP1D(INTERVAL,A,B,C,D,F,BCLEFT,BCRIGHT)

solve a 1D boundary value probme (BVP)
-(a(x)*u'(x))' + b(x)*u'(x) + c(x)*u(x) = d(x)*f(x)
with boundary conditions at the two endpoints
  * Dirichlet: u(x) = g_D
  * Neumann: a(x)*u'(x) = g_N1 + g_N2*u(x)

parameters:
  * INTERVAL the discretized interval for the BVP
  * A constant, vector or function handle to evaluate a(x)
  * B constant, vector or function handle to evaluate b(x)
  * C constant, vector or function handle to evaluate c(x)
  * D constant, vector or function handle to evaluate d(x)
  * F constant, vector or function handle to evaluate f(x)
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D
    * for a Neumann condition specify the values [G_N1,G_N2]

return values
  * X the nodes in the given inteval
  * U the values of the solution at the nodes
```

The function `GenerateFEM1D()` is used by `BVP1D()`, `IBVP1D()`, `I2BVP1D()`, `IBVP1DNL()` and `I2BVP1DNL()` to generate the matrices required to solve the BVP and IBVP.

GenerateFEM1D()

```
[A,M,XNEW] = GenerateFEM1D(X,A,B,C,D)

generate the matrices A and M to discretize the expression
-d/dx (a(x) d/dx u(x)) + b(x)*d/dx u(x) + c(x)*u(x) - d(x)*f(x)
by A*u - M*f

parameters:
  * A function handle to evaluate the coefficient a(x), vectorized
  * B function handle to evaluate the coefficient b(x), vectorized
  * C function handle to evaluate the coefficient c(x), vectorized
  * D function handle to evaluate the coefficient d(x), vectorized
return values
  * A matrix discretizing the expressions involving u(x)
  * M matrix discretizing the evalaution of d(x)*f(x)
  * XNEW vector with the grid points
for an elementary demo use "demo GenerateFEM1D"
```

`BVP1D()` will determine the values of the solution at the nodes. If more values are required use a piece–wise quadratic interpolation, i.e. the command `pwquadinterp()`. The values of the function and the first and second derivatives can be evaluated.

pwquadinterp()

```
yi = pwquadinterp(xdata,ydata,xi)               % evaluate the values of the function
[yi, yi_x, yi_xx] = pwquadinterp(xdata,ydata,xi)% evaluate first and second derivatives

Use the data (xdata,ydata) to determine a piecewise quadratic function
and then evaluate this function at the points xi.
With multiple return arguments derivatives are evaluated.
The function requires that:
  there are an odd number of data points,
  xdata and xi are both in increasing order,
  the xi values lie between xdata(1) and xdata(end).
```

The above command will evaluate the function and derivatives at arbitrary points in the interval. If the values of the derivative are only required at the nodes, use the command `FEM1DEvaluateDu()`.

FEM1DEvaluateDu()

```
DU = FEM1DEvaluateDu(X,U)
  evaluate the first derivative at the nodes x

  parameters:
    * X coordinates of the nodes, generated by BVP1D()
    * U values of the function at the nodes
  return values
    * DU the values of the derivative at the nodes
```

To evaluate the integral of a function given at the nodes use `FEM1DIntegrate()`. This function uses Simpson's rule to evaluate the integral numerically. This is a good match for the second order elements used by `FEMoctave` since piecewise polynomials of degree 2 are integrated exactly.

FEM1DIntegrate.m

```
function res = FEM1DIntegrate(x,u)
  %% Result = FEM1DIntegrate(x,u)
  %% numerical integration of FEM expression, using Simpson's rule
  %% the grid x requires the shape of FEMoctave 1D grids
  dd = diff(x(:));  w = ([dd;0]+[0;dd])/3;  w(2:2:end) *=2;
  res = w'*u(:);
endfunction
```

For some problems the locations of the Gauss points are required. Use the function `FEM1DGaussPoints()`. This function also returns matrices to interpolate the values of a function and its derivative at the Gauss points, using the values at the nodes. See Section 7.2 for the details, equation (93) interpolates the values of the function.

FEM1DGaussPoints()

```
[XGAUSS,NODES2GAUSSU,NODES2GAUSSDU] = FEM1DGaussPoints(X)
determine the coordinates of the Gauss points and interpolation matrices

parameters:
  * X coordinates of the nodes, generated by BVP1D()
return values
  * XGAUSS coordinates of the Gauss points
  * NODES2GAUSSU matrix to evaluate u at the Gauss points
  * NODES2GAUSSDU matrix to evaluate u' at the Gauss points
```

The commands `IBVP1D()` and `I2BVP1D()` use the internal command `GenerateWeight1D()` to solve dynamic 1D problems.

GenerateWeight1D()

```
[XNEW,W1MAT,W2MAT] = GenerateWeight1D(X,W1,W2)

generate the weight matrices W1MAT and W2MAT

parameters:
  * W1 constant, vector or function handle to evaluate the  coefficient w1(x), vectorized
  * W2 (optional) constant, vector or function handle to evaluate the coefficient w2(x),
       vectorized

return values
  * XNEW vector with the grid points
  * W1MAT weight matrix discretizing W1
  * W2MAT (optional) weight matrix discretizing W2
```

4.12 Solving 1D dynamic problems of order 1, `IBVP1D()`

To solve a dynamic boundary value problem (13), i.e.

$$w(x)\,\frac{\partial\,u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t)$$

with boundary conditions as in (12)

$$\begin{aligned} u(x_i) &= g_D && \text{Dirichlet} \\ a(x_i)\,u'(x_i) &= g_{N1} + g_{N2}\,u(x_i) && \text{Neumann} \end{aligned}$$

and an initial condition $u(x, t_0) = u_0(x)$ use the command `IBVP1D()`.

Most of the parameters for a call of

[x, u, t] = IBVP1D(interval, w, a, b, c, d, f, BCleft, BCright, u0, t0, tend, steps, varargin)

are very similar to the above call of `BVP1D()`.

- `w`: is the coefficient function for the weight $w(x)$ and can be given as scalar value, vector of values at the Gauss points or function handle.
- `f`: this function can be given as constant scalar, as vector or as a vectorized function handle of the form `f = @(x,t)`. `f` will determine the values of the function $f(x, t)$ at the nodes, i.e. the end- and mid-points of the subintervals.
- `u0`: the initial value $u(x, t_0) = u_0(x)$, either as single scalar, vector of values at the nodes or a function handle.
- `t0`: `tend`: initial and end time.
- `steps`: number of steps to be taken by the time stepping algorithm.
  - If `steps = n` is a single, positive integer, then n steps will be taken and thus $n + 1$ results returned, including the initial value $u_0(x)$.
  - If `steps = [n nint]` is a vector of two positive integers, then n steps will be taken and thus $n + 1$ results returned. In between the returned results `nint` additional steps will be performed, such that the time step Δt will be smaller.

  The length of one time step is given by $\Delta t = \frac{\text{tend}-\text{t0}}{\text{n}*\text{nint}}$.
- With the optional argument the type of time stepping algorithm can be selected. Use the string `"solver"` as name for the option and a string for the name of the algorithm. Find information on these time steppers in Section 7.6, starting on page 215.
  - `"CN"`: the standard Crank–Nicolson algorithm. This is the default algorithm. It is consistent of order 2 and unconditionally stable, but not L–stable.
  - `"implicit"`: the standard implicit solver. It is consistent of order 1, unconditionally stable and L–stable.
  - `"explicit"`: the standard explicit solver. It is consistent of order 1, only conditionally stable and certainly not L–stable. It is **not recommended** to use this solver on real problems. If instabilities are very likely to show up, `FEMoctave` issues a warning message. It can be used to demonstrate the effect of conditional stability, see page 216.
  - `"RK"`: an implicit Runge–Kutta algorithm. It is consistent of order 2, unconditionally stable and L–stable. The computational effort is larger than for the other algorithms, but still small for 1D problems.

There are three return arguments:

- `x` is a vector with coordinates of the nodes, i.e. all end- and mid–points of the subintervals.
- `u` is a matrix with the values of the (approximate) solutions $u(x)$ at the nodes. The first column contains the initial values. Each column contains the values for one time, given in the return argument `t`.
- `t` is a vector with the times at which the values are returned in `u`.

Internally `IBVP1D()` will call the function `GenerateFEM1D()` to approximate the solution of the initial boundary value problem by a linear system of ordinary differential equations $\mathbf{W}\,\frac{d}{dt}\,\vec{u}(t) + \mathbf{A}\vec{u}(t) = \mathbf{M}\,\vec{f}(t)$. The code in `IBVP1D()` implements the boundary conditions and the time steppers.

IBVP1D()

```
[X,U,T] = IBVP1D(INTERVAL,W,A,B,C,D,F,BCLEFT,BCRIGHT,U0,T0,TEND,STEPS,OPTIONS)

solve a 1D initial boundary value problem (IBVP)

w(x)*d/dt u(x,t) - (a(x)*u'(x,t))' + b(x)*u'(x,t) + c(x)*u(x,t) =  d(x)*f(x,t)

with initial condition u(x,t0) = u0(x) and boundary conditions at the two endpoints
  * Dirichlet:        u(x,t) = g_D
  * Neumann:    a(x)*u'(x,t) = g_N1 + g_N2*u(x)

parameters:
  * INTERVAL the discretized interval for the BVP
  * W constant, vector or function handle to evaluate w(x)
  * A constant, vector or function handle to evaluate a(x)
  * B constant, vector or function handle to evaluate b(x)
  * C constant, vector or function handle to evaluate c(x)
  * D constant, vector or function handle to evaluate d(x)
  * F constant, vector or function handle to evaluate the f(x)
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D
    * for a Neumann condition specify the values [G_N1,G_N2]
  * U0 constant, vector with the initial values at the nodes or a
      function handle to evaluate u(t0)
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value.
    Currently only the stepping algorithm can be selected as "SOLVER"
    and the possible values
      * "CN" the standard Crank-Nicolson (default)
      * "IMPLICIT" the standard implicit solver
      * "EXPLICIT" the standard explicit solver
      * "RK" an L-stable, implicit Runge-Kutta solver

return values
  * X the nodes in the given interval
  * U is a matrix with n+1 columns with the values of the solution at
     the nodes at different times T
  * T is the vector with the values of the times at which the solutions are returned.
```

4.13 Solving 1D dynamic problems of order 2, `I2BVP1D()`

To solve a dynamic boundary value problem (14), i.e.

$$w_2(x)\,\frac{\partial^2 u(x,t)}{\partial t^2} + w_1(x)\,\frac{\partial\, u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\, u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\, u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t)$$

again with Dirichlet or Neumann boundary conditions and an initial value $u(x,t_0) = u_0(x)$ and initial velocity $\frac{\partial}{\partial t}\,u(x,t_0) = u_1(x)$ use the command `I2BVP1D()`. The syntax is similar to the above `IBVP1D()`. The only essential difference is the specification of the initial velocity $u_1(x)$. As time stepper select either the unconditionally stable implicit solver or the conditionally stable explicit solver. Both are consistent of order 2 .

I2BVP1D()

```
[X,U,T] = I2BVP1D(INTERVAL,W1,A,B,C,D,F,BCLEFT,BCRIGHT,U0,U1,T0,TEND,STEPS,OPTIONS)

solve a second order 1D initial boundary value problem (IBVP)

w2(x)*d^2/dt^2 u(x,t) + w1(x)*d/dt u(x,t) - (a(x)*u'(x,t))' +
                  b(x)*u'(x,t) + c(x)*u(x,t) = d(x)*f(x,t)
```

```
with initial condition u(x,t0) = u0(x) and d/dt u(x,t0) = u1 and
boundary conditions at the two endpoints
  * Dirichlet: u(x,t) = g_D
  * Neumann: a(x)*u'(x,t) = g_N1 + g_N2*u(x)

parameters:
  * INTERVAL the discretized interval for the BVP
  * W2 constant, vector or function handle to evaluate w2(x)
  * W1 constant, vector or function handle to evaluate w1(x)
  * A constant, vector or function handle to evaluate a(x)
  * B constant, vector or function handle to evaluate b(x)
  * C constant, vector or function handle to evaluate c(x)
  * D constant, vector or function handle to evaluate d(x)
  * F constant, vector or function handle to evaluate the f(x,t)
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D
    * for a Neumann condition specify the values [G_N1,G_N2]
  * U0 constant, vector with the initial values at the nodes or a
    function handle to evaluate u(t0)
  * U1 constant, vector with the initial velocitiesat the nodes or
    a function handle to evaluate u(t0)
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value. Currently only the
    stepping algorithm can be selected as "SOLVER" and the possible values
    * "IMPLICIT" the standard implicit solver (default)
    * "EXPLICIT" the standard explicit solver

return values
  * X the nodes in the given interval
  * U is a matrix with n+1 columns with the values of the solution
    at the nodes at different times T
  * T is the vector with the values of the times at which the solutions are returned.
```

4.14 Solving 1D eigenvalue problems: `BVP1Deig()`

To solve an 1D eigenvalue problem of the form (15)

$$\begin{aligned} -\frac{d}{dx}\left(a(x)\,\frac{d\,u(x)}{dx}\right) + b(x)\,\frac{d\,u(x)}{dx} + c(x)\,u(x) &= \lambda\, w(x)\,u(x) && \text{for } x_0 < x < x_n \\ u(x_i) &= 0 && \text{Dirichlet BC} \\ a(x_i)\,\frac{d\,u(x_i)}{dx} &= g_{N2}\,u(x_i) && \text{Neumann BC} \end{aligned}$$

use

Octave

```
[x,eVal,eVec,errorbound] = BVP1Deig(interval,a,b,c,w,BCleft,BCright,nVec,options)
```

where the coefficient functions can be given as described in Section 4.11, as constants, vectors or function handles. There are (possibly) two additional arguments

- The integer parameter `nVec` indicates the number of smallest eigenvalues to be be computed.
- The optional parameter `tol` is given as pair string/value. `"TOL",tol` will lead to the iteration stopping if the relative change from one step to the next is smaller than `tol`. If the parameter is not given, then a default value of `tol`$= 10^{-5}$ is used.
- The optional parameter `Mode` is given as pair string/value. It will select what eigenvalues will be determined. The default strings `"Mode","sm"` determine the eigenvalues with smallest magnitude. With `"Mode",sigma` the eigenvalues closest to the scalar `sigma` will be evaluated.

The function can be called with two (`x,Eval`) or three (`[x,Eval,Evev]`) return arguments. A possible fourth return argument (`[x,Eval,Evec,errorbound]`) is of limited use, since with newer versions of FEMoctave `eigs()` is used, instead of an inverse power iteration.

- The first argument `x` contains the nodes at which the solutions are evaluated.
- The second return value `eVal` is a column vector containing the estimated values of the eigenvalues λ_i.
- If the third return value `eVec` is asked for, then a matrix will be returned. Each column contains the values of a normalized eigenfunction at the nodes.
- The fourth return argument `errorbound` is similar to the command `BVP2Deig()` (Section 4.5, page 79).

Examples of 1D eigenvalue problems are given in Section 3.2 and 9.24.2.

BVP1Deig()

```
[X,EVAL,EVEC,ERRORBOUND] = BVP1Deig(INTERVAL,A,B,C,W,BCLEFT,BCRIGHT,NVEC,OPTIONS)

determine the smallest eigenvalues EVAL and eigenfunctions EVEC for the BVP

   -(a(x)*u'(x))' + b(x)*u'(x) + c(x)*u(x) = eVal*w(x)*u(x)
                                      u = 0 on Dirichlet boundary
                                   a*u' = g_N2*u on Neumann boundary

parameters:
  * INTERVAL the discretized interval for the BVP
  * A constant, vector or function handle to evaluate a(x)
  * B constant, vector or function handle to evaluate b(x)
  * C constant, vector or function handle to evaluate c(x)
  * W constant, vector or function handle to evaluate d(x)
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D = 0
    * for a Neumann condition specify the values [G_N1,G_N2] =[0,G_N2]
  * NVEC the number of smallest eigenvalues to be computed
  * TOL tolerance for the eigenvalue iteration, default 1e-5
  * OPTIONS additional options, given as pairs name/value.
    * TOL tolerance for the eigenvalue iteration, default 1e-5
    * "MODE" select the eigenvalues
      * "SM": smallest magnitude, default
      * "SA": smallest algebraic
      * "LM": largest magnitude
      * SIGMA: closest to scalar sigma
      * see "help eigs" for more options

return values
  * X the nodes in the given interval
  * EVAL the eigenvalues of the solution at the nodes
  * EVEC the matric of eigenvectors of the solutions at the nodes
  * ERRORBOUND a matrix with error bounds of the eigenvalues
```

4.15 Solving nonlinear 1D boundary value problems: `BVP1DNL()`

Search for solutions of nonlinear boundary value problems given in equation (17)

$$-\frac{\partial}{\partial x}\left(a(x,u(x),u'(x))\,\frac{\partial\,u(x)}{\partial x}\right)+b(x)\,\frac{\partial\,u(x)}{\partial x}+c(x)\,u(x)=d(x)\,f(x,u(x),u'(x))$$

with linear Dirichlet or Neumann boundary conditions.

- For the dependence on the function f on u and u' Newton's method is used, based on the linear Taylor approximation

$$f(x,u+\phi,u'+\phi')\approx f(x,u,u')+\frac{\partial\,f}{\partial u}\,\phi+\frac{\partial\,f}{\partial u'}\,\phi'$$

- If the coefficient a depends on u or u' a partial substitution method is used.
- If a and f are nonlinear, a combination of Newton and substitution is used. The basic idea is spelled out in Algorithm 1. Find more details in Section 7.8 on page 222.

Algorithm 1: The algorithm of BVP1DNL()

evaluate $a_n = a(x, u_0(x), u_0'(x))$ and $f_0 = f(x, u_0(x), u_0'(x))$
solve $-(a_n\, u_n')' + b\, u_n' + c\, u_n = f_0$
repeat
  set $u_{old} = u_n$
  evaluate $f_n = f(x, u_n(x), u_n'(x))$, $f_u = \frac{\partial}{\partial u} f(x, u_n(x), u_n'(x))$ and $f_{u'} = \frac{\partial}{\partial u'} f(x, u_n(x), u'(x))$
  evaluate the coefficients $b_n = b - d\, f_u$ and $c_n = c - d\, f_{u'}$
  solve $-(a_n\, \phi')' + b_n\, \phi' + c_n\, \phi = +(a_n\, u_n')' - b_n\, u_n' - c_n\, u_n + d\, f_n$
  set $u_n = u_n + \phi$, i.e. one Newton step
  evaluate $a_n = a(x, u_n(x), u_n'(x))$ and $f_n = f(x, u_n(x), u_n'(x))$
  solve $(a_n\, u_n')' + b\, u_n' + c\, u_n = d\, f_n$, i.e one substitution step
until *RMS of* $u_{old} - u_n$ *small enough or too many iterations*;
return the results

BVP1DNL()

```
[X,U] = BVP1DNL(INTERVAL,A,B,C,D,F,BCLEFT,BCRIGHT,U0,OPTIONS)

solve a nonlinear 1D boundary value problem (BVP)

-(a(x,u,u')*u'(x))' + b(x)*u'(x) + c(x)*u(x) = d(x)*f(x,u,u')
with boundary conditions at the two endpoints
  * Dirichlet:          u(x) = g_D
  * Neumann: a(x,u,u')*u'(x) = g_N1 + g_N2*u(x)

parameters:
  * INTERVAL the discretized interval for the BVP
  * A constant, vector or function handle to evaluate a(x), a(x,u) or a(x,u,u')
    at the Gauss points.
    * A a constant or vector of values of a(x).
    * A = @(X) a function handle to evaluate f(x).
    * A = @(X,U) assumes that the function a(x,u) depends on x and u.
    * A = @(X,U,U') assumes that a(x,u,u') depends on x, u and u'.
  * B constant, vector or function handle to evaluate b(x) at Gauss points
  * C constant, vector or function handle to evaluate c(x) at Gauss points
  * D constant, vector or function handle to evaluate d(x) at Gauss points
  * F constant, vector or function handle to evaluate f(x), f(x,u)  or
    f(x,u,u') and the partial derivatives at nodes
    * F a constant or vector of values of f(x) at the nodes.
    * F = @(X) a function handle to evaluate f(x) at the nodes.
    * F = {@(X,U), @(X,U)} assumes that the function f depends on x and u.
      The two function handles evaluate f(x,u) and the partial derivative f_u(x,u).
    * F = {@(X,U,U'), @(X,U,U') , @(X,U,U')} assumes that f depends on x, u and u'.
      The three function handles evaluate f(x,u,u') and the partial derivatives
      f_u(x,u,u') and f_u'(x,u,u').
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D
    * for a Neumann condition specify the two values [G_N1,G_N2]
  * U0 constant, vector or function handle to evaluate u0(x) at the nodes.
    This is the starting value for the iteration.
  * OPTIONS additional options, given as pairs name/value.
    * "TOL" the tolerance for the iteration to stop, given as pair [TOLREL,TOLABS]
      for the relative and absolute tolerance.  The iteration stops if the
      absolute or relative error is smaller than the specified tolerance.
      RMS (root means square) values are used.  If only TOLREL is specified
      TOLABS = TOLREL is used.  The default values are TOLREL = TOLABS = 1E-5.
```

```
    * "MAXITER" the maximal number of iterations to be used. The default value is 10.
    * "DISPLAY" should information be displayed for the iterations
       * "OFF" no display, default
       * "ITER" display the number of the iteration and the RMS size of the update

return values
  * X the nodes in the given interval
  * U the values of the solution at the nodes
  * INFORM a structure with information on the performance of the algorithm
    * INFORM.INFO = 1 if the algorithm converged with the desired tolerance, -1 if not.
    * INFORM.ITER the number of iterations used.
    * INFORM.ABSERROR the RMS value of the last correction applied.
```

4.16 Solving dynamic nonlinear 1D boundary value problems: `IBVP1DNL()`

Search for solutions of nonlinear boundary value problems given in equation (18)

$$w(x)\,\frac{\partial\,u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t,u(x,t))$$

with linear boundary conditions, Dirichlet or Neumann and a given initial condition $u(x,t_0) = u_0(x)$. Three time stepping algorithms are available.

- "CN":
  - A standard Crank–Nicolson scheme is used.
  - For each time step a system of nonlinear equations has to be solved. A Newton method is used, based on

$$f(x,t,u+\phi) \approx f(x,t,u) + \frac{\partial\,f(x,t,u)}{\partial u}\,\phi\,.$$

  - This is the default solver. For each time step a different nonlinear system of equations has to be solved.
  - Some details are shown in Section 7.6.5 on page 217.
- "CNEXP":
  - A standard Crank–Nicolson scheme is used for the linear contributions and a single explicit expression for the nonlinear contribution.
  - This is a very elementary algorithm and only consistent of order 1. The same linear system of equations has to be solved once for each time step.
  - Some details are shown in Section 7.6.6 on page 218.
- "CNPC":
  - A standard Crank–Nicolson scheme is used for the linear contributions and a predictor–corrector step for the for the nonlinear contribution.
  - This solver is recommended if no expression for $\frac{\partial\,f}{\partial u}$ is available. The same linear system of equations has to be solved twice for each time step. The algorithm is (most likely[15]) consistent of order 2.
  - Some details are shown in Section 7.6.6 on page 218.

IBVP1DNL()

```
[X,U,T] = IBVP1DNL(INTERVAL,W,A,B,C,D,F,BCLEFT,BCRIGHT,U0,T0,TEND,STEPS,OPTIONS)

solve a 1D initial boundary value problem (IBVP)

w(x)*d/dt u(x,t) - (a(x)*u'(x,t))' + b(x)*u'(x,t) + c(x)*u(x,t) = d(x)*f(x,t,u(x,t))
with initial condition u(x,t0) = u0(x) and boundary conditions at the two endpoints
  * Dirichlet:     u(x,t) = g_D
  * Neumann: a(x)*u'(x,t) = g_N1 + g_N2*u(x)
```

[15] I have strong hints, but no proofs yet.

```
parameters:
  * INTERVAL the discretized interval for the BVP
  * W constant, vector or function handle to evaluate w(x)
  * A constant, vector or function handle to evaluate a(x)
  * B constant, vector or function handle to evaluate b(x)
  * C constant, vector or function handle to evaluate c(x)
  * D constant, vector or function handle to evaluate d(x)
  * F the nonlinear function. For the algorithm CN this is a structure
      F = {@(X,T,U), @(X,T,U)} with the two function handles to evaluate f(x,t,u) and
      the partial derivative f_u(x,t,u). For the stepping algorithms CNEXP and CNPC this is
      is a function handle to evalute f(x,t,u).
  * F a structure F = {@(X,T,U), @(X,T,U)}.  The two function handles evaluate
    f(x,t,u) and the partial derivative f_u(x,t,u).
  * BCLEFT and BCRIGHT the two boundary conditions
    * for a Dirichlet condition specify a single value G_D
    * for a Neumann condition specify the values [G_N1,G_N2]
  * U0 constant or vector with the initial values at the nodes or
    a function handle to evaluate u(x,t0)
  * T0, TEND are the initial and final times
  * STEPS is a vector with one or two positive integers.
    * If STEPS = n, then n steps are taken and the n+1 results returned.
    * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
  * OPTIONS additional options, given as pairs name/value.
    * "SOLVER" to select the time stepping algorithm.
      * "CN": the standard Crank-Nicolson solver, leading to nonlinear systems to be solved
        for each time step. The nonlinear function and its partial derivative are required.
        This is the default solver.
      * "CNPC": a modified Crank-Nicolson solver with a predictor corrector step,
        using the nonlinear contribution f(x,t,u).
      * "CNEXP": a modified Crank-Nicolson solver, using an explicit expression for the
        nonlinear contribution f(x,t,u).
    * "TOL" the tolerance for the iteration at each time step with the algorithm CN.
      Given as pair [TOLREL,TOLABS] for the relative and absolute tolerance.
      The iteration stops if the absolute or relative error is smaller than the specified
      tolerance. RMS (root means square) values are used. If only TOLREL is specified
      TOLABS = TOLREL is used. The default values are TOLREL = TOLABS = 1E-5.
    * "MAXITER" the maximal number of iterations to be used for each step of the CN
      algorithm. The default value is 10.

return values
  * X the nodes in the given interval
  * U is a matrix with n+1 columns with the values of the solution at the nodes at times T
  * T is the vector with the values of the times at which the solutions are returned.
```

4.17 Solving second order dynamic nonlinear 1D boundary value problems: `I2BVP1DNL()`

Search for solutions of nonlinear boundary value problems given in equation (19)

$$w_2(x)\,\frac{\partial u}{\partial t} + w_1(x)\,\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial u}{\partial x}\right) + b(x)\,\frac{\partial u}{\partial x} + c(x)\,u = d(x)\,f(x,t,u(x,t)) \tag{43}$$

with linear boundary conditions and given initial conditions $u(x,t_0) = u_0(x)$ and $\frac{\partial}{\partial t}\,u(x,t_0) = v_0(x)$. Two time stepping algorithms are available, see Section 7.7. Most of the help for `I2BVP1DNL()` is very similar to `IBVP1D()` and thus only a few differences are shown below.

I2BVP1DNL()

```
[X,U,T]: = I2BVP1DNL(INTERVAL,W2,W1,A,B,C,D,F,BCLEFT,BCRIGHT,U0,U1,T0,TEND,STEPS,OPTIONS)

solve a second order 1D initial boundary value problem (IBVP)

w2(x)*d^2/dt^2u(x,t)+w1(x)*d/dt u(x,t)-(a(x)*u'(x,t))'+b(x)*u'(x,t)+c(x)*u(x,t)=d(x)*f(x,t,u)
```

```
with initial conditions u(x,t0) = u0(x) and d/dt u(x,t0) = u1 and
boundary conditions at the two endpoints
  * Dirichlet:     u(x,t) = g_D
  * Neumann: a(x)*u'(x,t) = g_N1 + g_N2*u(x)

parameters:
  * INTERVAL the discretized interval for the BVP
  * W2 constant, vector or function handle to evaluate w2(x)
  ...
  * F = @(X,T,U) a function handle to evaluate the nonlinear function f(x,t,u)
  ...
  * U0 constant, vector with the initial values at the nodes or a function handle to evaluate u(t0)
  * U1 constant, vector with the initial velocities at the nodes or a function handle
    to evaluate d/dt u(t0)
  ...
  * OPTIONS additional options, given as pairs name/value.
    Currently only the stepping algorithm can be selected as "SOLVER" and the possible values
     * "IMPLICIT" the standard implicit solver (default)
     * "EXPLICIT" the standard explicit solver
  ...
```

4.18 Plane elasticity problems

For a plane stress problem the total energy is given by (29), i.e.

$$
\begin{aligned}
U(\vec{u}) \;=\; & \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dA - \\
& - \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\;dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\;ds\;,
\end{aligned}
$$

respecting the boundary conditions in (20)

$$
\begin{aligned}
\vec{u} &= \vec{g}_D && \text{on Dirichlet boundary } \Gamma_1\text{, i.e. prescribed displacement} \\
\text{force density} &= \vec{g}_N && \text{on Neumann boundary } \Gamma_2\text{, i.e. prescribed force density} \\
\text{force density} &= \vec{0} && \text{on free boundary } \Gamma_3
\end{aligned}
$$

The corresponding Euler–Lagrange equations are shown in (30).

For a plane strain problem find the expression for the total energy in (37).

$$
\begin{aligned}
U(\vec{u}) \;=\; & U_{elast} + U_{Vol} + U_{Surf} \\
=\; & \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2\,(1-2\,\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dA - \\
& - \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\;dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\;ds \\
=\; & \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-(\nu^\star)^2)}\,\left\langle \begin{bmatrix} 1 & \nu^\star & 0 \\ \nu^\star & 1 & 0 \\ 0 & 0 & 2\,(1-\nu^\star) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dA - \\
& - \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\;dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\;ds\;.
\end{aligned}
$$

is minimized, again respecting the boundary conditions (20). Thus the resulting Euler–Lagrange equations are very similar to (30), but with $E^\star$ and $\nu^\star$, given in (36) by

$$\nu^\star = \frac{\nu}{1-\nu} > \nu \quad \text{and} \quad E^\star = \frac{E}{1-\nu^2} > E \,.$$

4.18.1 Solving plane stress and plane strain problems: `PlaneStress()`, `PlaneStrain()`

To solve a plane stress problem use the command `PlaneStress()`.

PlaneStress()

```
[U1,U2] = PlaneStress(MESH,E,NU,F,GD,GN,OPTIONS)
   solve an plane stress problem

     plane stress equation    in domain
                   u = gD     on Gamma_1
      force density = gN      on Gamma_2
      force density = 0       on Gamma_3


   parameters:
     * MESH is the mesh describing the domain and the boundary types
     * E,NU Young's modulus and Poisson's ratio for the material
     * F = {F1,F2} a cell array with the two components of the volume forces
     * GD = {GD1,GD2} a cell array with the two components of the
       prescribed displacements on the boundary section Gamma_1
     * GN = {GN1,GN2} a cell array with the two components of the
       surface forces on the boundary section Gamma_2
     * Any constant function can be given by its scalar value
     * Any function can be given by a string with the function name or a
       function handle
     * The functions E, NU, F1 and F2 may also be given as vectors
       with the values of the function at the Gauss points
     * OPTIONS additional options, given as pairs name/value.
       Currently only one option is possible
       * "THERMAL" and value ALPHADELTAT has to be the function to evaluate the product of
         ALPHA (coefficient of heat expansion) and DELTAT the assumed difference of the
         temperature. ALPHADELTAT can be given as scalar, vector with the values at the
         Gauss points or as function or function handle.
   return values
     * U1 vector with the values of the x-displacement at the nodes
     * U2 vector with the values of the y-displacement at the nodes
```

The code for `PlaneStrain()` is almost identical to `PlaneStress()` .

PlaneStrain()

```
[U1,U2] = PlaneStrain(MESH,E,NU,F,GD,GN,OPTIONS)
     solve an plane strain problem

     plane strain equation    in domain
                   u = gD     on Gamma_1
      force density = gN      on Gamma_2
      force density = 0       on Gamma_3

   parameters:
     * MESH is the mesh describing the domain and the boundary types
     * E,NU Young's modulus and Poisson's ratio for the material
     * F = {F1,F2} a cell array with the two components of the volume forces
     * GD = {GD1,GD2} a cell array with the two components of the
       prescribed displacements on the boundary section Gamma_1
     * GN = {GN1,GN2} a cell array with the two components of the
       surface forces on the boundary section Gamma_2
     * Any constant function can be given by its scalar value
```

```
    * Any function can be given by a string with the function name or a
      function handle
    * The functions E, NU, F1 and F2 may also be given as vectors
      with the values of the function at the Gauss points
    * OPTIONS additional options, given as pairs name/value.
      Currently only one option is possible
      * "THERMAL" and value ALPHADELTAT has to be the function to evaluate the product of
        ALPHA (coefficient of heat expansion) and DELTAT the assumed difference of the
        temperature. ALPHADELTAT can be given as scalar, vector with the values at the
        Gauss points or as function or function handle.
  return values
    * U1 vector with the values of the x-displacement at the nodes
    * U2 vector with the values of the y-displacement at the nodes
```

4.18.2 Eigenvalue problems, `PlaneStressEig()`, `PlaneStrainEig()`

For a domain Ω and material parameter E, ν and the density ρ the eigenvalue problem (32)

$$
\begin{aligned}
-\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{\partial u_1}{\partial x}+\nu\frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2}\left(\frac{\partial u_1}{\partial y}+\frac{\partial u_2}{\partial x}\right)\end{pmatrix}\right) &= \lambda\,\rho\,u_1 \\
-\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{1-\nu}{2}\left(\frac{\partial u_1}{\partial y}+\frac{\partial u_2}{\partial x}\right) \\ \frac{\partial u_2}{\partial y}+\nu\frac{\partial u_1}{\partial x}\end{pmatrix}\right) &= \lambda\,\rho\,u_2
\end{aligned}
$$

is examined by the command `PlaneStressEig()`.

PlaneStressEig()

```
[LA,U1,U2] = PlaneStressEig(MESH,E,NU,W,NVEC,TOL)

solve a plane stress eigenvalue problem

             A*u = la*w*u   in domain, plane stress equation
               u = 0        on Gamma_1
   force density = 0        on Gamma_2
   force density = 0        on Gamma_3

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * E,NU Young's modulus and Poisson's ratio for the material
  * W the material density
    * Any constant function can be given by its scalar value
    * Any function can be given by a string with the function name
    * The functions E, NU and W may also be given as vectors
      with the values of the function at the Gauss points
  * NVEC the number of smallest eigenvalues to be determined
  * TOL optional tolerance for for the eigenvalue iteration, default 1e-5

return values
  * LA the eigenvalues
  * U1 matrix with the values of the x-displacement at the nodes
  * U2 matrix with the values of the y-displacement at the nodes
```

Examples are given in Sections 3.13.4, 5.15, 5.18, 9.50 and 9.51 .

The command `PlaneStrainEig()` is very similar. The only difference is the usage of the plane strain assumption.

4.18.3 Dynamic elasticity problems, `PlaneStressDynamic()`, `PlaneStrainDynamic()`

To solve a dynamic plain stress problem (31)

$$
\begin{aligned}
-\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{\partial u_1}{\partial x}+\nu\frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2}\left(\frac{\partial u_1}{\partial y}+\frac{\partial u_2}{\partial x}\right)\end{pmatrix}\right)+f_1 &= \rho\frac{\partial^2}{\partial t^2}u_1\\
-\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{1-\nu}{2}\left(\frac{\partial u_1}{\partial y}+\frac{\partial u_2}{\partial x}\right) \\ \frac{\partial u_2}{\partial y}+\nu\frac{\partial u_1}{\partial x}\end{pmatrix}\right)+f_2 &= \rho\frac{\partial^2}{\partial t^2}u_2
\end{aligned}
$$

use the command `PlaneStressDynamic()`.

PlaneStressDynamic()

```
[U1,U2,T] = PlaneStressDynamic(MESH,E,NU,RHO,F,GD,GN,U0,V0,T0,TEND,STEPS,OPTIONS)

solve a dynamic plane stress problem

parameters:
  * MESH is the mesh describing the domain and the boundary types
  * E,NU Young's modulus and Poisson's ratio for the material
  * RHO,the density of the material
  * F = {F1,F2} a cell array with the two components of the volume forces.
    The functions take two arguments, coordinates xy and time t,
    i.e. of the form F1(XY,T).
  * GD = {GD1,GD2} a cell array with the two components of the prescribed
    displacements on the boundary section Gamma_1
  * GN = {GN1,GN2} a cell array with the two components of the surface forces on
    the boundary section Gamma_2
  * U0, V0 the initial displacement and initial velocity.
     * Any constant function can be given by its scalar value
     * Any function can be given by a string with the function name
     * The functions E, NU and RHO may also be given as vectors with the values of
       the function at the Gauss points
     * The functions F1, F2, U0 and V0 may also be given as vectors with the values
       of the function at the nodes
   * T0, TEND are the initial and final times
   * STEPS is a vector with one or two positive integers.
     * If STEPS = n, then n steps are taken and the n+1 results returned.
     * If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
   * OPTIONS additional options, given as pairs name/value.
     Currently only the time stepping algorithm can be selected as "SOLVER" and the
     possible values
      * "IMPLICIT" an implicit solver (default)
      * "EXPLICIT" the standard explicit solver

return values
  * U1, U2 matrices with the values of the x- and y-displacements at the nodes.
    Matrices with n+1 columns with the values of the solution at the nodes
    at different times T
  * T vector with the n+1 values of the times at which the solutions are returned.
```

To solve a dynamic plane strain problem use the command `PlaneStrainDynamic()`. The code is a wrapper of the plane stress function, adapting to the modified material parameters

$$
E \;\to\; E^\star = \frac{E}{1-\nu^2} \geq E \quad \text{and} \quad \nu \;\to\; \nu^\star = \frac{\nu}{1-\nu} \geq \nu\,.
$$

You can as well call `PlaneStressDynamic()` with the modified parameters.

Examples are given in Sections 3.13.4 and 9.54.

4.18.4 Evaluating plane stress and plane strain solutions

In Table 9 find the commands related to solving plane elasticity problems and analyzing their solutions.

The functions `EvaluateStress()`, `EvaluateStrain()`, `EvaluateVonMises()`, `EvaluateTresca()` and `EvaluatePrincipalStress()` determine the values at the nodes of the mesh. For many applications using these functions has to be followed by a call of `FEMgriddata()` to evaluate at arbitrary points.

Observe that there are two computational paths:

1. Evaluate the partial derivatives (e.g. $\frac{\partial u_1}{\partial x}$) by a piecewise interpolation of the values of u_1 at the nodes. This is used by the command `FEMgriddata()` with 2 or 3 return arguments.
2. Evaluate the strains (e.g. ε_{xx}) at the nodes, followed by a piecewise interpolation to determine the value at the arbitrary point (x, y). This is used by the command `EvaluateStrain()` to determine the values at the nodes.

They will **NOT** generate identical results. The difference should be small, but can be substantial, in particular for first order elements.

1. The value of `eps_xx_1` is evaluated using the values of u_1 at the nodes and then a piecewise linear or quadratic interpolation leads to the value of the partial derivative $\frac{\partial u_1}{\partial x}$ at the arbitrary point (x, y).
2. The second option `eps_xx_2` will first find values of the strain ε_{xx} at each of the nodes, by taking an average of the partial derivatives $\frac{\partial u_1}{\partial x}$ at the node in the different triangles touching the node. Then a piecewise linear or quadratic interpolation of the values of ε_{xx} at the nodes is used to estimate $\varepsilon_{xx} = \frac{\partial u_1}{\partial x}$ at the arbitrary point (x, y).

```
[~,eps_xx_1,~] = FEMgriddata(FEMmesh,u1,x,y)
[eps_xx,eps_yy,tau_xy] = EvaluateStrain(FEMmesh,u1,u2);
 eps_xx_2 = FEMgriddata(FEMmesh,eps_xx,x,y)
```

4.18.5 Displaying the deformed domain, `ShowDeformation()`

With the command `ShowDeformation()` the original and deformed domain are displayed.

ShowDeformation()

```
ShowDeformation(MESH,U1,U2,FACTOR)

display the original domain and the deformed domain

parameters:
  * MESH is the mesh describing the domain
  * U1 vector with the values of the x-displacements at the nodes
  * U2 vector with the values of the y-displacements at the nodes
  * FACTOR is the scaling factor for the displacements U1 and U2
```

4.18.6 Evaluation of basic strain and stress: `EvaluateStrain()`, `EvaluateStress()`

Given the displacements $\vec{u}_1$ and $\vec{u}_2$ with the corresponding mesh use the function `EvaluateStrain()` to determine the normal and shearing strains at the nodes of the mesh or at a given set of points.

- At the nodes the average values of the strains in the neighboring elements is used, by calling the function `FEMEvaluateGradient()`.
- At arbitrary points `xy` the values at these points are used, by calling `FEMEgriddata()`.

The same function can be used for plane stress and plane strain problems. The missing normal strain ε_{zz} in z–direction can be determined independently.

- For a plane stress setup use $\varepsilon_{zz} = \frac{-\nu}{1-\nu}(\varepsilon_{xx} + \varepsilon_{yy})$.
- For a plane strain setup the assumption is $\varepsilon_{zz} = 0$.

command	purpose
`PlaneStress()`	solve a plane stress problem
`PlaneStrain()`	solve a plane strain problem
`PlaneStressEig()`	solve a plane stress eigenvalue problem
`PlaneStrainEig()`	solve a plane strain eigenvalue problem
`PlaneStressDynamic()`	solve a dynamic plane stress problem
`PlaneStrainDynamic()`	solve a dynamic plane strain problem
`PStressEquationM()`	generate plane stress equations, order 1
`PStressEquationQuadM()`	generate plane stress equations, order 2
`PStressEquationCubicM()`	generate plane stress equations, order 3
`PStressEquationWM()`	generate matrices for plane stress, eigen and dynamic, order 1
`PStressEquationQuadWM()`	generate matrices for plane stress, eigen and dynamic, order 2
`PStressEquationCubicWM()`	generate matrices for plane stress, eigen and dynamic, order 3
`ShowDeformation()`	display the original and deformed domain
`EvaluateStrain()`	given the displacement evaluate the strains at the nodes
`EvaluateStress()`	given the displacement evaluate the stresses at the nodes
`EvaluateVonMises()`	evaluate the von Mises stress at the nodes
`EvaluatePrincipalStress()`	evaluate the three principal stresses at the nodes
`EvaluateTresca()`	evaluate the Tresca stress at the nodes
`EvaluateEnergyDensity()`	evaluate the energy density at the nodes

Table 9: Commands to solve and examine plane elasticity problems

EvaluateStrain()

```
[EPS_XX,EPS_YY,EPS_XY]: = EvaluateStrain(MESH,U1,U2,XY)

  evaluate the normal and shearing strains at the nodes or at arbitary points

  calling the function:
    * with three arguments the values at the nodes are evaluated, i.e.
      [EPS_XX,EPS_YY,EPS_XY] = EvaluateStrain(MESH,U1,U2)
    * with four arguments the values at the points XY are evaluated, i.e.
      [EPS_XX,EPS_YY,EPS_XY] = EvaluateStrain(MESH,U1,U2,XY)

  parameters:
    * MESH is the mesh describing the domain
    * U1 vector with the values of the x-displacements at the nodes
    * U2 vector with the values of the y-displacements at the nodes
    * XY (optional) coodinates of points at which strain is evaluated

  return values:
    * EPS_XX values of normal strain in x direction at the nodes
    * EPS_YY values of normal strain in y direction at the nodes
    * EPS_XY values of shearing strain at the nodes
```

Given the displacements $\vec{u}_1$ and $\vec{u}_2$ with the corresponding mesh use the function `EvaluateStress()` to determine the normal and shearing stresses at the nodes or at a given set of points. Since Hooke's law is used to determine the stresses the material parameters E and ν have to be provided. Use the same function for plane stress and plane strain problems, but with different arguments.

- For a plane stress setup ask for three return arguments σ_x, σ_y and τ_{xy}. All other components of the stress tensor are zero, based on the plane stress assumption.

- For a plane strain setup ask for four return arguments σ_x, σ_y, τ_{xy} and σ_z. Based on Hooke's law the other shearing stresses are given by $\tau_{xz} = \tau_{yz} = 0$.

EvaluateStress()

```
[SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z] =  EvaluateStress(MESH,U1,U2,E,NU,OPTIONS)

  evaluate the normal and shearing stresses at the nodes, using
  Hooke's law for plane stress or plane strain setups

  * [SIGMA_X,SIGMA_Y,TAU_XY] = EvaluateStress(MESH,U1,U2,E,NU)
      with three return arguments assumes a plane stress situation
  * [SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z] =  EvaluateStress(MESH,U1,U2,E,NU)
      with four return arguments assumes a plane strain situation

  parameters:
    * MESH is the mesh describing the domain
    * U1 vector with the values of the x-displacements at the nodes
    * U2 vector with the values of the y-displacements at the nodes
    * E Young's modulus of elasticity, either as constant or as
      string with the function name
    * NU Young's modulus of elasticity, either as constant of as
      string with the function name
    * OPTIONS additional options, given as pairs name/value.  Currently two options are possible
       * "THERMAL" and value ALPHADELTAT has to be the function to evaluate the product of ALPHA
        (coefficient of heat expansion) and DELTAT the assumed difference of the temperature
       * "curve" and the value is a n*2 matrix with the coordinates of the
         points at which the stresses are evaluated

   return values:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * TAU_XY values of shearing strain at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes,
      only for plane strain situations
```

There is again a minor difference depending whether the values are evaluated at the nodes of the arbitrary points `xy`.

- When evaluation at the nodes the material constants are evaluated at the nodes. The strains are not continuous across element. The values at the nodes are the averages over the neighboring triangular elements. The calling sequence is `EvaluateStress()` $\longrightarrow$ `EvaluateStrain()` $\longrightarrow$ `FEMEvaluateGradient()`.
- When using `EvaluateStress()` with an option `curve` the evaluations are (very likely strictly inside the triangular elements. The calling sequence is `EvaluateStress()` $\longrightarrow$ `EvaluateStrain()` $\longrightarrow$ `FEMgriddata()`.

4.18.7 Evaluation of stress expressions: von Mises stress, principal stresses and Tresca stress

There are many stress expressions used for post processing elasticity problems. The commands `EvaluateVonMises()`, `EvaluatePrincipalStress()` and `EvaluateTresca()` allow to evaluate a few of them at the nodes of the given mesh.

The **von Mises stress** σ_M is useful as an indicator for material failure for ductile materials, e.g. most metals. It is one of the most common output expressions used for mechanical FEM simulations. It is a measure for the differences of the principal stresses, since

$$\sigma_M^2 = \frac{1}{2}\left((\sigma_1-\sigma_2)^2+(\sigma_2-\sigma_3)^2+(\sigma_3-\sigma_1)^2\right) .$$

- For a plane stress setup use $\sigma_z = \tau_{xz} = \tau_{yz} = 0$ to simplify the expression for the von Mises stress.

$$\begin{aligned}\sigma_M^2 &= \frac{1}{2}\left((\sigma_x-\sigma_y)^2+(\sigma_y-\sigma_z)^2+(\sigma_z-\sigma_x)^2\right)+3\left(\tau_{xy}^2+\tau_{yz}^2+\tau_{zx}^2\right)\\ &= \frac{1}{2}\left((\sigma_x-\sigma_y)^2+\sigma_y^2+\sigma_x^2\right)+3\,\tau_{xy}^2 \;=\; \sigma_x^2+\sigma_y^2-\sigma_y\sigma_x+3\,\tau_{xy}^2\end{aligned}$$

- For a plane strain setup use $\tau_{xz} = \tau_{yz} = 0$ to simplify the expression for the von Mises stress slightly.

$$\sigma_M^2 = \frac{1}{2}\left((\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2\right) + 3\,\tau_{xy}^2$$

Select the plane stress or plane strain setup by calling the function `EvaluateVonMises()` with three or four input arguments.

- If three arguments σ_x, σ_y and τ_{xy} are given, then a plane stress situation is used.
- If four arguments σ_x, σ_y, τ_{xy} and σ_z are given, then a plane strain situation is used.

EvaluateVonMises()

```
VONMISES = EvaluateVonMises(SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z)

evaluate the von Mises stress at the nodes

  * VONMISES = EvaluateVonMises(SIGMA_X,SIGMA_Y,TAU_XY)
    with three input arguments assumes a plane stress situation
  * VONMISES = EvaluateVonMises(SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z)
    with four input arguments assumes a plane strain situation
  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * TAU_XY values of shearing strain at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes,
              only for plane strain situations
  return values:
      * VONMISES values of the von Mises stress at the nodes
```

By selecting an appropriate (local) coordinate system the shearing stresses will vanish and only the three **principal stresses** σ_1, σ_2 and σ_3 are used. They are the eigenvalues of the stress matrix

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} .$$

- For a plane stress problem determine the principal stresses σ_1 and σ_2 by solving a quadratic equation.

$$\begin{aligned} 0 &= \det \begin{bmatrix} \sigma_x - \sigma & \tau_{xy} \\ \tau_{xy} & \sigma_y - \sigma \end{bmatrix} = \sigma^2 - \sigma\,(\sigma_x + \sigma_y) + \sigma_x\sigma_y - \tau_{xy}^2 \\ \sigma_{1,2} &= \frac{1}{2}\left((\sigma_x + \sigma_y) \pm \sqrt{(\sigma_x + \sigma_y)^2 - 4\,\sigma_x\sigma_y + 4\,\tau_{xy}^2}\right) \\ &= \frac{1}{2}\left((\sigma_x + \sigma_y) \pm \sqrt{(\sigma_x - \sigma_y)^2 + 4\,\tau_{xy}^2}\right) \end{aligned}$$

  For a plane stress problem the third principal stress is given by $\sigma_3 = 0$.

- For a plane strain setup the first two of the above principal stresses remain unchanged. The values of σ_3 are determined by

$$\sigma_3 = \sigma_z = \frac{E\,\nu\,(\varepsilon_{xx} + \varepsilon_{yy})}{(1+\nu)\,(1-2\,\nu)} = \nu\,(\sigma_1 + \sigma_2) = \nu\,(\sigma_x + \sigma_y)\ .$$

  and returned by the function `EvaluateStress()`.

Thus there is no need for code to compute the values of σ_3.

EvaluatePrincipalStress()

```
[SIGMA_1,SIGMA_2] =  EvaluatePrincipalStress(SIGMA_X,SIGMA_Y,TAU_XY)

evaluate the first two principal stresses at the nodes

  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * TAU_XY values of shearing strain at the nodes
  return values:
    * SIGMA_1 first principal stress at the nodes
    * SIGMA_2 second principal stress at the nodes
```

The **Tresca stress** is another indicator for material failure for ductile materials. The Tresca stress measures the differences of the principal stresses and is given by

$$\sigma_T = \max\{|\sigma_1 - \sigma_2|,\, |\sigma_2 - \sigma_3|,\, |\sigma_3 - \sigma_1|\}\ .$$

Select the plane stress or plane strain setup by calling the function `EvaluateTresca()` with three or four input arguments.

- If three input arguments σ_x, σ_y and τ_{xy} are given, then a plane stress situation is used.
- If four input arguments σ_x, σ_y, τ_{xy} and σ_z are given, then a plane strain situation is used.

EvaluateTresca()

```
TRESCA = EvaluateTresca(SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z)

evaluate the Tresca stress at the nodes

  * TRESCA = EvaluateTresca(SIGMA_X,SIGMA_Y,TAU_XY)
           with three input arguments assumes a plane stress situation
  * TRESCA = EvaluateTresca(SIGMA_X,SIGMA_Y,TAU_XY,SIGMA_Z)
           with four input arguments assumes a plane strain situation
  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * TAU_XY values of shearing strain at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes,
      only for plane strain situations

   return values:
     * TRESCA Tresca stress at the nodes
```

4.18.8 Evaluation of the elastic energy density, `EvaluateEnergyDensity()`

To evaluate the elastic energy density W for plane stress or plane strain configurations at the nodes use the command `EvaluateEnergyDensity()`. The result is based on the formulas

$$\begin{aligned}
W_{stress} &= \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy}+2\,(1-\nu)\,\varepsilon_{xy}^2\right)+\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy})\\
W_{strain} &= \frac{E\,(1-\nu)}{2\,(1+\nu)\,(1-2\,\nu)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\frac{\nu}{1-\nu}\,\varepsilon_{xx}\varepsilon_{yy}+2\,\frac{1-2\,\nu}{1-\nu}\,\varepsilon_{xy}^2\right)+\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy})\\
&= \frac{E^\star}{2\,(1-(\nu^\star)^2)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\nu^\star\,\varepsilon_{xx}\varepsilon_{yy}+2\,(1-\nu^\star)\,\varepsilon_{xy}^2\right)+\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy})
\end{aligned}$$

EvaluateEnergyDensity()

```
W = EvaluateEnergyDensity(MESH,EPS_XX,EPS_YY,EPS_XY,E,NU,OPTIONS)

evaluate the elastic energy density at the nodes

parameters:
  * MESH is the mesh describing the domain
  * EPS_XX, EPS_YY, EPS_XY vectors with the values of the strains at the nodes
  * E Young's modulus of elasticity, either as constant or as string with the function name
  * NU Poisson's ratio, either as constant or as string with the function name
  *  OPTIONS additional options, given as pairs name/value.
     * "MODEL" to select the type of plane elasticity problem
        * "PSTRESS" for the plain stress assumption (default)
        * "PSTRAIN" for the plain strain assumption
     * "THERMAL" and value ALPHADELTAT has to be the function to evaluate the product of
       ALPHA (coefficient of heat expansion) and DELTAT the assumed difference of the temperature.
       ALPHADELTAT can be given as scalar, vector with the values at the Gauss points
       or as function or function handle. Default : ALPHA = 0
return value:
  * W values of the elastic energy density at the nodes
```

4.19 Solving axisymmetric elasticity problems, `AxiStress()`

By minimizing the energy given by equation (40) on page 25 an axisymmetric elasticity problem can be solved. The construction of the elements is shown in Section 8.7 starting on page 236. The commands to solve axially symmetric problems and analyze their solutions are shown in Table 10.

4.19.1 Evaluating axisymmetric solutions

To determine the radial displacement u_r and the z–displacement u_z use the command `AxiStress()`.

AxiStress()

```
[UR,UZ] = AxiStress(MESH,E,NU,F,GD,GN)

solve an axisymmetric elasticity problem

plane stress equation    in domain
              u = gD    on Gamma_1
  force density = gN    on Gamma_2
  force density = 0     on Gamma_3

  parameters:
    * MESH is the mesh describing the domain and the boundary types
    * E,NU Young's modulus and Poisson's ratio for the material
    * F = {F1,F2} a cell array with the two components of the volume forces
    * GD = {GD1,GD2} a cell array with the two components of the
      prescribed displacements on the boundary section Gamma_1
    * GN = {GN1,GN2} a cell array with the two components of the
      surface forces on the boundary section Gamma_2
    * Any constant function can be given by its scalar value
    * Any function can be given by a string with the function name
    * The functions E, NU, F1 and F2 may also be given as vectors
      with the values of the function at the Gauss points

  return values
    * UR vector with the values of the r-displacement at the nodes
    * UZ vector with the values of the z-displacement at the nodes
```

command	purpose
`AxiStress()`	solve an axially symmetric elasticity problem
`AxiStressEquationM()`	generate the equations, elements of order 1
`AxiStressEquationQuadM()`	generate the equations, elements of order 2
`AxiStressEquationCubicM()`	generate the equations, elements of order 3
`EvaluateStrainAxi()`	given the displacement evaluate the strains at the nodes
`EvaluateStressAxi()`	given the displacement evaluate the stresses at the nodes
`EvaluateVonMisesAxi()`	evaluate the von Mises stress at the nodes
`EvaluatePrincipalStressAxi()`	evaluate the three principal stresses at the nodes
`EvaluateTrescaAxi()`	evaluate the Tresca stress at the nodes
`EvaluateEnergyDensityAxi()`	evaluate the energy density at the nodes

Table 10: Commands to solve and examine axially symmetric elasticity problems

4.19.2 Evaluation of strains and stress for axisymmetric problems

Based on Section 2.20 the strains for an axisymmetric problem with displacements $u_r(r,z)$ and $u_z(r,z)$ in the plane $y=0$ are given by

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_r}{\partial r} & 0 & \frac{1}{2}(\frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r}) \\ 0 & \frac{1}{r}u_r & 0 \\ \frac{1}{2}(\frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r}) & 0 & \frac{\partial u_z}{\partial z} \end{bmatrix}.$$

Using Hooke's law this leads to the stresses

$$\begin{aligned} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} &= \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \\ &= \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \frac{\partial u_r}{\partial r} \\ \frac{1}{r}u_r \\ \frac{\partial u_z}{\partial z} \end{pmatrix} \\ \tau_{zx} &= \frac{E}{1+\nu}\,\frac{1}{2}(\frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r}) \end{aligned}$$

Based on this the stresses and strains for axisymmetric problems can be evaluated. The codes are similar to the corresponding codes for plane elasticity problems, see Section 4.18.4 starting on page 100.

- **EvaluateStrainAxi()**

```
[EPS_XX,EPS_YY,EPS_ZZ,EPS_XZ] = EvaluateStrainAxi(MESH,UR,UZ)
evaluate the normal and shearing strains at the nodes

  parameters:
    * MESH is the mesh describing the domain
    * UR vector with the values of the r-displacements at the nodes
    * UZ vector with the values of the z-displacements at the nodes
  return values:
    * EPS_XX values of normal strain in x direction at the nodes
    * EPS_YY values of normal strain in y direction at the nodes
    * EPS_ZZ values of normal strain in z direction at the nodes
    * EPS_XZ values of shearing strain at the nodes
```

- **EvaluateStressAxi()**

```
[SIGMA_X,SIGMA_Y,SIGMA_Z,TAU_XZ] = EvaluateStressAxi(MESH,UR,UZ,E,NU)
evaluate the normal and shearing stresses at the nodes, using Hooke's law

  parameters:
    * MESH is the mesh describing the domain
    * UR vector with the values of the r-displacements at the nodes
    * UZ vector with the values of the z-displacements at the nodes
    * E Young's modulus of elasticity, either as constant or as
      string with the function name
    * NU Young's modulus of elasticity, either as constant or as
      string with the function name
  return values:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes
    * TAU_XZ values of shearing strain at the nodes
```

- **EvaluateVonMisesAxi()**

```
VONMISES = EvaluateVonMisesAxi(SIGMA_X,SIGMA_Y,SIGMA_Z,TAU_XZ)
evaluate the von Mises stress at the nodes

  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes
    * TAU_XZ values of shearing strain at the nodes
  return values:
    * VONMISES values of the von Mises stress at the nodes
```

- Based on the stress matrix

$$\begin{bmatrix} \sigma_x & 0 & \tau_{xz} \\ 0 & \sigma_y & 0 \\ \tau_{xz} & 0 & \sigma_z \end{bmatrix}$$

two principal stresses are given by solving a quadratic equation.

$$\begin{aligned} 0 &= \det \begin{bmatrix} \sigma_x - \sigma & \tau_{xz} \\ \tau_{xz} & \sigma_z - \sigma \end{bmatrix} = \sigma^2 - \sigma\,(\sigma_x + \sigma_z) + \sigma_x\sigma_z - \tau_{xz}^2 \\ \sigma_{1,2} &= \frac{1}{2}\left((\sigma_x + \sigma_z) \pm \sqrt{(\sigma_x + \sigma_z)^2 - 4\,\sigma_x\sigma_z + 4\,\tau_{xz}^2}\right) \\ &= \frac{1}{2}\left((\sigma_x + \sigma_z) \pm \sqrt{(\sigma_x - \sigma_z)^2 + 4\,\tau_{xz}^2}\right) \end{aligned}$$

The third principal stress is given by $\sigma_3 = \sigma_y$.

EvaluatePrincipalStressAxi()

```
[SIGMA_1,SIGMA_2] =  EvaluatePrincipalStressAxi(SIGMA_X,SIGMA_Z,TAU_XZ)
evaluate two principal stresses at the nodes

  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes
    * TAU_XZ  values of shearing strain at the nodes
  return values:
    * SIGMA_1 first principal stress at the nodes
    * SIGMA_2 second principal stress at the nodes
```

- **EvaluateTrescaAxi()**

```
TRESCA = EvaluateTrescaAxi(SIGMA_X,SIGMA_Y,SIGMA_Z,TAU_XZ)
evaluate the Tresca stress at the nodes

  parameters:
    * SIGMA_X values of normal stress in x direction at the nodes
    * SIGMA_Y values of normal stress in y direction at the nodes
    * SIGMA_Z values of normal stress in z direction at the nodes
    * TAU_XZ values of shearing strain at the nodes
  return values:
    * TRESCA Tresca stress at the nodes
```

4.19.3 Evaluation of the elastic energy density, `EvaluateEnergyDensityAxi()`

To evaluate the elastic energy density W at the nodes use the command `EvaluateEnergyDensityAxi()`. The result is based on

$$W(r,z) = \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\left((1-\nu)\,(\varepsilon_{rr}^2+\varepsilon_{zz}^2+\frac{1}{r^2}\,u_r^2)+2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz}+\frac{1}{r}\,u_r\,(\varepsilon_{rr}+\varepsilon_{zz}))\right)+\frac{E}{1+\nu}\,\varepsilon_{rz}^2$$

Using the values of the displacements u_r and u_z the command `EvaluateStrainAxi()` will evaluate the required strains.

EvaluateEnergyDensityAxi()

```
W = EvaluateEnergyDensityAxi(MESH,EPS_XX,EPS_YY,EPS_ZZ,EPS_XZ,E,NU)

evaluate the elastic energy density at the nodes for an axially symmetric setup

parameters:
  * MESH is the mesh describing the domain
  * EPS_XX, EPS_YY, EPS_ZZ, EPS_XZ vectors with the values of the strains at the nodes
  * E Young's modulus of elasticity, either as constant or as string with the function name
  * NU Poisson's ratio, either as constant or as string with the function name

return value:
  * W values of the elastic energy density at the nodes
```

4.20 Internal commands in FEMoctave

In this section a few internal commands are documented. Usually these commands are not called directely when solving boundary value problems or elasticity problems. They contain the essential codes to generate the matrices and vectors required to solve the problems. The coding is based on the algorithms shown in Section 6, starting on page 161. They can also be useful to illustrate the essential steps of finite element algorithms, e.g. individual element stiffness matrices.

Within the commands `BVP2D()`, `BVP2Dsym()`, `BVP2Deig()`, `IBVP2D()`, `IBVP2Dsym()`, `IBVP2DNL()`, `I2BVP2D()` and `I2BVP1DNL()` these functions are used.

- If the command `FEMEquation*.oct` is available it is called.
- If the coefficients are complex, the command `FEMEquation*Complex.oct` is searched for and used, if available.
- Otherwise the script file `FEMEquation*M.m` is called. These script files can solve problems with real or complex coefficients. But they are by factors slower that the compiled codes.
- Within `BVP2D()` the command `FEMSolve()` is called to solve the resulting system and apply the Dirichlet boundary conditions.
- Within `BVP2Deig()` the command `eigSmall()` is called to find the few smallest eigenvalues and eigenvectors.

name	purpose
FEMEquationM.m	script to generate equations for linear elements
FEMEquation.cc	to be compiled
FEMEquationComplex.cc	to be compiled, for complex coefficients
FEMEquationQuadM.m	script to generate equations for quadratic elements
FEMEquationQuad.cc	to be compiled
FEMEquationQuadComplex.cc	to be compiled, for complex coefficients
FEMEquationCubicM.m	script to generate equations for cubic elements
FEMEquationCubic.cc	to be compiled
FEMEquationCubicComplex.cc	to be compiled, for complex coefficients
FEMEquationHermite.m	script to generate equations for Hermite elements
fDOF2BF.m	convert Hermite DOF to monomial basis
fBF2uGPmat.m	generate matrices to evaluate in Hermite elements
FEMSolve.m	script to solve the linear equations

Table 11: Internal commands of FEMoctave

4.20.1 Linear elements: FEMEquation.cc, FEMEquationM.m and FEMEquationComplex.cc

This is the fundamental function to transform a BVP to a system of linear equations. First order triangular elements are used. For speed reasons FEMEquation.cc and FEMEquationComplex.cc are written in C++, leading to the compiled codes in FEMEquation.oct and FEMEquationComplex.oct.

FEMEquation()

```
[A,B] = FEMEquation(MESH,A,B0,BX,BY,F,GD,GN1,GN2)

sets up the system of linear equations for a numerical solution of
a PDE using a triangular mesh with elements of order 1

     -div(a*grad u - u*(bx,by))+ b0*u = f          in domain
                                  u = gD           on Dirichlet boundary
            n*(a*grad u - u*(bx,by)) = gN1+g2N*u on Neumann boundary

parameters:
 * MESH triangular mesh of order 1 describing the domain and the boundary type
 * A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
   Any constant function can be given by its scalar value.
   The functions A,B0,BX,BY and F may also be given as vectors
   with the values of the function at the Gauss points.
 * The coefficient A can als be a symmetric matrix A=[AXX,AXY;AXY,AYY]
   given by the row vector [AXX,AYY,AXY].
   It can be given as row vector or as string with the function
   name or as nx3 matrix with the values at the Gauss points.

return values:
 * A, B: matrix and vector for the linear system to be solved, A*u-B=0

See also: FEMEquationQuad, FEMEquationCubic, FEMEquationComplex,
          FEMEquationQuadComplex, FEMEquationCubicComplex, BVP2D, BVP2Dsym,
          BVP2eig, IBVP2D, I2BVP2D, CreateMeshRect, CreateMeshTriangle.
```

The script function FEMEquationM.m performs the same task and is easier to read and understand, but considerably slower than the compiled code. The code in FEMEquationComplex.cc can generate equations with complex coefficients.

4.20.2 Quadratic elements: FEMEquationQuad.cc, FEMEquationQuadM.m and FEMEquationQuadComplex.cc

This is the fundamental function to transform a BVP to a system of linear equations. Second order triangular elements are used. To speed it up it is written in C++.

FEMEquationQuad()

```
[A,B] = FEMEquationQuad(MESH,A,B0,BX,BY,F,GD,GN1,GN2)

sets up the system of linear equations for a numerical solution of
a PDE using a triangular mesh with elements of order 2

     -div(a*grad u - u*(bx,by))+ b0*u = f          in domain
                                    u = gD         on Dirichlet boundary
            n*(a*grad u - u*(bx,by)) = gN1+g2N*u on Neumann boundary

parameters:
 * MESH triangular mesh of order 2 describing the domain and the boundary types
 * A,B0,BX,BY,F,GD,GN1,GN2 are the coefficients and functions describing the PDE.
   Any constant function can be given by its scalar value.
   The functions A,B0,BX,BY and F may also be given as vectors
   with the values of the function at the Gauss points.
 * The coefficient A can als be a symmetric matrix A=[AXX,AXY;AXY,AYY]
   given by the row vector [AXX,AYY,AXY].
   It can be given as row vector or as string with the function
   name or as nx3 matrix with the values at the Gauss points.

return values:
 * A, B: matrix and vector for the linear system to be solved, A*u-B=0

See also: FEMEquation, FEMEquationCubic, FEMEquationComplex,
          FEMEquationQuadComplex, FEMEquationCubicComplex, BVP2D, BVP2Dsym,
          BVP2eig, IBVP2D, I2BVP2D, CreateMeshRect, CreateMeshTriangle.
```

The script function FEMEquationQuadM.m performs the same task and is easier to read and understand, but considerably slower than the compiled code. The code in FEMEquationQuadComplex.cc can generate equations with complex coefficients.

4.20.3 Cubic elements: FEMEquationCubic.cc, FEMEquationCubicM.m and FEMEquationCubicComplex.cc

These three commands are very similar to the above, but use triangular elements of order 3.

4.20.4 Hermite elements: FEMEquationHermite.m

Again very similar to the above, but use triangular Hermite elements.

The command fDOF2BF() takes the DOF (Degrees Of Freedom) of one triangular element and evaluates the coefficients for the basic monomials. Currently Hermite elements can only be used for elliptic problems, i.e. for BVP2D() and BVP2Dsym().

With the command BF2uGPmat() generate matrices to use the DOF in one element to evaluate the values of the function and its partial derivatives of order one. Find details in Section 6.7.

4.20.5 The command FEMSolve.m to solve the linear system

The command FEMSolve() uses the matrix $\mathbf{A}$ and the vector $\vec{b}$ generated by the above FEMEquation* commands to solve the linear system and apply the Dirichlet boundary conditions.

FEMSolve()

```
u = FEMSolve(FEMmesh,A,b,gD)

solves the system of linear equations for a numerical solution of a PDE

nodes contains information about the mesh, see ReadMesh() for the description of the format
```

```
A    is the matrix of the system to be solved.
     It is stored as a full matrix
b    is the RHS of the system to be solved.
'gD' is the function describing the Dirichlet boundary condition

u    is the vector with the values of the solution
```

4.20.6 Effect of right hand side for dynamic problems: `FEMInterpolWeight()`

For the time stepping in parabolic and hyperbolic problems many systems of linear equations have to be solved using the RHS $f(t,x,y)$ for different values of the time t. Thus a function to keep track of the influence of f is useful, `FEMInterpolWeight()`. This function returns a sparse matrix $\mathbf{wMat}$ such that the RHS of the system to be solved is given by $\mathbf{wMat} \cdot \vec{f}$.

FEMInterpolWeight()

```
WMAT = FEMInterpolWeight(FEMMESH,WFUNC)

   create the matrix to determine the contribution of w*f to a IBVP or BVP
   the contribution of w*f is the determined by wMat*f, where f is the
   vector with the values at the "free" nodes

       -div(a*grad u)+ b0*u = w*f       in domain
                        u = gD        on Dirichlet boundary
              n*(a*grad u) = gN1+gN2*u on Neumann boundary

   parameters:
      * MESH is the mesh describing the domain and the boundary types
      * WFUNC is the weight function w
        It may be given as a function name, a vector with the values
        at the Gauss points or as a scalar value

   return value
      * WMAT is the sparse weight matrix
```

This function is used in `IBVP2D()`, `I2BVP2D()` and `IBVP2Dsym()`.

4.20.7 Effect of the Dirichlet values: `FEMInterpolBoundaryWeight()`

If the same system has to be solved for many different Dirichlet values `gD` on the boundary, one can generate the equation once and then only recompute the changes for different boundary values `gD`.

FEMInterpolBoundaryWeight()

```
WMAT = FEMInterpolBoundaryWeight(FEMMESH,A,B0)

   create the matrix to determine the contribution of gD to a IBVP or BVP
   the contribution of gD is the determined by wMat*gD, where gD is
   the vector with the values at the Dirichlet nodes

       -div(a*grad u)+ b0*u = f         in domain
                        u = gD        on Dirichlet boundary
              n*(a*grad u) = gN1+gN2*u on Neumann boundary

   parameters:
      * FEMMESH is the mesh describing the domain and the boundary types.
      * A,B0 are the coefficients and functions describing the PDE.
   return value:
      * WMAT is the sparse weight matrix
```

4.20.8 Determine a few small eigenvalues: `eigSmall()`

In the function `BVP2Deig()` a few small eigenvalues are determined with the help of the wrapper `eigSmall()` for the Octave function `eigs()`. Usually generalized eigenvalues are used in FEMoctave.

eigSmall()

```
[Lambda,{Ev,err}] = eigSmall(A,V,tol)
      solve A*Ev = Ev*diag(Lambda) standard eigenvalue problem

[Lambda,{Ev,err}] = eigSmall(A,B,V,tol)
      solve A*Ev = B*Ev*diag(Lambda) generalized eigenvalue problem

 A    is a (sparse) mxm matrix
 B    is a (sparse) mxm matrix
 V    is either n, the number of desired eigenvalues
      or a mxn matrix, the initial eigenvectors for the iteration
 tol  is the relative error, used as the stopping criterion
 Mode is used to select the eigenvalues, see "help eigs"

 X    is a column vector with the eigenvalues
 EV   is a matrix whose columns represent normalized eigenvectors
 err  is a vector with the aposteriori error estimates for the eigenvalues

 this implementation is based on using eigs()
```

With the optional parameter `Mode` select other eigenvalue, i.e. not the smallest. In case of other wishes modify the code in `eigSmall.m`, it is rather short.

4.20.9 Generating the equations for elasticity problems

The codes `PStressEquationM.m`, `PStressEquationQuadM.m` and `PStressEquationCubicM.m` generate the linear system of equations to be solved for plane stress and plane strain problems. They are used in `PlaneStress()` and `PlaneStrain()`. The *Octave* codes are based on the algorithms in Section 8 (starting on page 224) and easier to read and understand than C++ code, which is not written (yet).

PStressEquationM.m

```
[gMat,gVec] = PStressEquationM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
setup the equation for a plane stress problem with linear elements
```

PStressEquationQuadM.m

```
[gMat,gVec] = PStressEquationQuadM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
setup the equation for a plane stress problem with quadratic elements
```

PStressEquationCubicM.m

```
[gMat,gVec] = PStressEquationCubicM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
setup the equation for a plane stress problem with cubic elements
```

A similar set of functions will generate the matrices for the elastic eigenvalue problems. The codes are in the files `PStressEquationWM.m`, `PStressEquationQuadWM.m` and `PStressEquationcubicWM.m` and they are used in `PlaneStressEig()` and `PlaneStrainEig()`. The same codes are used to generate the matrices for the dynamic elasticity problems, used in `PlaneStressDynamic()` and `PlaneStrainDynamic()`.

For axially symmetric problems `FEMoctave` uses similar commands. Find them in the script files `AxiStressEquationM.m`, `AxiStressEquationQuadM.m` and `AxiStressEquationCubicM.m`.

AxiStressEquationM.m

```
[gMat,gVec] = AxiStressEquationM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%  [gMat,gVec] = AxiStressEquationM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%
%%  setup the equation for an axisymmetric problem with linear elements
```

AxiStressEquationQuadM.m

```
[gMat,gVec] = AxiStressEquationQuadM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%  [gMat,gVec] = AxiStressEquationQuadM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%
%%  setup the equation for an axisymmetric problem with quadratic elements
```

AxiStressEquationCubicM.m

```
[gMat,gVec] = AxiStressEquationCubicM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%  [gMat,gVec] = AxiStressEquationCubicM(Mesh,EFunc,nuFunc,fFunc,gDFunc,gNFunc)
%%
%%  setup the equation for an axisymmetric problem with cubic elements
```

These *Octave* codes might be replaced by compiled codes for speed reasons.

4.21 External programs

- **Triangle**: To construct nonuniform triangular meshes FEMoctave uses an external program. The source code is not distributed with `FEMoctave`, due to copyright worries. Find it on the web at www.cs.cmu.edu/∼quake/triangle.html. Here are instructions on how to install Triangle on a Linux system:
  - On Debian or Ubuntu based system use the package `triangle-bin`.
  - Download the zip file at http://www.netlib.org/voronoi/triangle.zip . Create a directory `triangle` and copy thefile `triangle.zip` into that directory.
  - In that directory apply the following commands in a terminal.

    ```
    unzip triangle.zip
    make
    sudo cp triangle /usr/local/bin/
    ```

    Now `triangle` is available on your system and can be used by `FEMoctave`.
- **CuthillMcKee** to obtain a good numbering. Not necessary any more, since the sparse factorizations provided by `Octave` do a better job.
- **tricountour.m** is a code by Duane Hanselman available at the Mathworks web site `matlabcentral`. It was used by previous versions of the function `FEMtricontour()`. The current version of FEMoctave contains a simple implementation of `tricontour.m` . Neither code is able to generate good labels for the contours.

5 Tools for Didactical Purposes

In this section a few effects of FEM are illustrated. This could be useful to teach a class on the FEM.

5.1 The convergence of the solutions as $h \to 0$ is examined, using an example. Observe the orders of convergence for linear, quadratic and cubic elements.

5.2 Some element stiffness matrices are examined. A path from FEM to a finite difference approximation is shown.

5.3 The behavior of FEM solutions within an element is examined. Find a visualization of the accuracy of linear, quadratics or cubic element.

5.4 The number of nodes, triangles and their effect on the sparsity of the global stiffness matrix is examined.

5.5 Elements of order 1, 2 or 3 are used to solve the same problem. The sizes of the resulting matrices and errors are examined.

5.6 The energy funtional of a BVP is evaluated, using different types of elements.

5.7 A few examples illustrate the second order elements are not C^1–conforming, i.e. the first order derivatives might jump across borders of elements.

5.8 The effect of approximating the domain by a union of triangles is illustrated by an example.

5.9 The effect of superconvergence is illustrated using a 1D boundary value problem.

5.10 The convergence of two modifications of the Crank–Nicolson solvers for semilinear problems are examined.

5.11 The stability of four different time steppers for dynamic heat problems is visualized.

5.12 The stability condition for time steppers for the wave equation is illustrated.

5.13 The effect of shear–locking for elasticity problems, solved with the help of linear elements, is explained and visualized. It is shown why quadratic elements do not suffer from possible shear–locking.

5.14 The standard problem of a bending Euler beam is solved with elements of order 1, 2 and 3 and on different meshes. The results are compared.

5.15 Eigenmodes of a slender bending beam are examined. The effect of different elements, mesh sizes and dimensions of the beam can be observed.

5.16 The effect of stress concentration is illustated with an extreme example.

5.17 The effect of adding missing constraints for a heat and an elasticity problem is examined.

5.18 The (bad) effect of missing boundary constraints for elasticity problems is explained with the help of an example.

5.1 Observe the convergence of the error as $h \to 0$

Consider the unit square $\Omega = [0,1] \times [0,1]$. One can verify that the function $u_e(x,y) = \sin(x) \cdot \sin(y)$ is the exact solution of the boundary value problem

$$\begin{array}{rcll} -\nabla \cdot \nabla u & = & +2\,\sin(x) \cdot \sin(y) & \text{for} \quad 0 \le x, y \le 1 \\ \frac{\partial\, u(x,1)}{\partial y} & = & +\sin(x) \cdot \cos(1) & \text{for} \quad 0 \le x \le 1 \quad \text{and} \quad y = 1 \\ u(x,y) & = & u_e(x,y) & \text{on the other sections of the boundary} \end{array} \quad .$$

Let $h > 0$ be the typical length of a side of a triangle. For second order elements $2\,h$ is used and for third order elements $3\,h$, such that the computational effort is comparable to first order elements. Nonuniform meshes are used, to avoid superconvergence. By choosing different values of h one should observe smaller errors for smaller values of h. The sizes of the matrices vary (approximately) from 50×50 to $58'000 \times 58'000$. The error is measured by computing the L_2 norms of the difference of the exact and approximate solutions, for the values of the functions and its partial derivative with respect to y. These are the expressions used in the theoretical convergence estimates stated in Section 6.8. A double logarithmic plot leads to Figure 51.

- For linear elements:

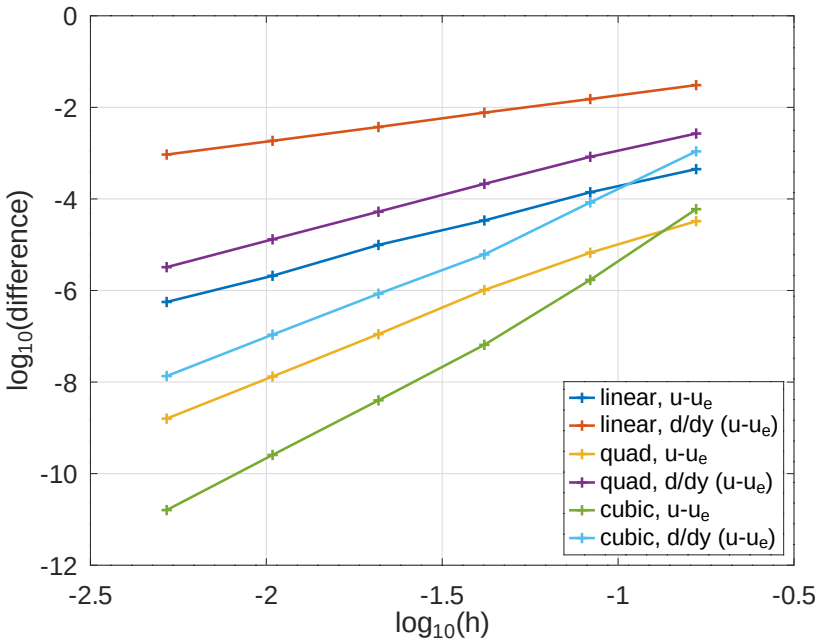


Figure 51: Convergence results for linear, quadratic and cubic elements

  - The slope of the curve for the absolute values of $u(x,y) - u_e(x,y)$ is approximately 2 and thus conclude that the error is proportional to h^2.
  - The slope of the curve for the absolute values of $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 1 and thus conclude that the error of the gradient is proportional to h.

- For quadratic elements:
  - The slope of the curve for the absolute values of $u(x,y) - u_e(x,y)$ is approximately 3 and thus conclude that the error is proportional to h^3.
  - The slope of the curve for the absolute values of $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 2 and thus conclude that the error of the gradient is proportional to h^2.

- For cubic elements:
  - The slope of the curve for the absolute values of $u(x,y) - u_e(x,y)$ is approximately 4 and thus conclude that the error is proportional to h^4.
  - The slope of the curve for the absolute values of $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 3 and thus conclude that the error of the gradient is proportional to h^3.

These observations confirm the theoretical error estimates in Section 6.8 on page 200. It is rather obvious from Figure 51 that higher order elements generate more accurate solutions for a comparable computational effort.

TestConvergence.m

```
a = 1; b0 = 0;  gN2 = 0; N = 6;
Npow = 6;  % use Npow = 6 for final run

function res = u_exact(xy,dymmy)  res =   sin(xy(:,1)).*sin(xy(:,2));   endfunction
function res = f(xy,dummy)        res = 2*sin(xy(:,1)).*sin(xy(:,2));   endfunction
function res = u_y(xy)            res =   sin(xy(:,1)).*cos(xy(:,2));   endfunction

for ii = 1:Npow
  Ni = N*2^(ii-1);   h(ii) = 1/(Ni); area = 0.5/(Ni)^2;
  FEMmesh1 = CreateMeshTriangle('TestConvergence',[0 0 -1;1 0 -1;1 1 -2;0 1 -1],area);
```

```
  FEMmesh2 = CreateMeshTriangle('TestConvergence',[0 0 -1;1 0 -1;1 1 -2;0 1 -1],4*area);
  FEMmesh2 = MeshUpgrade(FEMmesh2,'quadratic');
  FEMmesh3 = CreateMeshTriangle('TestConvergence',[0 0 -1;1 0 -1;1 1 -2;0 1 -1],9*area);
  FEMmesh3 = MeshUpgrade(FEMmesh3,'cubic');

%%% solve with first order elements
  u1 = BVP2Dsym(FEMmesh1,a,b0,'f','u_exact','u_y',gN2);
  Difference(ii) = sqrt(FEMIntegrate(FEMmesh1,(u1-u_exact(FEMmesh1.nodes)).^2));
  [ux,uy] = FEMEvaluateGradient(FEMmesh1,u1);
  DifferenceUy(ii) = sqrt(FEMIntegrate(FEMmesh1,(uy-u_y(FEMmesh1.nodes)).^2));

%%% now for second order elements
  u2 = BVP2Dsym(FEMmesh2,a,b0,'f','u_exact','u_y',gN2);
  DifferenceQ(ii) = sqrt(FEMIntegrate(FEMmesh2,(u2-u_exact(FEMmesh2.nodes)).^2));
  [ux,uy] = FEMEvaluateGradient(FEMmesh2,u2);
  DifferenceUyQ(ii) = sqrt(FEMIntegrate(FEMmesh2,(uy-u_y(FEMmesh2.nodes)).^2));

%%% now for third order elements
  u3 = BVP2Dsym(FEMmesh3,a,b0,'f','u_exact','u_y',gN2);
  DifferenceC(ii) = sqrt(FEMIntegrate(FEMmesh3,(u3-u_exact(FEMmesh3.nodes)).^2));
  [ux,uy] = FEMEvaluateGradient(FEMmesh3,u3);
  DifferenceUyC(ii) = sqrt(FEMIntegrate(FEMmesh3,(uy-u_y(FEMmesh3.nodes)).^2));
endfor
figure(1); plot(log10(h),log10(Difference), '+-',log10(h),log10(DifferenceUy), '+-',
                log10(h),log10(DifferenceQ),'+-',log10(h),log10(DifferenceUyQ),'+-',
                log10(h),log10(DifferenceC),'+-',log10(h),log10(DifferenceUyC),'+-')
                xlabel('log_{10}(h)'); ylabel('log_{10}(difference)')
                legend('linear, u-u_e','linear, d/dy (u-u_e)',
                'quad, u-u_e','quad, d/dy (u-u_e)','cubic, u-u_e','cubic, d/dy (u-u_e)',
                'location','southeast'); xlim([-2.5,-0.5])
```

5.2 Some element stiffness matrices

5.2.1 Element contributions for equilateral triangles

Generate the trivial mesh consisting of a single equilateral triangle with the help of `CreateMeshTriangle()`. The code in `CreateTriangle.m` generates the mesh and Figure 52.

CreateTriangle.m

```
%% corners of an equilateral triangle
corners = 1*[0,0,-2;1,0,-2;0.5,sqrt(3)/2,-2];
mm = CreateMeshTriangle('one_triangle',corners,max(corners(:).^2))
plot([mm.nodes(:,1);mm.nodes(1,1)],[mm.nodes(:,2);mm.nodes(1,2)],'o-r',
      mm.GP(:,1),mm.GP(:,2),'b*')
xlabel('x'); ylabel('y'); title('triangle, with Gauss points'); axis equal
```

For the PDE $-\Delta u = 1$ generate the element stiffness matrix $\mathbf{A}$ and the element vector $\vec{f}$ by using the command `FEMEquation()`.

```
[A,f] = FEMEquation (mm,1,0,0,0,1,0,0,0);
Element_Matrix = full(A)
Element_Vector = f
-->
Element_Matrix =   0.57735  -0.28868  -0.28868
                  -0.28868   0.57735  -0.28868
                  -0.28868  -0.28868   0.57735

Element_Vector =  -0.14434
                  -0.14434
                  -0.14434
```

This result corresponds to the exact result for the element stiffness matrix in Figure 52.

$$\mathbf{A} = \frac{\sqrt{3}}{6}\begin{bmatrix} +2 & -1 & -1 \\ -1 & +2 & -1 \\ -1 & -1 & +2 \end{bmatrix}$$

$$\vec{b} = \frac{\sqrt{3}}{4\cdot 3}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} = \frac{\text{area of triangle}}{3}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

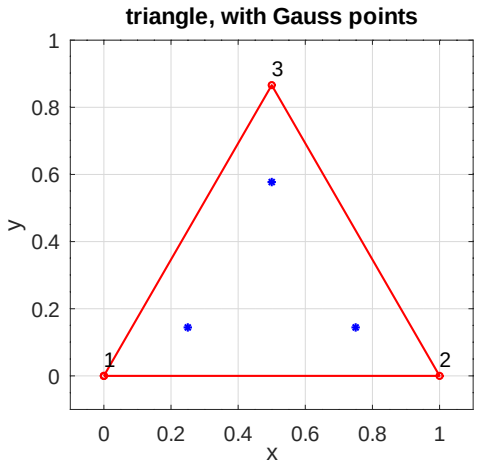


Figure 52: An linear, equilateral triangle, the Gauss integration points and the element stiffness matrix

Using the same idea one can examine the contributions of the different terms to the element stiffness matrix. As example consider the term caused by $b_0\, u = 1\, u$ in the PDE.

```
B = FEMEquation(mm,0,1,0,0,0,0,0,0);
B = full(B)
-->
B =   0.072169   0.036084   0.036084
      0.036084   0.072169   0.036084
      0.036084   0.036084   0.072169
```

The result confirms

$$\mathbf{B} = \frac{\text{area of triangle}}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

Examine a mesh consisting of equilateral triangles, as shown in Figure 53. Then examine the linear equation corresponding to an interior point at (x_i, y_i).

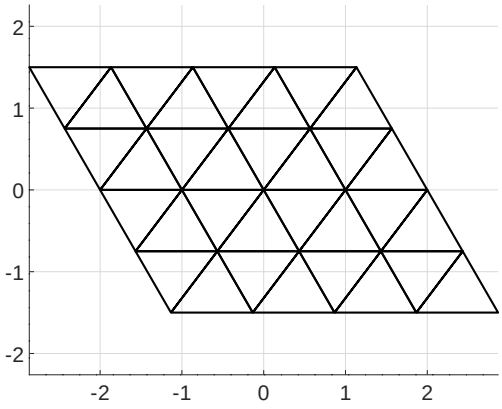


Figure 53: Uniform meshes consisting of equilateral triangles

- The node is corner of 6 triangles, thus the coefficient $a_{i,i}$ of the global stiffness matrix consists of 6 contributions found on the diagonal in the element stiffness matrix $\mathbf{A}$ in Figure 52, i.e. $a_{i,i} = 6\,\frac{+2}{2\sqrt{3}} = \frac{6}{\sqrt{3}}$.

- If a node at (x_j, y_j) shares two triangles with (x_i, y_i) then the entry $a_{i,j}$ in the global stiffness matrix consists of 2 contributions found off the diagonal in the element stiffness matrix $\mathbf{A}$ in Figure 52, i.e. $a_{i,j} = 2\,\frac{-1}{2\sqrt{3}} = \frac{-1}{\sqrt{3}}$.
- If the function f in $-\nabla^2 u = f$ is constant, then there will be 6 contributions from the six neighboring triangle. If the length of one side of a triangle equals h, then the area is $\frac{\sqrt{3}}{4}\,h^2$. Thus find $b_i = 6\,\frac{\text{area of triangle}}{3}\,(-f) = -\frac{\sqrt{3}}{2}\,h^2\,f$.

As a result find the equation for the node at (x_i, y_i).

$$\begin{aligned} \frac{1}{h^2}\left(\frac{6}{\sqrt{3}}\,u(x_i,y_i) - \frac{1}{\sqrt{3}}\sum_{\text{neigbours}} u(x_j,y_j)\right) &= +\frac{\sqrt{3}}{2}\,f \\ \frac{1}{h^2}\left(u(x_i,y_i) - \frac{1}{6}\sum_{\text{neigbours}} u(x_j,y_j)\right) &= +\frac{1}{4}\,f \end{aligned}$$

This is somewhat similar to a finite difference approximation. For each row of the global stiffness matrix the entry on the diagonal and 6 more will be different from 0.

One can examine second order elements and the resulting element stiffness matrix and vector for quadratic elements for the PDE $-\Delta u = 1$. The triangular, equilateral element and the matrix are shown in Figure 54. The vector

$$\mathbf{A} = \frac{\sqrt{3}}{18}\begin{bmatrix} 6 & 1 & 1 & 0 & -4 & -4 \\ 1 & 6 & 1 & -4 & 0 & -4 \\ 1 & 1 & 6 & -4 & -4 & 0 \\ 0 & -4 & -4 & 24 & -8 & -8 \\ -4 & 0 & -4 & -8 & 24 & -8 \\ -4 & -4 & 0 & -8 & -8 & 24 \end{bmatrix}$$

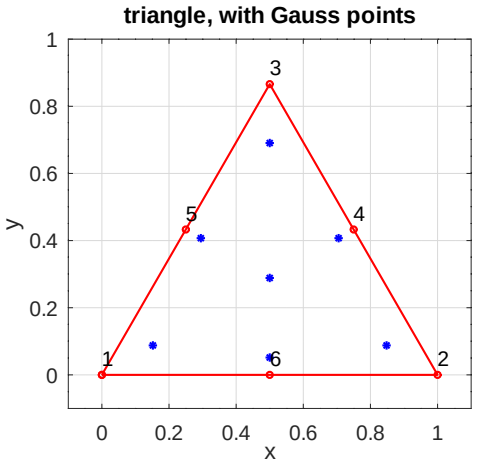


Figure 54: An equilateral, quadratic triangle, the Gauss integration points and the element stiffness matrix

is given by

$$\vec{b} = \frac{\sqrt{3}}{4\cdot 3}\begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix} = \frac{\text{area of triangle}}{3}\begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix}.$$

For the global stiffness matrix for the very regular mesh in Figure 53

- on each row of the matrix corresponding to a corner of the triangle the entry on the diagonal and 12 more will be different from 0. If the mesh is not as regular even 19 entries on each row might be different from zero.
- on each row of the matrix corresponding to a midpoint of a side of the triangle the entry on the diagonal and 6 more will be different from 0. If the mesh is not as regular even 9 entries on each row might be different from zero.

5.2.2 From FEM to a finite difference approximation

Generate the trivial mesh consisting of a single right triangle with the help of `CeateMeshTriangle`. The code shown in `CreateTriangle.m` generates the mesh and Figure 55. For the PDE $-\Delta u = 1$ generate the element stiffness matrix $\mathbf{A}$ and the element vector $\vec{b}$ by using `FEMEquation()` or `FEMEquationM()`.

CreateTriangle.m

```
%% corners of a right triangle
corners = 1*[0,0,-2;1,0,-2;0,1,-2];
CreateMeshTriangle('one_triangle',corners,max(corners(:).^2))
mm = ReadMeshTriangle('one_triangle.1');
[A,f]   = FEMEquation(mm,1,0,0,0,1,0,0,0); %% using compiled code
Element_Matrix = full(A)
Element_Vector = f
-->
Element_Matrix =  1.00000  -0.50000  -0.50000
                 -0.50000   0.50000   0.00000
                 -0.50000   0.00000   0.50000

Element_Vector = -0.16667
                 -0.16667
                 -0.16667
```

$$\mathbf{A} = \begin{bmatrix} +1 & -0.5 & -0.5 \\ -0.5 & +1 & 0 \\ -0.5 & 0 & +1 \end{bmatrix}, \quad \vec{b} = \frac{1}{6}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

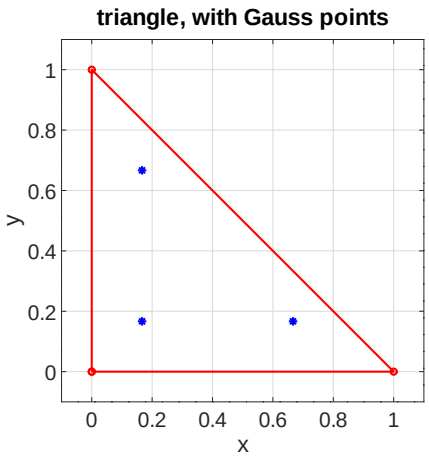
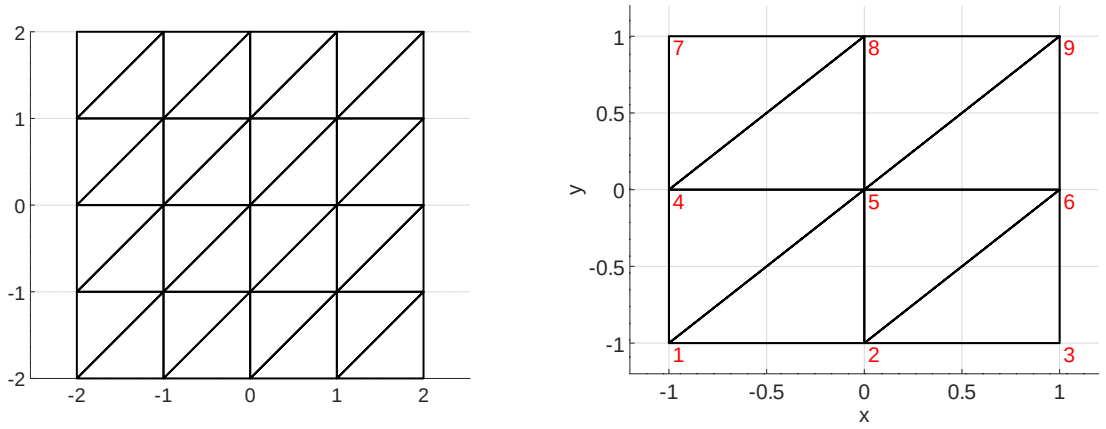


Figure 55: A right triangle, the Gauss integration points and the element stiffness matrix

Based on elements of the above type there is a connection of FEM to the finite difference method. Examine a rectangular grid, shown in Figure 56 and the PDE $-\Delta u = \pi$ with Neumann boundary conditions. Use the command `FEMEquation()` to generate the matrix $\mathbf{A}$ and the vector $\vec{b}$. Then the linear system of equations $\mathbf{A}\,\vec{u} + \vec{b}$ has to be solved. The code displays the equation at node 5.

```
x = [-1,0,1];
FEMmesh = CreateMeshRect(x,x,-2,-2,-2,-2)
figure(1); clf
ShowMesh(FEMmesh.nodes,FEMmesh.elem)
xlabel('x'); ylabel('y')
axis(1.2*[-1,1,-1,1]*max(x))
hold on
for kk = 1:length(FEMmesh.nodes)
   text(FEMmesh.nodes(kk,1)+0.02,FEMmesh.nodes(kk,2)-0.07,num2str(kk),'color',[1 0 0])
endfor
hold off

a=1; b0=bx=by= 0; f=pi;
```



(a) a section

(b) a small section, numbered

Figure 56: Uniform meshes consisting of rectangular triangles

```
[A,b] = FEMEquation(FEMmesh,a,b0,bx,by,f,0,0,0);
A5 = full(A(5,:))
b5 = b(5)
-->
A5 =  0  -1   0  -1   4  -1   0  -1   0
b5 = -3.1416
```

The results imply that the equation to be solved is

$$-u_2 - u_4 + 4\,u_5 - u_6 - u_8 = \pi\,.$$

Running the code again with `x = [1,0,1]/2` will not change $\mathbf{A}$, but lead to $b_5 = -\pi\,4$. Thus for a width h of the triangles the equation to be solved is

$$\frac{-u(x-h,y) - u(x,y-h) + 4\,u(x,y) - u(x+h,y) - u(x,y+h)}{h^2} = f(x,y)\,.$$

This is the usual finite difference approximation of $-\Delta u = f$.

One can examine second order elements and the resulting element stiffness matrix and vector for quadratic elements for the PDE $-\Delta u = 1$. The element and the matrix are shown in Figure 57. The vector is given by

$$\mathbf{A} = \frac{1}{6}\begin{bmatrix} 6 & 1 & 1 & 0 & -4 & -4 \\ 1 & 3 & 0 & 0 & 0 & -4 \\ 1 & 0 & 3 & 0 & -4 & 0 \\ 0 & 0 & 0 & 16 & -8 & -8 \\ -4 & 0 & -4 & -8 & 16 & 0 \\ -4 & -4 & 0 & -8 & 0 & 16 \end{bmatrix}$$

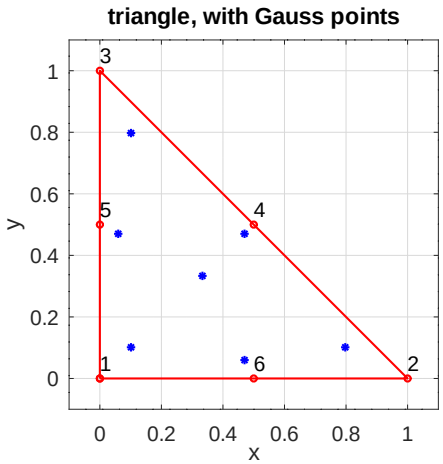


Figure 57: A right angle triangle, the Gauss integration points and the element stiffness matrix

$$\vec{b} = \frac{1}{2 \cdot 3} \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix} = \frac{\text{area of triangle}}{3} \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix} .$$

5.2.3 Element stiffness matrices for 1D problems

To examine an ODE of the form (11)

$$-\left(a(x)\,u'(x)\right)' + b(x)\,u'(x) + c(x)\,u(x) + d(x)\,f(x) = 0$$

with the help of FEM the element stiffness matrix for each subinterval has to be determined. In the case of constant coefficients the formulas from Section 7.2 (page 210) can be simplified, mainly to examine the structure of the matrices. Four contributions have to be taken into account. Use code similar to

```
h = 1;
[A,M,x] = GenerateFEM1D([0 h],1,0,0,1);
A = full(A)
A_rat = rats(A*h)
M = full(M)
M_rat = rats(M/h)
```

to determine the matrices. With the notations $u_- = u(-h/2)$, $u_0 = u(0)$ and $u_+ = u(+h/2)$ obtain

$$I_2 = \int_{-h/2}^{+h/2} u'(x)\,\phi'(x)\,dr \approx \langle \frac{1}{3\,h} \begin{bmatrix} +7 & -8 & +1 \\ -8 & +16 & -8 \\ +1 & -8 & +7 \end{bmatrix} \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}, \begin{pmatrix} \phi_- \\ \phi_0 \\ \phi_+ \end{pmatrix} \rangle = \langle \mathbf{A}_2 \vec{u}, \vec{\phi} \rangle$$

$$I_1 = \int_{-h/2}^{+h/2} u'(x)\,\phi(x)\,dx \approx \langle \frac{1}{6} \begin{bmatrix} -3 & +4 & -1 \\ -4 & 0 & +4 \\ +1 & -4 & +3 \end{bmatrix} \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}, \begin{pmatrix} \phi_- \\ \phi_0 \\ \phi_+ \end{pmatrix} \rangle = \langle \mathbf{A}_1 \vec{u}, \vec{\phi} \rangle$$

$$I_0 = \int_{-h/2}^{+h/2} u(x)\,\phi(x)\,dx \approx \langle \frac{h}{30} \begin{bmatrix} +4 & +2 & -1 \\ +2 & +16 & +2 \\ -1 & +2 & +4 \end{bmatrix} \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}, \begin{pmatrix} \phi_- \\ \phi_0 \\ \phi_+ \end{pmatrix} \rangle = \langle \mathbf{A}_0 \vec{u}, \vec{\phi} \rangle$$

$$I_f = \int_{-h/2}^{+h/2} f(x)\,\phi(x)\,dx \approx \langle \frac{h}{30} \begin{bmatrix} +4 & +2 & -1 \\ +2 & +16 & +2 \\ -1 & +2 & +4 \end{bmatrix} \begin{pmatrix} f_- \\ f_0 \\ f_+ \end{pmatrix}, \begin{pmatrix} \phi_- \\ \phi_0 \\ \phi_+ \end{pmatrix} \rangle = \langle \mathbf{M}_e \vec{f}, \vec{\phi} \rangle$$

The contribution by one element to the linear system $\mathbf{A}\vec{u} = \mathbf{M}\vec{f}$ is

$$(a\,\mathbf{A}_2 + b\,\mathbf{A}_1 + c\,\mathbf{A}_0)\;\vec{u} \quad \text{and} \quad \mathbf{M}_e\,\vec{f}\,.$$

As a very simple example examine the ODE $-u''(x) = f$ on three intervals of length h. Use

$$\mathbf{M}_e \begin{pmatrix} f \\ f \\ f \end{pmatrix} = \frac{f}{6} \begin{pmatrix} 1 \\ 4 \\ 1 \end{pmatrix}$$

and the above element stiffnes matrix $\mathbf{A}_2$ to find the linear system

$$\frac{1}{3\,h}\begin{bmatrix} +7 & -8 & +1 & & & & \\ -8 & +16 & -8 & & & & \\ +1 & -8 & +14 & -8 & +1 & & \\ & & -8 & +16 & -8 & & \\ & & +1 & -8 & +14 & -8 & +1 \\ & & & & -8 & +16 & -8 \\ & & & & +1 & -8 & +7 \end{bmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \frac{f\,h}{6} \begin{pmatrix} 1 \\ 4 \\ 2 \\ 4 \\ 2 \\ 4 \\ 1 \end{pmatrix}$$

- The matrix is symmetric and shows a semi–bandwidth of 3, i.e. at most five entries in each row or column, around the diagonal.
- The above system of 7 equations does not have a unique solution. For a constant vector $\vec{u} = \vec{1}$ the matrix multiplication leads to the zero vector. This is caused by the missing boundary conditions. With the additional constraints $u_0 = u_6 = 0$ the system has a unique solution. In the above matrix the first and last rows and columns have to be removed.
- The second of the above equations is identical to the well known finite difference formula for the second derivative, i.e.

  $$\frac{-u_0 + 2\,u_1 + u_2}{(h/2)^2} = f$$

  and similar for the fourth and sixth equation. The third equation

  $$\frac{u_0 - 8\,u_1 + 14\,u_2 - 8\,u_3 + u_4}{h^2} = f$$

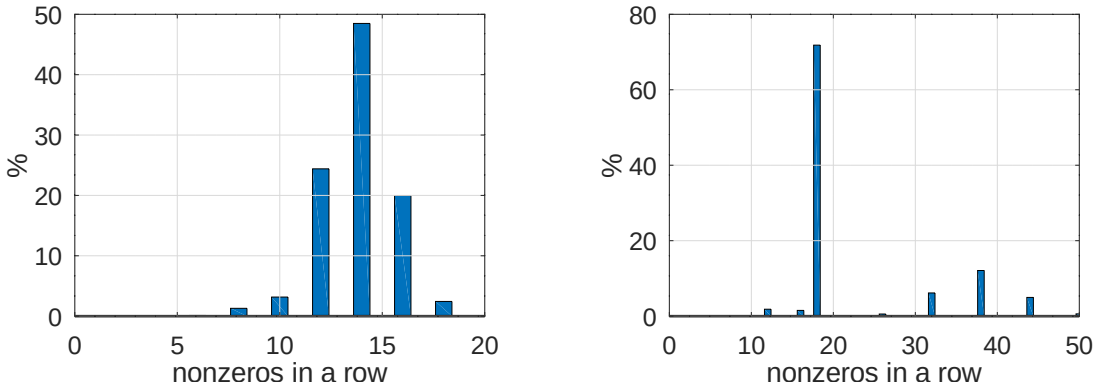
  shows a five point approximation of the second derivative and similar for the fifth equation

5.2.4 Element stiffness matrices for elasticity problems

For the equilateral triangle in Figure 54 examine the symmetric element stiffness matrix for parameters $E = 1$ and $\nu = 0.3$ for linear elements.

$$\mathbf{A}_1 \approx \begin{bmatrix} 0.531 & -0.420 & -0.111 & 0.179 & -0.014 & -0.165 \\ -0.420 & 0.531 & -0.111 & 0.014 & -0.179 & 0.165 \\ -0.111 & -0.111 & 0.222 & -0.192 & 0.192 & 0 \\ 0.179 & 0.014 & -0.192 & 0.325 & -0.008 & -0.317 \\ -0.014 & -0.179 & 0.192 & -0.008 & 0.325 & -0.317 \\ -0.165 & 0.165 & 0 & -0.317 & -0.317 & 0.634 \end{bmatrix}$$

If the location of the corners of the triangle are slightly perturbed, then all entries are different from 0. On a mesh similar to Figure 53 (but not as uniform) with $14'040$ degrees of freedom the number of nonzero entries in each row of the resulting matrix leads to the histogram in Figure 58(a). If each of the nodes would connect to 6 other nodes, then 14 nonzero entries per row are expected. The average observed on the examined mesh is 13.8 nonzeros in each row or column. Thus only $\approx 1‰$ of the entries in the matrix are not zero, i.e. it is a very sparse matrix.



(a) for linear elements (b) for quadratic elements

Figure 58: The number of nonzero entries in each row

For quadratic elements the 12×12 element stiffness matrix is given by

$$\mathbf{A}_2 \approx \begin{bmatrix}
0.53 & 0.14 & 0.04 & 0 & -0.15 & -0.56 & 0.18 & 0 & 0.05 & 0 & -0.22 & -0.02 \\
0.14 & 0.53 & 0.04 & -0.15 & 0 & -0.56 & 0 & -0.18 & -0.05 & 0.22 & 0 & 0.02 \\
0.04 & 0.04 & 0.22 & -0.15 & -0.15 & 0 & 0.06 & -0.06 & 0 & 0.26 & -0.26 & 0 \\
0 & -0.15 & -0.15 & 1.71 & -1.12 & -0.30 & 0 & 0.26 & 0.22 & 0 & 0 & -0.48 \\
-0.15 & 0 & -0.15 & -1.12 & 1.71 & -0.30 & -0.26 & 0 & -0.22 & 0 & 0 & 0.48 \\
-0.56 & -0.56 & 0 & -0.30 & -0.30 & 1.71 & 0.02 & -0.02 & 0 & -0.48 & 0.48 & 0 \\
0.18 & 0 & 0.06 & 0 & -0.26 & 0.02 & 0.33 & 0 & 0.11 & 0 & -0.42 & -0.01 \\
0 & -0.18 & -0.06 & 0.26 & 0 & -0.02 & 0 & 0.33 & 0.11 & -0.42 & 0 & -0.01 \\
0.05 & -0.05 & 0 & 0.22 & -0.22 & 0 & 0.11 & 0.11 & 0.63 & -0.42 & -0.42 & 0 \\
0 & 0.22 & 0.26 & 0 & 0 & -0.48 & 0 & -0.42 & -0.42 & 1.71 & -0.02 & -0.85 \\
-0.22 & 0 & -0.26 & 0 & 0 & 0.48 & -0.42 & -0.00 & -0.42 & -0.02 & 1.71 & -0.85 \\
-0.02 & 0.02 & 0.00 & -0.48 & 0.48 & -0.00 & -0.01 & -0.01 & -0.00 & -0.85 & -0.85 & 1.71
\end{bmatrix}$$

and for a slight perturbation of the corners again all 144 entries are different from zero. On a mesh similar to Figure 53 (but not as uniform) with $56'700$ degrees of freedom the number of nonzero entries in each row of the resulting matrix leads to the histogram in Figure 58(b) with an average of 22.7 nozeros per row or column. For a corner of a triangle contacting 6 triangles expect $6 \cdot 6 + 2 = 38$ nonzero entries. For a midpoint of a triangle expect $2 \cdot 9 = 18$ nonzero entries. The midpoints outnumber the corners by a factor of three. Thus expect an average of $\frac{3 \cdot 18 + 38}{4} = 23$ nonzero entries in each row of the matrix. Thus only $\approx 0.4‰$ of the entries in the matrix are not zero, i.e. it is a very sparse matrix.

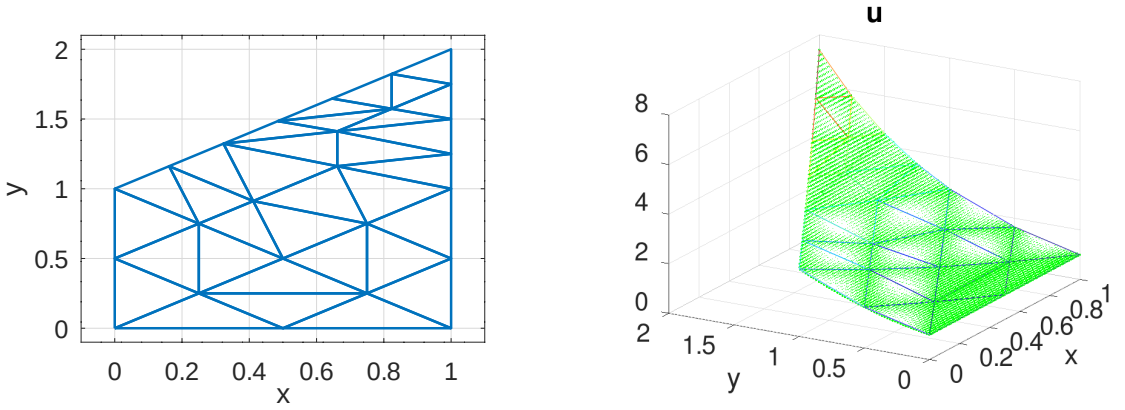
5.3 Behavior of a FEM solution within triangular elements

To examine the behavior of a solution within each of the triangular elements use the boundary value problem

$$\begin{aligned} -\Delta u &= -\exp(y) \quad \text{for} \quad (x,y) \in \Omega \\ u(x,y) &= +\exp(y) \quad \text{for} \quad (x,y) \in \Gamma \end{aligned} .$$

on the domain Ω displayed in Figure 59(a). The exact solution is given by $u(x,y) = \exp(y)$, shown in Figure 59(b). The problem is solved twice:

1. using 32 triangular elements of order 1.
2. using 8 triangular elements of order 2.

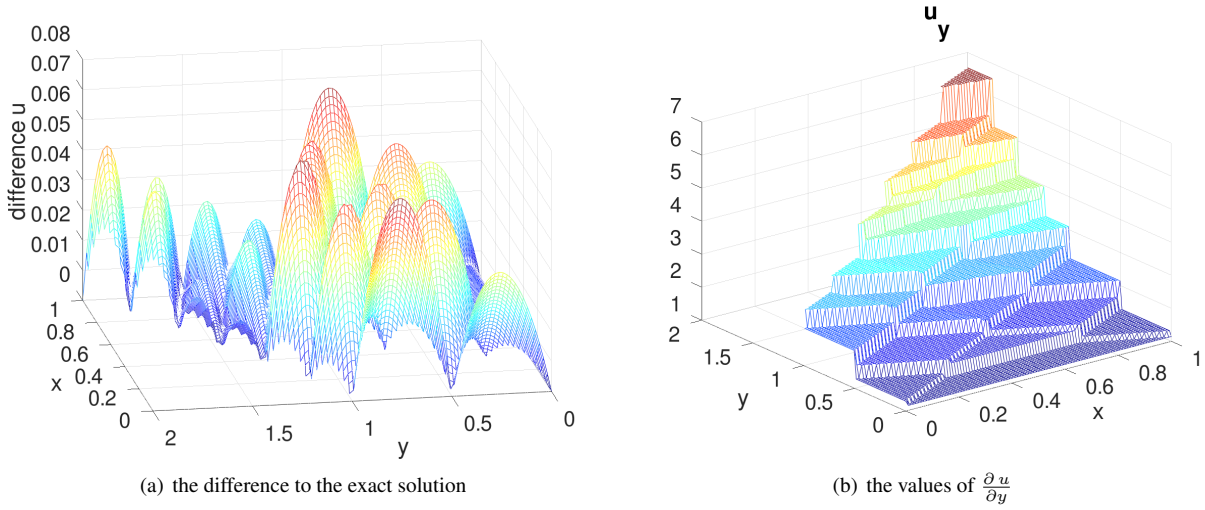


(a) the mesh (b) the solution

Figure 59: The mesh and the solution for a BVP

The nodes used coincide for the two approaches, i.e four triangles in Figure 59(a) for the linear elements correspond to one of the eight triangles for the quadratic elements.

Figure 60(a) shows the difference of the computed solution with first order elements to the exact solution. Within each of the 32 elements the difference is not too far from a quadratic function. Figure 60(b) shows the values of the partial derivative $\frac{\partial u}{\partial y}$. It is clearly visible that the gradient is constant within each triangle, and not continuous across element borders.

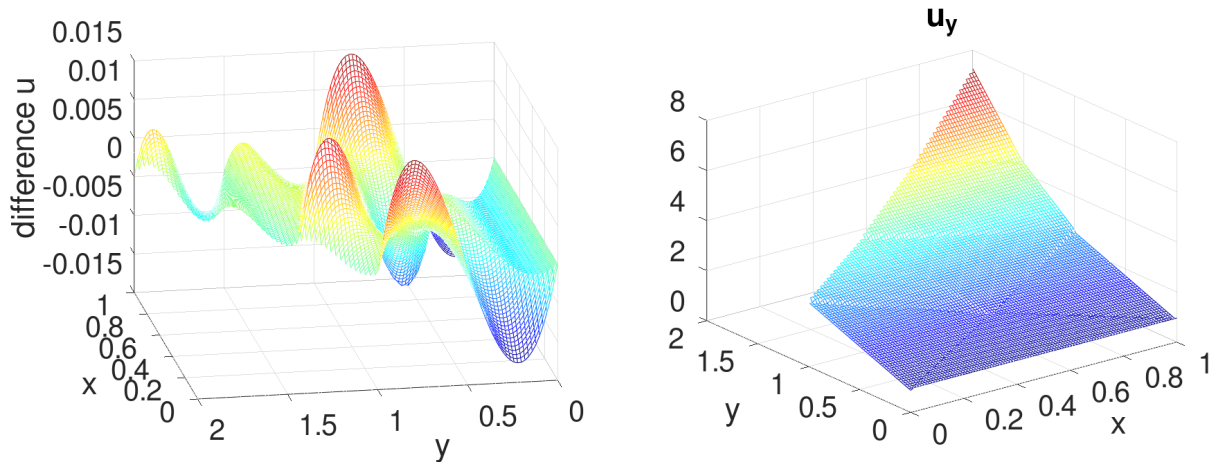


(a) the difference to the exact solution (b) the values of $\frac{\partial u}{\partial y}$

Figure 60: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a first order mesh

Figure 61(a) shows the difference of the computed solution with second order elements to the exact solution. The error is considerably smaller than for linear elements, using identical degrees of freedom. Within each of the 8 elements the difference does not show a simple structure. Figure 61(b) shows the values of the partial derivative $\frac{\partial u}{\partial y}$. It is clearly visible that the gradient is not constant within the triangles. By a careful visual inspection one has to accept that the gradient is not continuous across element borders, but the jumps are considerably smaller than for linear elements. These elements are not C^1–conforming. Figure 62 shows the errors for the partial derivative $\frac{\partial u}{\partial y}$ and confirms this observation.

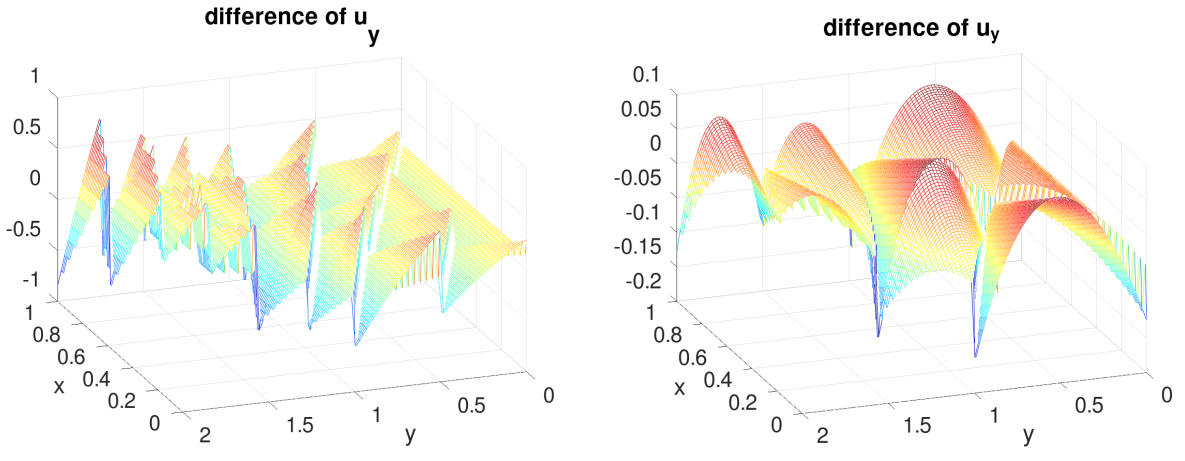
In Figure 63 find the differences of the values of the solution and the partial derivative with respect to y for the same computation using cubic elements. Observe that the approximation errors are considerably smaller. The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ are not continuous across the limits of the triangles, since these third order elements are not C^1–conforming.



(a) the difference to the exact solution

(b) the values of $\frac{\partial u}{\partial y}$

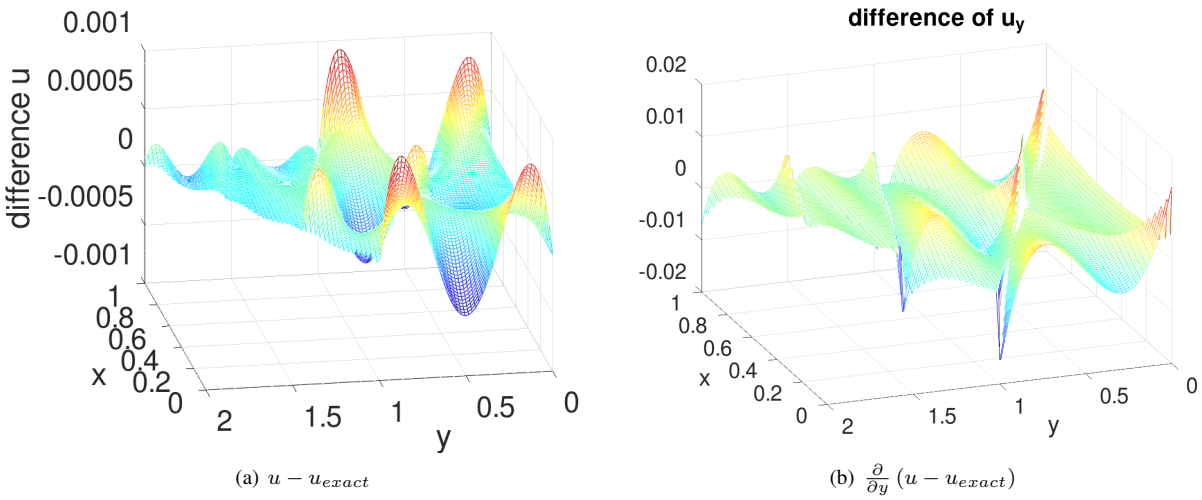
Figure 61: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a second order mesh



(a) using linear elements

(b) using quadratic elements

Figure 62: Difference of the approximate values of $\frac{\partial u}{\partial y}$ to the exact values



(a) $u - u_{exact}$ (b) $\frac{\partial}{\partial y}(u - u_{exact})$

Figure 63: Difference of the approximate values of u and $\frac{\partial u}{\partial y}$ to the exact values for cubic elements

FEMInsideElement

```
N = 2; MeshType = 'quadratic'  %% use 'linear', 'quadratic' or 'cubic'
Mesh = CreateMeshTriangle('test',[0 0 -1;1 0 -1;1 2 -1; 0 1 -1],1/N^2);
switch MeshType
  case 'quadratic'
    Mesh = MeshUpgrade(Mesh,'quadratic');
  case 'cubic'
    Mesh = MeshUpgrade(Mesh,'cubic');
endswitch

xi = linspace(0.2,1.1,5); yi = xi*0.8+0.05;
Ngrid = 100;  [xi,yi] = meshgrid(linspace(0,1,Ngrid),linspace(0,2,Ngrid));

figure(1); FEMtrimesh(Mesh)
           xlabel('x'); ylabel('y'); xlim([-0.1,1.1]); ylim([-0.1,2.1])

function res = u_exact(xy)   res = +exp(xy(:,2)) ; endfunction
function u   = f(xy)           u = -exp(xy(:,2)); endfunction

u_ex = reshape(u_exact([xi(:),yi(:)]),Ngrid,Ngrid);
u = BVP2Dsym(Mesh,1,0,'f','u_exact',0,0);
[ui,uxi,uyi] = FEMgriddata(Mesh,u,xi,yi);

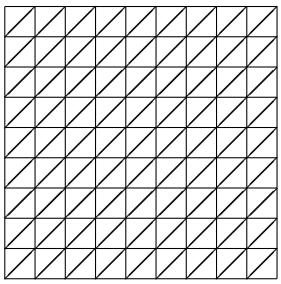
figure(2); FEMtrimesh(Mesh,u);     hold on
           plot3(xi,yi,ui,'g.');   hold off;
           xlabel('x'); ylabel('y'); title('u'); view([-60 25])
figure(3); mesh(xi,yi,uyi)
           xlabel('x'); ylabel('y'); title('u_y')
figure(4); mesh(xi,yi,uyi-u_ex)
           xlabel('x'); ylabel('y'); title('difference of u_y'); view([-110, 30])
```

5.4 Estimate the number of nodes and triangles in a mesh and the effect on the sparse matrix

Let $\Omega \subset \mathbb{R}^2$ be a domain with a triangular mesh with many triangles. There is a connection between

$$N = \text{number of nodes and} \quad T = \text{number of triangles.}$$

Examine the typical mesh on the right and consider only triangles and nodes inside the mesh, as the number of contributions by the borders are considerably smaller for large meshes.



- each triangle has three corners
- each (internal) corner is touched by 6 triangles
- each triangle has 3 midpoints of edges and each of the midpoints is shared by 2 triangles
- For first order elements the nodes are the corners of the triangles.

$$N \approx \frac{1}{6} T\, 3 = \frac{1}{2} T$$

Thus the number N of nodes is approximately half the number T of triangles.

- For second order elements the nodes are the corners of the triangles and the midpoints of the edges. Each midpoint is shared by two triangles.

$$N \approx \frac{1}{2} T + \frac{3}{2} T = 2\, T$$

Thus the number N of nodes is approximately twice the number T of triangles.

- For third order elements the nodes are the corners of the triangles, two points each edge and the central point. Each point on an edge is shared by two triangles.

$$N \approx \frac{1}{2} T + \frac{2 \cdot 3}{2} T + T = \frac{9}{2} T$$

Thus the number N of nodes is approximately 4.5 times the number T of triangles.

- For Hermite elements the number N of nodes and the DOF do **not** coincide. Each corner of a triangle leads to 3 DOF and each triangle adds one of DOF, the value of the solution at the centroid.

$$DOF = 3\, N + T \approx 5\, N \approx \frac{5}{2} T$$

The above implies that the number of degrees of freedom to solve a problem with second or third order elements with a typical diameter h of the triangles is approximately equal to using linear elements on triangles with diameter $h/2$ (quadratic) or $h/3$ (cubic).

The above estimates also allow to estimate how many entries in the sparse matrix resulting from an FEM algorithm will be different from zero.

- For linear elements each node typically touches 6 triangles and each of the involved corners is shared by two triangles. Thus there might be $6 + 1 = 7$ nonzero entries in each row of the matrix.
- For second order triangles distinguish between corners and midpoints.
  - Each corner touches typically six triangles and thus expect up to $6 \times 3 + 1 = 19$ nonzero entries in the corresponding row of the matrix.
  - Each midpoint touches two triangles and two of the corner points are shared. Thus expect up to $2 + 2 \times 3 + 1 = 9$ nonzero entries in the corresponding row of the matrix.

  The midpoints outnumber the corners by a factor of three. Thus expect an average of $\frac{3 \cdot 9 + 19}{4} = 11.5$ nonzero entries in each row of the matrix.
- For third order triangles distinguish between corners, points on edges and center points.
  - Each corner touches typically six triangles and thus expect up to $6 \times 6 + 1 = 37$ nonzero entries in the corresponding row of the matrix.
  - Each point on an edge touches two triangles and four points on the same edge are shared. Thus expect up to 16 nonzero entries in the corresponding row of the matrix.

  - Each center point leads to 10 nonzero entries.

  There are approximately C corners points, $2\,C$ midpoints and on the $3\,C$ edges find $6\,C$ points. Thus expect an average of $\frac{1\cdot 37+6\cdot 16+2\cdot 10}{2+6+1} = \frac{153}{9} = 17$ nonzero entries in each row of the matrix.

- The above estimates are not correct for equations with constant coefficients or horizontal or vertical edges. Then expect fewer nonzero entries in each row of the matrix.

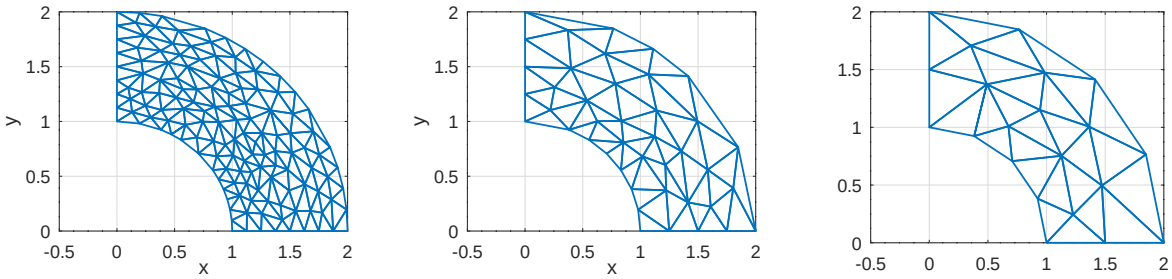
This points to about a factor of $\frac{11.5}{7} \approx 1.6$ more nonzero entries in the matrix for quadratic elements for the same number of degrees of freedom. For cubic elements expect a factor of $\frac{17}{7} \approx 2.4$. This implies that the computational effort is larger, the actual effect depends on the linear solver used.

5.5 Compare linear, quadratic and cubic elements

To examine the performance of the different order elements examine the BVP

$$\begin{aligned} -\nabla\left((1+x^2)\nabla u(x,y)\right) &= -4\,(1+x^2)\,\exp(-2\,y) && \text{for} \quad (x,y) \in \Omega \\ \tfrac{\partial\, u(y,0)}{\partial x} &= 0 && \text{for} \quad 1 \le y \le 2 \\ u(x,y) &= \exp(-2\,y) && \text{on other sections of the boundary} \end{aligned} \quad .$$

on the domain shown in Figure 64. The exact solution is given by $u_e(x,y) = \exp(-2\,y)$. For different values of the



(a) linear elements (b) quadratic elements (c) cubic elements

Figure 64: Meshes for linear, quadratic and cubic elements, leading to similar size linear systems to be solved.

typical element size h for linear elements the three types of elements are used.

- For quadratic elements use $h_{quad} = 2\,h$ to aim for the same number of degrees of freedom, i.e. the same size of linear system of equations to be examined. For cubic elements $h_{cubic} = 3\,h$ is used. This leads to meshes shown in Figure 64. Observe that the mesh for cubic elements is not as good as the mesh for linear elements to approximate the deformed domain, caused by the larger elements.
- For each solution u determine the L_2 error, i.e.

$$\text{error} = \left(\iint\limits_{\Omega} |u(x,y) - u_e(x,y)|^2 \, dA \right)^{1/2} \quad .$$

- For each setup determine the size $n \times n$ of the matrix $\mathbf{A}$ for the linear system to be solved.
- For each setup determine the number of nonzero entries in the sparse matrix $\mathbf{A}$ and then the average number of nonzeros in each row of $\mathbf{A}$.
- When different values h_1 and h_2 are used the expression the errors are expected to be proportional to h^k, with the order of convergence k. Thus if h is replaced by $h/2$ expect ratios of 2, 4, 8 or 16 for the L_2 errors, according to the theoretical results shown in Section 6.8 on page 200.
- If h is replaced by $h/2$ expect the number of elements and the size of the matrix $\mathbf{A}$to be multiplied by 4. The number of nonzero entries in each row should not change drastically.

The results in Table 12 confirm the theoretical estimates of the errors and the number of nonzero entries in the matrix $\mathbf{A}$.

Element	linear		quadratic		cubic	
width h of elements	0.025	0.0125	0.050	0.0250	0.075	0.0375
number of elements	3944	15912	998	3944	432	1764
size n of matrix	1920	7850	1920	7850	1896	7842
L_2 error	$2.2 \cdot 10^{-4}$	$6.4 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$	$1.4 \cdot 10^{-6}$	$8.4 \cdot 10^{-7}$	$5.6 \cdot 10^{-8}$
ratio of L_2 errors		≈ 2.9		≈ 4.7		≈ 15
nonzeros per row	6.8	6.9	11.0	11.2	16.1	16.6

Table 12: Results for elements of order 1, 2 and 3

5.6 Minimizing the energy, depending on type of element

The elements used in `FEMoctave` are used to illustrate the minimizing aspect of the FEM solutions.

A solution of the boundary value problem

$$\begin{array}{rcll} \Delta u + b\,u & = & f & \text{in the domain } 0 < x, y < 1 \\ u(x,y) & = & -2 + x + 2\,y & \text{on the boundary } (x,0) \text{ and } (0,y) \\ \frac{\partial}{\partial n}\, u(x,y) & = & 0 & \text{on the boundary } (x,1) \text{ and } (1,y) \end{array} \tag{44}$$

is a minimizer of the functional

$$F(u) = \int\limits_{0<x,y<1} \frac{1}{2}\,(\nabla u)^2 + \frac{b}{2}\,u^2 - f \cdot u \; dA \; . \tag{45}$$

The solution for $b = 3$ and $f = 10$ is shown in Figure 65.

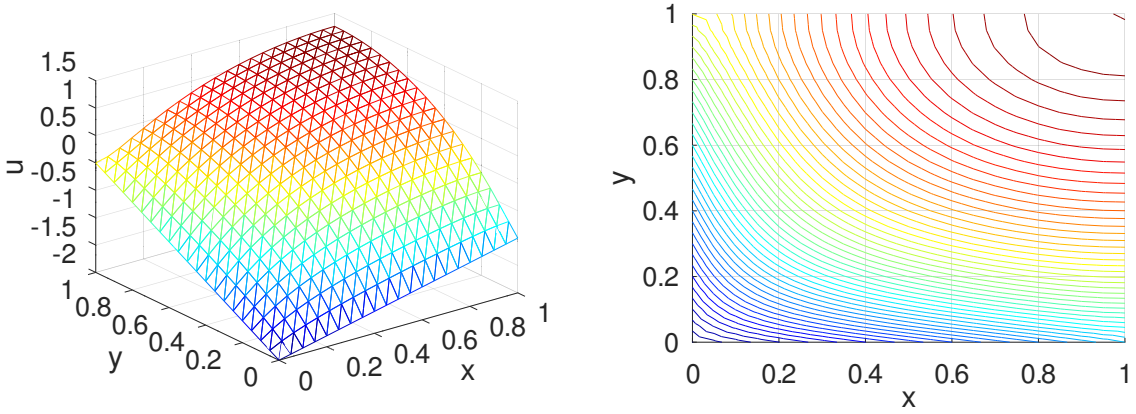


Figure 65: Graph and contour lines for the solution of the BVP (44)

The code `MinimizeEnergy.m` shown below solves the BVP (44) with $b = 3$ and $f = 10$ on different meshes with different types of elements. For each type of element and mesh the number of elements, the number of DOF, the RMS difference to the exact[16] solution and the resulting energy (45) is evaluated. The results are shown in Table 13.

Li An $(N+1) \times (N+1) = 11 \times 11$ grid of points leads to 200 triangles with linear elements. This system has 100 DOF.

Qu The above mesh is updated to 200 triangles with quadratic elements, leading to 400 DOF. As expected the error is smaller and the minimal energy is smaller, since a larger class of functions is used.

Li4 The above mesh with quadratic elements is converted to a linear mesh with the same DOF. The error and the minimal energy are increased. This illustrates that in most cases quadratic elements are more efficient than linear elements.

[16] Generated with the help of cubic elements on a finer mesh. Thus the "exact" shall not be taken to seriously.

Cu The original mesh with linear elements is updated 200 triangles with cubic elements, leading to 900 DOF. As expected the error is smaller and the minimal energy is even smaller, since a larger class of functions is used, i.e. a piecewise cubic interpolation.

Li9 The above mesh with cubic elements is converted to a linear mesh with the same DOF. The error and the minimal energy are increased. This illustrates that in most cases cubic elements are more efficient than linear elements.

He The original mesh with linear elements is updated 200 triangles with Hermite elements, leading to 542 DOF. The error is rather large, illustrating a problem with Hermite elements, caused by the dubious implementation of the Dirichlet boundary conditions, see Section 6.7.7. The estimated energy is small, in particular smaller that the minimal energy generated by cubic elements. This is caused by the fewer boundary constraint imposed by the Hermite elements, i.e. at the midpoints of the boundary segments the Dirichlet condition is not enforced.

	type of element	elements	DOF	error	energy
Li	Linear	200	100	$5.96 \cdot 10^{-3}$	5.95212
Qu	Quadratic	200	400	$2.53 \cdot 10^{-4}$	5.91112
Li4	Linear, from quadratic	800	400	$1.52 \cdot 10^{-3}$	5.92148
Cu	Cubic	200	900	$3.00 \cdot 10^{-5}$	5.91035
Li9	Linear, from cubic	1800	900	$6.74 \cdot 10^{-4}$	5.91538
He	Hermite	200	542	$1.01 \cdot 10^{-2}$	5.86885

Table 13: The minimal energy for different types of elements

MinimizeEnergy.m

```
Ngrid = 11;  N = (Ngrid-1)^2;
Mesh = CreateMeshRect(linspace(0,1,Ngrid),linspace(0,1,Ngrid),-1,-2,-1,-2);
figure(1); FEMtrimesh(Mesh); xlabel('x'); ylabel('y')

a = 1; f = 10; b = 3;
function res = gD(xy)
  res = -2 + 1*xy(:,1) + 2*xy(:,2);
endfunction

%  case 'Li'
u_Li = BVP2Dsym(Mesh,a,b,f,'gD',0,0);
figure(2); FEMtrimesh(Mesh,u_Li); xlabel('x'); ylabel('y'); zlabel('u')
figure(3); FEMtricontour(Mesh,u_Li,40); xlabel('x'); ylabel('y');
[uGP,graduGP] = FEMEvaluateGP(Mesh,u_Li);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_Li = FEMIntegrate(Mesh,EnergyDensity);
disp(sprintf('Linear elements %i, DOF = %i, energy = %7.5f',
             2*N,Mesh.nDOF,Energy_Li))
%  case 'Qu'
MeshQu = MeshUpgrade(Mesh,'quadratic');
u_Qu = BVP2Dsym(MeshQu,a,b,f,'gD',0,0);
[uGP,graduGP] = FEMEvaluateGP(MeshQu,u_Qu);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_Qu = FEMIntegrate(MeshQu,EnergyDensity);
disp(sprintf('Quadratic elements %i, DOF = %i, energy = %7.5f',
             2*N, MeshQu.nDOF,Energy_Qu))
%  case 'Li4'
MeshLi4 = MeshQuad2Linear(MeshUpgrade(Mesh,'quadratic'));
u_Li4 = BVP2Dsym(MeshLi4,a,b,f,'gD',0,0);
[uGP,graduGP] = FEMEvaluateGP(MeshLi4,u_Li4);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_Li4 = FEMIntegrate(MeshLi4,EnergyDensity);
disp(sprintf('Linear elements %i, DOF = %i, energy = %7.5f',
             8*N, MeshLi4.nDOF,Energy_Li4))
%  case 'Cu'
```

```
MeshCu = MeshUpgrade(Mesh,'cubic');
u_Cu = BVP2Dsym(MeshCu,a,b,f,'gD',0,0);
[uGP,graduGP] = FEMEvaluateGP(MeshCu,u_Cu);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_Cu = FEMIntegrate(MeshCu,EnergyDensity);
disp(sprintf('Cubic elements %i, DOF = %i, energy = %7.5f',
             2*N, MeshCu.nDOF,Energy_Cu))
%  case 'Li9'
MeshLi9 = MeshCubic2Linear(MeshUpgrade(Mesh,'cubic'));
u_Li9 = BVP2Dsym(MeshLi9,a,b,f,'gD',0,0);
[uGP,graduGP] = FEMEvaluateGP(MeshLi9,u_Li9);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_Li9 = FEMIntegrate(MeshLi9,EnergyDensity);
disp(sprintf('Linear elements %i, DOF = %i, energy = %7.5f',
             18*N, MeshLi9.nDOF,Energy_Li9))
%  case 'He'
MeshHe = MeshUpgrade(Mesh,'Hermite');
u_He = BVP2Dsym(MeshHe,a,b,f,'gD',0,0);
[uGP,graduGP] = FEMEvaluateGP(MeshHe,u_He);
EnergyDensity = a/2*sum(graduGP.^2,2) + b/2*uGP.^2 - f*uGP;
Energy_He = FEMIntegrate(MeshHe,EnergyDensity);
disp(sprintf('Hermite elements %i, DOF = %i, energy = %7.5f',
             2*N, MeshHe.nDOF,Energy_He))

ind = [1:Ngrid^2];
MeshCu2 = MeshUpgrade(MeshCubic2Linear(MeshCu),'cubic');
u_Cu2 = BVP2Dsym(MeshCu2,a,b,f,'gD',0,0);  u_exact = u_Cu2(ind);
Error_Li  = sqrt(mean((u_Li-u_exact).^2))
Error_Qu  = sqrt(mean((u_Qu(ind)-u_exact).^2))
Error_Li4 = sqrt(mean((u_Li4(ind)-u_exact).^2))
Error_Cu  = sqrt(mean((u_Cu(ind)-u_exact).^2))
Error_Li9 = sqrt(mean((u_Li9(ind)-u_exact).^2))
Error_He  = sqrt(mean((u_He(ind)-u_exact).^2))

figure(11); FEMtrimesh(Mesh,u_He(ind)-u_exact);
            xlabel('x'); ylabel('y'); zlabel('difference')
figure(12); FEMtrimesh(MeshQu,u_Qu);
            xlabel('x'); ylabel('y'); zlabel('u')
figure(13); FEMtricontour(MeshQu,u_Qu,50);
            xlabel('x'); ylabel('y'); zlabel('u')
```

5.7 Are second order elements C1 conforming?

The command `BVP1D()` uses second order elements to solve two–point boundary value problems. Since the values of the piecewise quadratic functions coincide at the limit of two neighboring subintervals the elements are C^0 conforming, i.e. the numerical solution is continuous. The "open" question is whether these elements are C^1 conforming, i.e. are first derivatives continuous?

This is illustrated by the code below and the resulting Figure 66. The BVP solved is

$$-u''(x) = \text{sign}(x) \quad \text{for} \quad -1 < x < +1 \quad \text{and} \quad u(-1) = u(+1) = 0\ .$$

One might expect special behavior at $x = 0$.

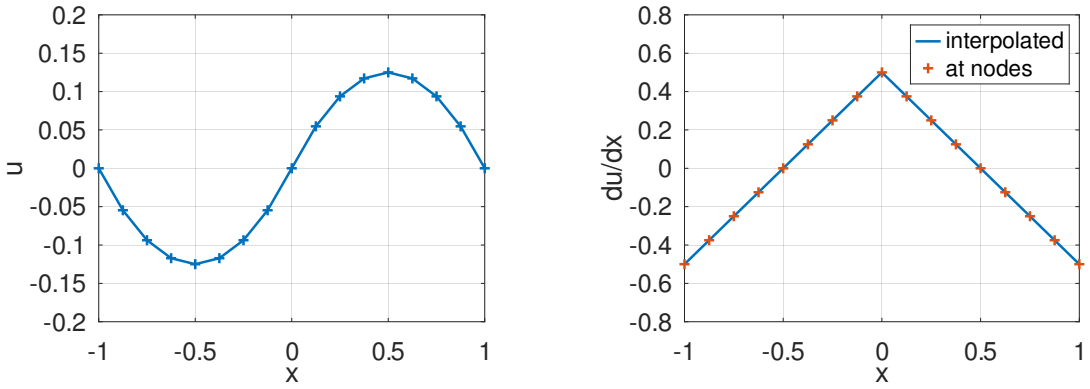
- The approximate solution $\vec{u}$ is determined by calling `BVP1D()`.
- The derivative is evaluated
  - by `pwquadinterp()` with a very high resolution.
  - by `FEM1DEvaluateDu()` at the nodes.

  Then both are plotted and no difference between the two is visible, i.e. the first derivative $u'(x)$ seems to be continuous. Zooming in around $x \approx 0$ confirms the observation.

Observe: since $-u''(x) = \pm 1$ is solved by $\mp\frac{1}{2}\,x^2$ the exact solution $u_{exact}(x)$ of the above problem consists of two quadratic functions, patched together at $x = 0$, in spite of the discontinous right hand side $\mathrm{sign}(x)$.

$$u_{exact}(x) = \begin{cases} \frac{1}{2}\,x\,(1+x) & \text{for} \quad -1 \le x \le 0 \\ \frac{1}{2}\,x\,(1-x) & \text{for} \quad 0 \le x \le +1 \end{cases}$$

The algorithm in `BVP1D()` is based on piecewise quadratic approximations and will thus generate the exact solution. Thus this example can not serve as a test for continuity of the first derivative.



(a) the solution $u(x)$ (b) the derivative $u'(x)$

Figure 66: The solution and the first derivative, evaluated at the nodes and by interpolation

Test_C1conforming.m

```
n = 8; x = linspace(-1,1,n+1)';
[x,u] = BVP1D(x,1,0,0,@(x)sign(x),1,0,0);
figure(1); plot(x,u,'+-')
           xlabel('x'); ylabel('u')
x_fine = linspace(-1,1,10001)'; [u_fine,du_fine] = pwquadinterp(x,u,x_fine);
du = FEM1DEvaluateDu(x,u);
figure(2); plot(x_fine,du_fine,x,du,'+')
           xlabel('x'); ylabel('du/dx'); legend('interpolated','at nodes')
```

As a second BVP examine

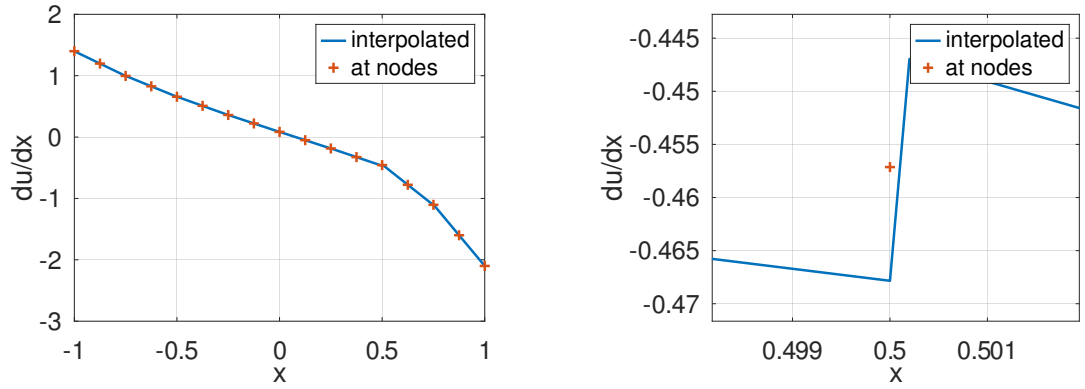
$$-u''(x) + \mathrm{sign}(x - 0.5)\,u'(x) = 1 \quad \text{for} \quad -1 < x < +1 \quad \text{and} \quad u(-1) = u(+1) = 0\,.$$

One might expect special behavior at $x = +0.5$.

- The approximate solution $\vec{u}$ is determined by calling `BVP1D()`.
- The derivative is evaluated
  - by `pwquadinterp()` with a very high resolution.
  - by `FEM1DEvaluateDu()` at the nodes.

  Then both are plotted and no difference between the two is visible in the full graph in Figure 67(a) i.e. the first derivative $u'(x)$ seems to be continuous. Zooming in around $x \approx 0.5$ leads to Figure 67(b) and a jump of the first derivative at $x = 0.5$ is visible.
- Since there is a node at $x = 0.5$ the code in `pwquadinterp()` returns the values of $u'(x)$ for the subinterval to the left of $x = 0.5$, and then jumps to the values on the right subinterval. `FEM1DEvaluateDu()` returns the average value of the slopes of $u(x)$ to the left and right of $x = 0.5$. This is visible in Figure 67(b).
- The size of the jump of $u'(x)$ is smaller if more elements are used, e.g. by `n=2*8` .

The consequence: second order element are not C^1 conforming. For 2D FEM algorithms the same is correct, visualized by Figure 61 on page 125.



(a) the derivative $u'(x)$ (b) zoom of the derivative $u'(x)$

Figure 67: The first derivative, complete graph and zoomed in at $x = 0.5$

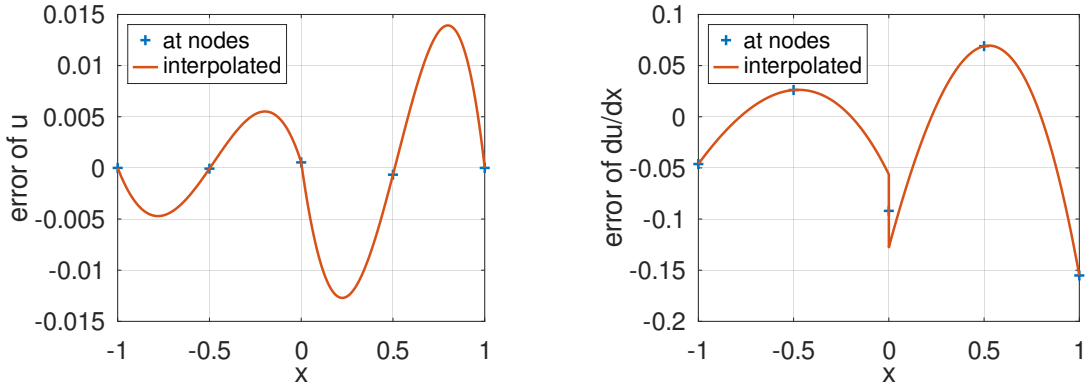
Test_C1conforming.m

```
n = 8; x = linspace(-1,1,n+1)';
[x,u] = BVP1D(x,1,@(x)sign(x-0.5),0,1,1,0,0);
figure(1); plot(x,u)
           xlabel('x'); ylabel('u')
x_fine = linspace(-1,1,10001)'; [u_fine,du_fine] = pwquadinterp(x,u,x_fine);
du = FEM1DEvaluateDu(x,u);
figure(2); plot(x_fine,du_fine,x,du,'+')
           xlabel('x'); ylabel('du/dx'); legend('interpolated','at nodes')
```

The above example might lead to the suspicion that the jump in the first derivative is caused by the noncontinuous coefficient $\text{sign}(x-0.5)$ of $u'(x)$. This is not the case, even extremely smooth problems lead to jumps in the derivative. The boundary value problem

$$-u''(x) = -\exp(x) \quad \text{for} \quad -1 < x < +1 \quad \text{and} \quad u(-1) = \exp(-1) \quad \text{and} \quad u(+1) = \exp(+1)$$

has the exact solution $u_{exact}(x) = \exp(x)$. Solving the problem with only 2 subintervals leads to the errors of $u(x)$ and $u'(x)$ in Figure 68. In Figure 68(b) the sizable jump of the derivative at $x = 0$ is clearly visible.



(a) error $u_{FEM}(x) - \exp(x)$ (b) error $u'_{FEM}(x) - \exp(x)$

Figure 68: Differences of the FEM solution $u_{FEM}(x)$ to the exact solution $u_{exact}(x) = \exp(x)$

Test_C1conforming.m

```
n = 2; x = linspace(-1,1,n+1)';
[x,u] = BVP1D(x,1,0,0,1,@(x)exp(x),exp(-1),exp(1));
figure(1); plot(x,u)
           xlabel('x'); ylabel('u')

x_fine = linspace(-1,1,10001)'; [u_fine,du_fine] = pwquadinterp(x,u,x_fine);
du = FEM1DEvaluateDu(x,u);
figure(2); plot(x_fine,du_fine,x,du,'+')
           xlabel('x'); ylabel('du/dx'); legend('interpolated','at nodes')
figure(3); plot(x,u,x_fine,u_fine,x_fine,exp(x_fine))
           xlabel('x'); ylabel('u');
           legend('at nodes','interpolated','exact','location','northwest');
figure(4); plot(x,u-exp(x),'+',x_fine,u_fine-exp(x_fine))
           xlabel('x'); ylabel('error of u')
           legend('at nodes','interpolated','location','northwest')
figure(5); plot(x,du-exp(x),'+',x_fine,du_fine-exp(x_fine))
           xlabel('x'); ylabel('error of du/dx')
           legend('at nodes','interpolated','location','northwest')
```

5.8 Contribution to the error by domain approximation

On a disk with radius $R = 1$ examine the boundary value problem

$$\begin{aligned} \Delta u &= 1 \qquad \text{for } x^2 + y^2 < R^2 \\ u(x,y) &= 0 \qquad \text{for } x^2 + y^2 = R^2 \end{aligned}$$

Since the solution will depend on the radius $r = \sqrt{x^2 + y^2}$ only the problem can be solved as a 1D boundary value problem.

$$\frac{\partial}{\partial r}\left(r \frac{\partial u(r)}{\partial r}\right) = r \quad \text{for } 0 < r < R \text{ and } \frac{\partial u(0)}{\partial r} = u(R) = 0$$

The code below solves these two problems and leads to the solutions in Figure 69.

DomainError.m

```
N = 36+1; alpha = linspace(0,2*pi,N); alpha = alpha(1:end-1)'; R = 1;
xy = [R*cos(alpha),R*sin(alpha),-ones(size(alpha))];
Mesh = CreateMeshTriangle('circle',xy,0.01);
Mesh = MeshUpgrade(Mesh,'quadratic');
%Mesh = MeshUpgrade(Mesh,'cubic');

a = 1; b0 = 0; bx = 0; by = 0;
u = BVP2D(Mesh,a,b0,bx,by,1,0,0,0);
figure(1); FEMtrimesh(Mesh,u); xlabel('x'); ylabel('y')

Interval = linspace(0,R,21);
[r,u1] = BVP1D(Interval,@(r)r,0,0,1,@(r)r,[0,0],0);
u2d = FEMgriddata(Mesh,u,r,zeros(size(r)));
figure(2); plot(r,u1,r,u2d); xlabel('r'); ylabel('u')
           legend('1D radial','2D slice')
```

The solution of the 1D approach equals the exact solution $u(r) = \frac{1}{4}(1 - r^2)$. The difference of the 1D and 2D solutions is shown on the left in Figure 70. The large contribution to the error appears close to the boundary at $r = R = 1$ and is caused by the approximate domain used for the 2D approach. The code generates a polygon with 10° segments and thus the approximate domain is smaller than the disk with radius $R = 1$. This is visible on the right in Figure 70. The approximate domain is smaller than the disk and thus the effect of the right hand side $f(x,y) = 1$ is smaller. The only difference is close to the boundary.

DomainError.m

```
figure(3); plot(r,u1-u2d); xlabel('r'); ylabel('difference of u')
figure(4); FEMtrimesh(Mesh); xlabel('x'); ylabel('y')
           alpha = linspace(0,2*pi,1001);
           hold on; plot(R*cos(alpha),R*sin(alpha),'k'); hold off
           xlim([0.9 1.02]); ylim([-0.2,0.2]);
```

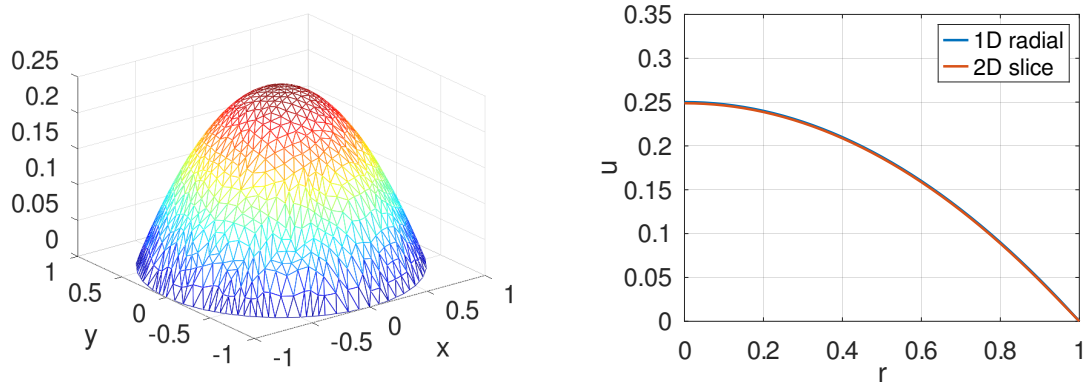


Figure 69: The 2D and 1D solutions of $\Delta u = 1$

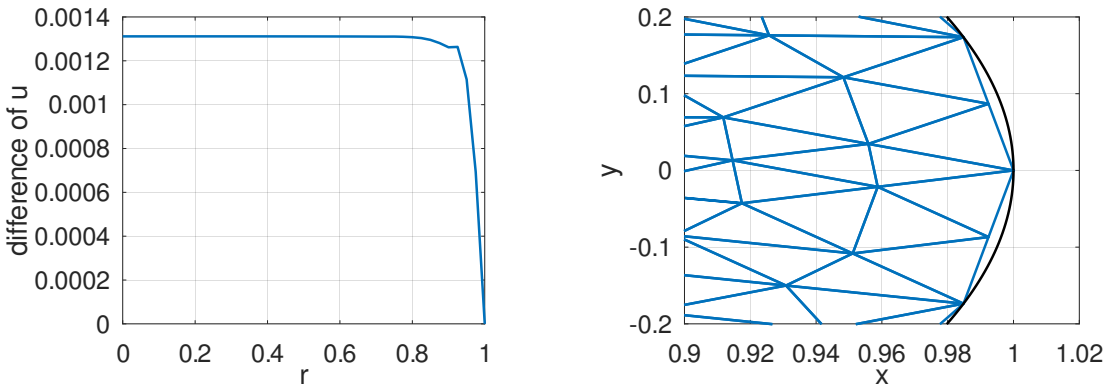


Figure 70: Difference of the two solutions and the mesh close to $(R, 0)$

There are three essential contributions to the difference of the approximate and exact solution:

1. The true solution is approximated by piecewise polynomials on each triangle. This contribution can be reduced by using quadratic or cubic polynomials.
2. On each triangle we have to use a numerical integration to determine the contributions. This contribution is usually very small, since the ingenious Gauss integration is used. The contribution can be relevant if the coefficients of the BVP are not continuous.
3. Not all domains $\Omega \subset \mathbb{R}^2$ can be decomposed into triangles. Curved boundary segements lead to approximation errors. This is the essential contribution to the error for this example.

To illustrate the above statements modify the code in `DomainError.m`.

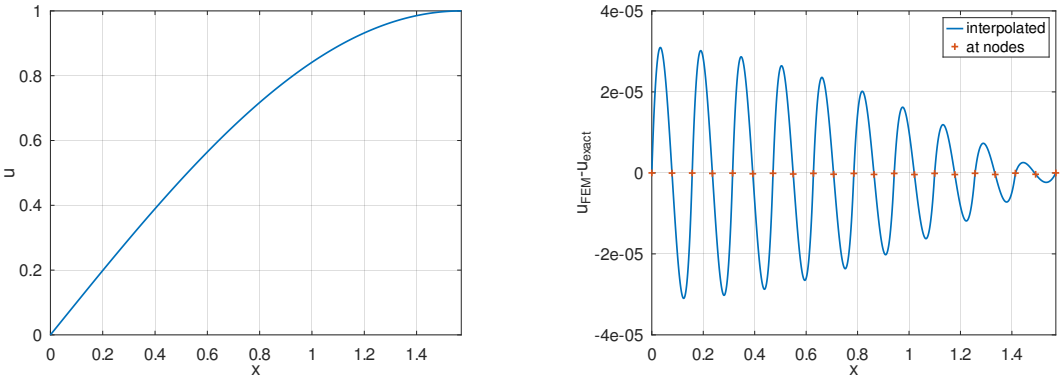
- Use linear, quadratic or cubic elements and observe no essential change of the error.
- Use a finer grid and observe no essential change of the error.
- Work with more than 36 boundary points and observe that the error will be smaller.

5.9 Superconvergence for a 1D BVP

Examine the solution of the 1D BVP

$$-u'' = \sin(x) \quad \text{for} \quad 0 \le x \le \frac{\pi}{2} \quad \text{with} \quad u(0) = 0 \quad \text{and} \quad u'(\frac{\pi}{2}) = 0$$

with the exact solution $u_{\text{exact}}(x) = \sin(x)$. Use the command `BVP1D()` to find the numerical approximation u_{FEM} on a coarse grid, and then `pwquadinterp()` to evaluate the solution on a finer grid.



(a) the solution

(b) difference to the exact solution

Figure 71: The solution and difference to the exact solution for a 1D BVP

Observe that

- the solution is rather accurate, even on a grid with few nodes, e.g. 21 nodes.
- The solution is considerably more accurate at the nodes, than in-between nodes. This effect is called superconvergence. You can not count on the effect of superconvergence, as it might not happen in your problem. The convergence in the L_2 norm is given by the theoretical results in Section 6.8 on page 200.

BVP1DSuperconvergence.m

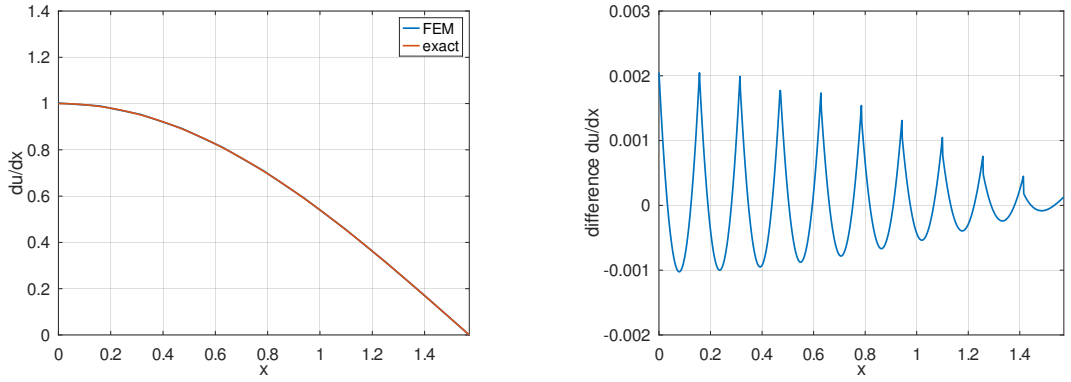
```
N = 10;              % number of elements, then 2*N+1 nodes
x = linspace(0,pi/2,N+1);
[xn,u] = BVP1D(x,1,0,0,1,@(x)-sin(x),0,1);

x_fine = linspace(0,pi/2,1001);
[u_fine du_fine, ddu_fine] = pwquadinterp(xn,u,x_fine);
figure(1); plot(x_fine,u_fine)
           xlabel('x'); ylabel('u')
figure(11); plot(x_fine,u_fine-sin(x_fine),xn,u-sin(xn),'+')
            xlabel('x'); ylabel('u_{FEM}-u_{exact}'); xlim([min(x_fine),max(x_fine)])
           legend('interpolated','at nodes')
```

The results generated by `pwquadinterp()` allow to display the derivatives and the deviation of the derivatives.

Observe that

- the first derivative is piecewise linear, but it is hard to see without zooming in.
- the difference to the exact derivative oscillates on each subinterval.



(a) first derivative of the solution

(b) difference of the first derivatives

Figure 72: The derivative of the numerical solution and difference to the exact solution

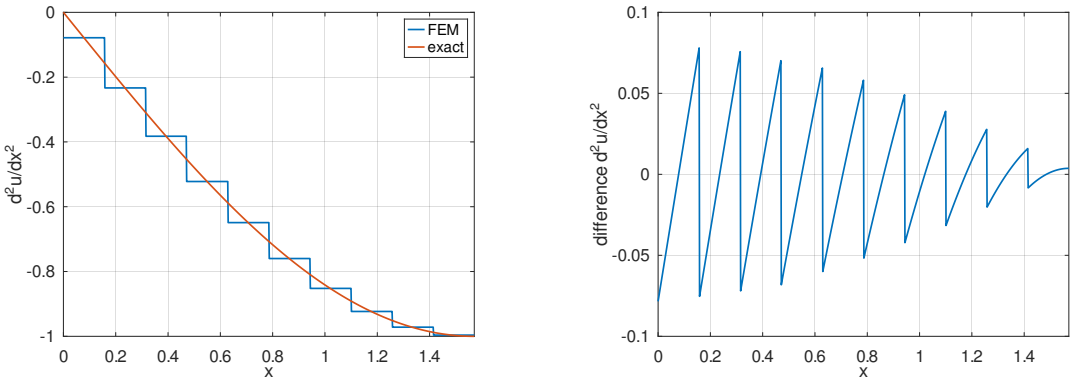
BVP1DSuperconvergence.m

```
figure(12); plot(x_fine,du_fine-cos(x_fine))
            xlabel('x'); ylabel('difference du/dx'); xlim([min(x_fine),max(x_fine)])
figure(3); plot(x_fine,ddu_fine,x_fine,-sin(x_fine))
            xlabel('x'); ylabel('d^2u/dx^2'); xlim([min(x_fine),max(x_fine)])
            legend('FEM','exact')
figure(13); plot(x_fine,ddu_fine+sin(x_fine))
            xlabel('x'); ylabel('difference d^2u/dx^2'); xlim([min(x_fine),max(x_fine)])
```

The results generated by `pwquadinterp()` allow to display the second derivatives and the deviation of the derivatives.

Observe that

- the second derivative is piecewise constant. This should be no surprise, since piecewise quadratic functions are used the generate the FEM solution.
- the difference to the exact second derivative changes the sign on each subinterval.



(a) second derivative of the solution

(b) difference of the second derivatives

Figure 73: The second derivative of the numerical solution and difference to the exact solution

5.10 Convergence of the modified Crank–Nicolson time steppers for semilinear problems

Examine the convergence of the modified Crank–Nicolson time stepper, presented in Section 7.6.6 (page 218) and used with the commands `IBVP1DNL()` and `IBVP2DNL()`. As example use a semilinear heat conduction problem in 2D

$$\begin{aligned} \frac{\partial}{\partial t}\, u - \Delta u &= u\,(1-u) + \exp(-2\,t)\,\sin^2(x)\,\sin^2(y) && \text{for } 0 < x, y < \pi, \quad t > 0 \\ u &= 0 && \text{on boundary} \\ u(x,y,0) &= \sin(x)\,\sin(y) \end{aligned}$$

with the exact solution

$$u(x,y,t) = \exp(-t)\,\sin(x)\,\sin(y)\ .$$

The code `HeatSemilinearLoop.m` shown below leads to Table 14 and Figure 74. Maximal errors and RMS values are examined.

- For the Crank–Nicolson algorithm CNPC with the predictor–corrector step for the nonlinear contribution:
  - for the half size steps the error is divided by 4.
  - the slope in the double logaritmic plot is 2.
  - the observed convergence is of order 2.
- For the Crank–Nicolson algorithm CNEXP with the explicit expression for the nonlinear contribution:
  - for the half size steps the error is divided by 2.
  - the slope in the double logaritmic plot is 1.
  - the observed convergence is of order 1.

These results clearly indicate that (most often) the stepper CNPC is more efficient than CNEXP, even if the computational effort for one time step is doubled.

HeatSemilinearLoop.m

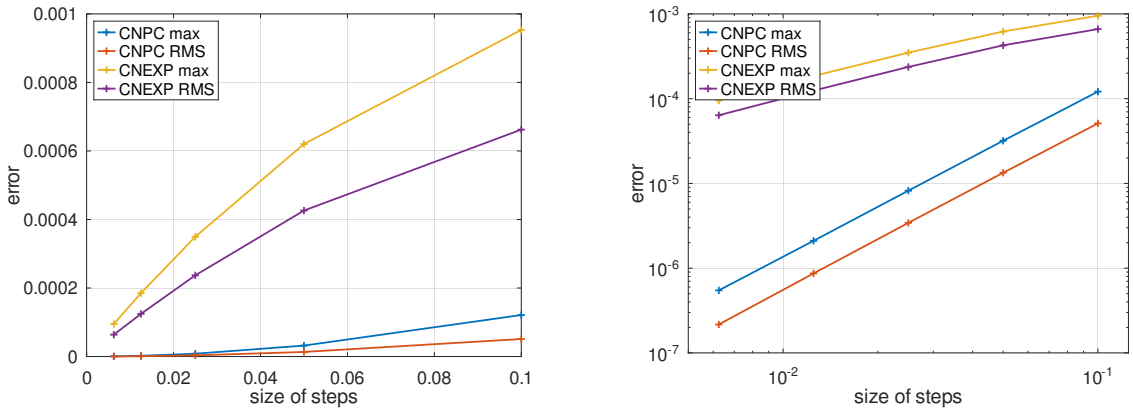
```
h  = 0.1; %%typical radius of triangles
nodes = [0 0 -1; pi 0, -1; pi pi -1; 0 pi -1];
FEMmesh = CreateMeshTriangle("test",nodes,h^2/2);
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

f = @(xy,t,u) u.*(1-u) + exp(-2*t)*sin(xy(:,1)).^2 .* sin(xy(:,2)).^2;
t_end = 1;
u0 = @(xy)sin(xy(:,1)) .* sin(xy(:,2));
u_exact = exp(-t_end)*sin(FEMmesh.nodes(:,1)) .* sin(FEMmesh.nodes(:,2));

Nsteps = [10,20,40,80,160];
```

	CNPC, predictor–corrector		CNEXP, explicit	
time steps	maximal	RMS	maximal	RMS
10	1.2113e-04	5.1133e-05	9.5328e-04	6.6238e-04
20	3.1893e-05	1.3362e-05	6.2027e-04	4.2606e-04
40	8.2164e-06	3.4258e-06	3.4952e-04	2.3698e-04
80	2.1039e-06	8.6741e-07	1.8506e-04	1.2450e-04
160	5.4723e-07	2.1655e-07	9.5170e-05	6.3761e-05

Table 14: Errors for the modified Crank–Nicolson time steppers, maximal values and root mean squared. Observe the quadratic convergence for the CNPC algorithm and the linear convergence for CNEXP.



(a) linear plot (b) double logarithmic plot

Figure 74: Errors for the modified Crank–Nicolson time steppers

```
Diff_Max_RMS = zeros(length(Nsteps),4);
tic();
for jj = 1:length(Nsteps)
  Steps = [1,Nsteps(jj)];
  Solver = 'CNPC';
  [u_dyn,t] = IBVP2DNL(FEMmesh,1,1,0,0,0,f,0,0,0,u0,0,t_end,Steps,'solver',Solver);
  u_end = u_dyn(:,end);
  Difference1 = abs(u_end-u_exact);
  Solver = 'CNexp';
  [u_dyn,t] = IBVP2DNL(FEMmesh,1,1,0,0,0,f,0,0,0,u0,0,t_end,Steps,'solver',Solver);
  u_end = u_dyn(:,end);
  Difference2 = abs(u_end-u_exact);
  Diff_Max_RMS(jj,:) = [max(Difference1),sqrt(mean(Difference1.^2)),...
                        max(Difference2),sqrt(mean(Difference2.^2))];
endfor
timing = toc()
h = t_end./Nsteps;
figure(1); plot(h,Diff_Max_RMS(:,1),'+ -',h,Diff_Max_RMS(:,2),'+ -',...
               h,Diff_Max_RMS(:,3),'+ -',h,Diff_Max_RMS(:,4),'+ -')
           xlabel('size of steps'); ylabel('error');
           legend('CNPC max', 'CNPC RMS','CNEXP max','CNEXP RMS','location','northwest');
figure(2); loglog(h,Diff_Max_RMS(:,1),'+ -',h,Diff_Max_RMS(:,2),'+ -',...
                 h,Diff_Max_RMS(:,3),'+ -',h,Diff_Max_RMS(:,4),'+ -')
           xlabel('size of steps'); ylabel('error'); xlim([1/200,1/8]);
           legend('CNPC max', 'CNPC RMS','CNEXP max','CNEXP RMS','location','northwest');
```

5.11 Stability of the time steppers, or lack thereof

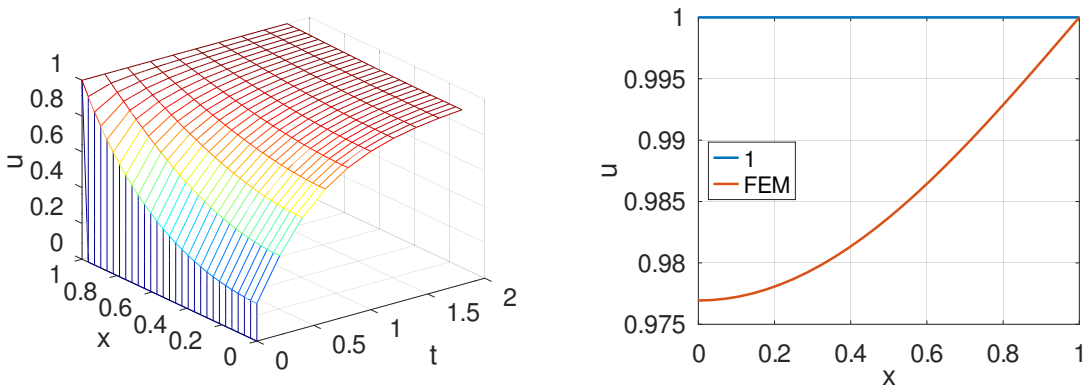
Examine the initial boundary value problem

$$\begin{array}{rcll} \frac{\partial}{\partial t}\,u(x,t) & = & \frac{\partial^2}{\partial x^2}\,u(x,t) & \text{for} \quad 0<x<1 \\ \frac{\partial}{\partial x}\,u(0,t) & = & 0 & \text{for} \quad t>0 \\ u(1,t) & = & 1 & \text{for} \quad t>0 \\ u(x,0) & = & 0 & \text{for} \quad 0<x<1 \end{array} \quad .$$

Observe that the initial condition $u(x,0)=0$ does not satisfy the boundary condition $u(1,t)=1$. As a consequence the behavior of the solution will be critical for x close to 1 . Examine the FEM solution generated by different time stepping algorithms. On the interval $[0,1]$ with 14 elements of order 2 use 10 time steps to find the solution at time $t=2$.

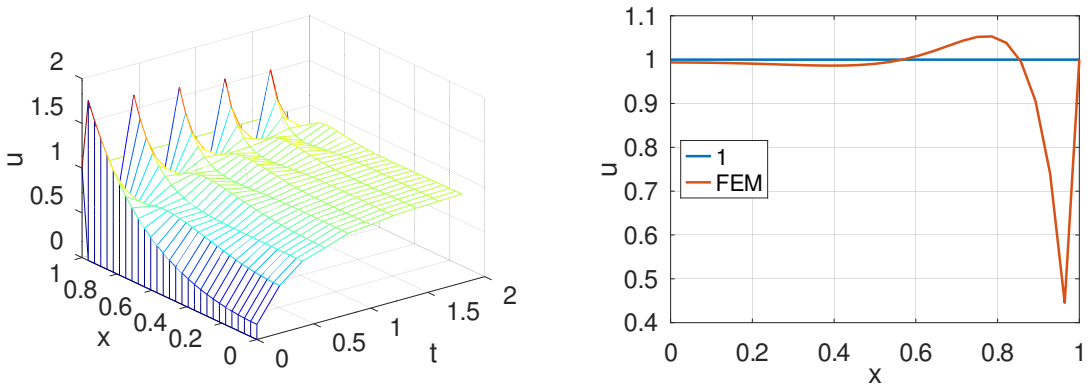
- The implicit time stepper is unconditionally stable and L–stable. Thus expect convergence of the solution, but slow convergence, since the scheme is only consistent of order 1 . Find the result in Figure 75.
- The Crank–Nicolson time stepper is unconditionally stable, but not L–stable and consistent of order 2 . Thus expect convergence of the solution, but CN will have a hard time to deal with the inconsistent initial condition. Find the result in Figure 76 and the problem around $x=1$ is obvious. The solution at $x=1$ oscillates wildly from step to step. This is caused by the stability function $g(z)=\frac{2-z}{2+z}$ for the Crank–Nicolson stepper (see page 216). For very large z the stability condition $|g(z)|<1$ is satisfied, but $g(z)$ is very close to -1. Thus instead of getting very small the corresponding contribution (eigen mode) will almost keep its amplitude, but flip its sign at each step.
- The explicit time stepper is only conditionally stable. Thus expect serious trouble for large time steps. Taking only 10 time steps leads to a large Δt. Find the (obviously invalid) result in Figure 77(a).
- The implicit Runge–Kutta time stepper is unconditionally stable, L–stable and and consistent of order 2 . Thus expect convergence of the solution, and it will take care of the inconsistent initial condition. Find the result in Figure 77(b).

Computations with finer grids and smaller time steps show that the value of $u(x,2)\approx 0.991$ is a good approximation of the true value.



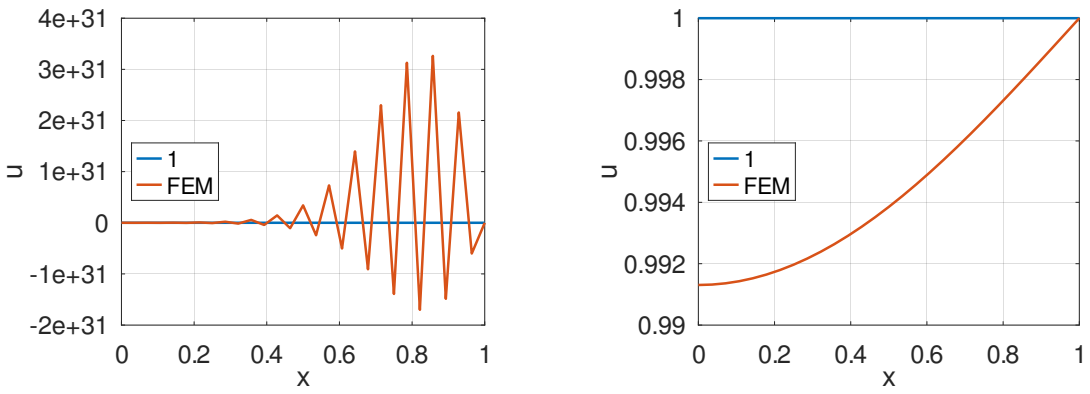
(a) as function of x and t (b) final solution at $t = 2$

Figure 75: The solution generated by the implicit time stepper



(a) as function of x and t (b) final solution at $t = 2$

Figure 76: The solution generated by the Crank–Nicolson time stepper



(a) explicit time stepper (b) Runke–Kutta time stepper

Figure 77: The solution at time $t = 2$ generated by the explicit and Runge–Kutta time steppers

Stability1D_dynamic.m

```
BCleft = [0,0]; BCright = 1; xMax = 1;
interval = linspace(0,xMax,15); u0 = 0;
t0 = 0 ; tend = 2; steps = [10,1];
w = 1; a = 1; b = 0; c = 0; d = 1; f = 0;
solver = 'implicit';  %% select one solver
%% solver = 'explicit';
%% solver = 'CN';
%% solver = 'RK';
[x,u,t] = IBVP1D(interval,w,a,b,c,d,f,BCleft,BCright,u0,t0,tend,steps,'solver',solver);

figure(1); mesh(t,x,u); ylim([0,xMax])
           xlabel('t'); ylabel('x'); zlabel('u')
u_at_0 = u(1,end)

u_ones = ones(size(x));
figure(2); plot(x,u_ones,x,u(:,end)); xlim([0,xMax])
           xlabel('x'); ylabel('u');  legend('1', 'FEM','location','west')
```

5.12 Conditional stability of the explicit time stepper for a wave equation

For heat problems the explicit solver is only conditional stable, i.e. the time steps Δt have to be small enough, see the above Section 5.11. The same is the case for hyperbolic problems. Examine the example in Section 3.9, i.e. solve $\frac{\partial^2}{\partial t^2}\, u = \frac{\partial^2}{\partial x^2}\, u$. The system of ODEs solved in `I2BVP1D()` is $\mathbf{W}_2 \frac{d^2}{dt^2}\,\vec{u}(t) = \mathbf{A}\,\vec{u}(t)$ and the stability condition is using the largest generalized eigenvalue λ_{max} of $\mathbf{A}\,\vec{u} = \lambda\,\mathbf{W}_2\vec{u}$ and given by

$$\Delta t \leq \frac{2}{\sqrt{\lambda_{max}}}\,.$$

The command `I2BVP1D()` will issue a warning if this condition is violated for the explicit solver, but attempts to return results anyhow.

To obtain the stability condition examine the discretization of the ordinary differential equation $\ddot{u} = -\lambda\, u$.

$$\begin{aligned}
u_{i-1} - 2\,u_i + u_{u+1} &= -(\Delta t)^2\lambda\, u_i \\
\begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix} &= \begin{pmatrix} u_i \\ (-(\Delta t)^2\lambda + 2)\,u_i - u_{i-1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2-(\Delta t)^2\lambda \end{bmatrix} \begin{pmatrix} u_{i-1} \\ u_i \end{pmatrix} \\
0 &= \det \begin{bmatrix} 0-\mu & 1 \\ -1 & (\Delta t)^2\lambda + 2 - \mu \end{bmatrix} = \mu^2 - (2-(\Delta t)^2\lambda)\,\mu + 1 \\
&= \mu^2 - (\mu_1+\mu_2)\,\mu + \mu_1\mu_2
\end{aligned}$$

Thus conclude that $\mu_1 \cdot \mu_2 = 1$. For the system to be stable the eigenvalues μ_i have to satisfy $|\mu_i| = 1$. This is the case iff the μ_i are complex, thus the discriminant of the quadratic equation has to be negative.

$$\begin{aligned}
0 &\geq (2-(\Delta t)^2\lambda)^2 - 4 = -4\,(\Delta t)^2\lambda + (\Delta t)^4\lambda^2 = (\Delta t)^2\lambda\,(-4 + (\Delta t)^2\,\lambda) \\
\lambda &\leq \frac{4}{(\Delta t)^2} \quad \text{or} \quad \Delta t \leq \frac{2}{\sqrt{\lambda}}
\end{aligned}$$

For the second order elements used in `I2BVP1D()` the value of the largest generalized eigenvalue λ_{max} is approximately[17] proportional to $(\Delta x)^2$. This allows to adapt the time step Δt such that the algorithm is stable. In the source code of `I2BVP1D.m` (or `I2BVP2D.m`) uncomment the lines below to observe the critical values.

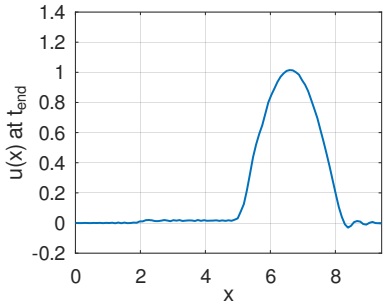
```
%% lambda = eigs(A,W2,1);
%% disp(sprintf("Values:  lambda = %g, dt = %g, 2/sqrt(lambda) = %g\n",...
%%                       lambda,dt,2/sqrt(lambda)))
```

[17] I am not aware of an exact formula.

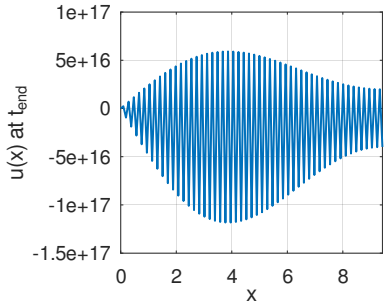
In the code `WaveExplicitTest.m` select the solver (explicit or implicit) and the number of time steps (105 or 100).

- For Figure 78(a) the explicit solver with 105 time steps is used. The value of Δt is just small enough for the algorithm to be stable.
- For Figure 78(b) the explicit solver with 100 time steps is used. The value of Δt is slightly to large for the algorithm to be stable. The blowup of the rapidly oscillating solution is obvious.
- For Figure 78(c) the implicit solver with 100 time steps is used. The implicit algorithm is unconditionally stable.

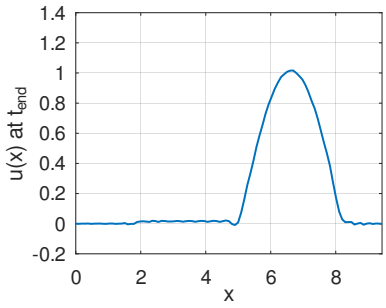
This example should clearly illustrate that respecting the stability condition for the explicit solver is essential. For the unstable situation you will in most cases not obtain number at all, but NaNs. This author recommends to use the implicit solver for 1D problems. The similar solver `I2BVP2D()` for 2D problems shows the same stability behavior.



(a) explicit solver with 105 steps



(b) explicit solver with 100 steps



(c) implicit solver with 100 steps

Figure 78: Solutions of the wave equation at the time $t = t_{end}$

WaveExplicitTest.m

```
%% test the stability 1D wave equation
if 1
 solver = 'explicit';
else
 solver = 'implicit';
endif

if 1
  steps = [20,5];  %% unstable for explicit solver
else
  steps = [21,5];  %% stable for both solvers
endif

a = 1; b = 0; c = 0; d = 1; f = 0; w2 = 1; w1 = 0; BCleft = 0; BCright = [0,0];
t0 = 0; tend = 5; interval = linspace(0,3*pi,51)';
```

```
u0 = @(x)sin(x).*(x<=pi); u1 = @(x)-cos(x).*(x<=pi);
[x,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,...
                  steps,'solver',solver);
figure(11); clf; mesh(t,x,u); xlabel('time t'); ylabel('position x'); zlabel('u')
            xlim([min(t),max(t)]); ylim([min(x),max(x)])
figure(12); clf; contour(t,x,u,21); xlabel('time t'); ylabel('position x');
figure(13); plot(x,u(:,end)); xlabel('x'); ylabel('u(x) at t_{end}');
            xlim([min(x),max(x)])
```

5.13 The shear–locking effect caused by linear elements

Examine a domain $\Omega = [\frac{-L}{2}, \frac{+L}{2}] \times [\frac{-H}{2}, \frac{+H}{2}] \subset \mathbb{R}^2$ with $L = H = 0.1$ and apply a horizontal deformation u_1 on the left and right edges at $x = \frac{\pm L}{2}$ of size $\pm c\, y = \pm 5 \cdot 10^{-4}\, y$. The upper and lower edge are force free. Use the material parameters $E = 100 \cdot 10^9$ and $\nu = 0$. Find the original and deformed domain in Figure 79. One can verify[18]

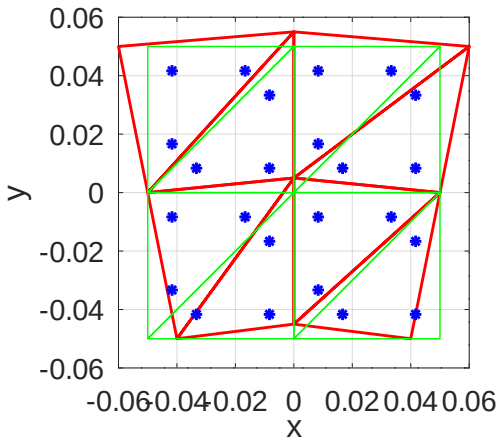


Figure 79: Original and deformed domain and the Gauss integration points for linear elements

that an exact solution of the boundary value problem is given by

$$u_1(x,y) = \frac{2\,c}{L}\,x\,y \quad , \quad u_2(x,y) = \frac{c}{L}\,(\frac{L^2}{4} - x^2) \quad \text{where} \quad c = 5 \cdot 10^{-4}\,.$$

This exact solution leads to the strains $\varepsilon_{xx} = \frac{2c}{L}\,y$ and $\varepsilon_{yy} = \varepsilon_{xy} = 0$ and thus the elastic energy (use the elastic energy density (28) with $\nu = 0$)

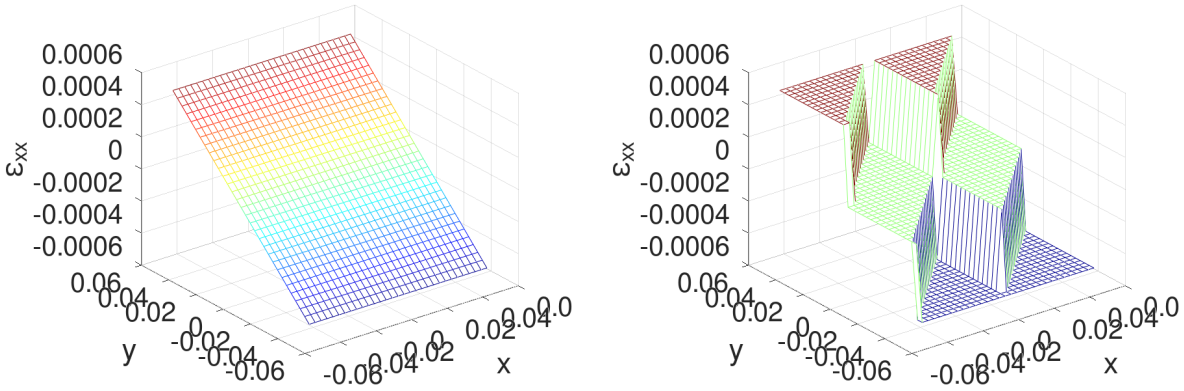
$$\begin{aligned} U_{elast} &= U_{\varepsilon_{xx}} + U_{\varepsilon_{yy}} + U_{\varepsilon_{xy}} \\ &= \frac{E}{2} \iint_\Omega \varepsilon_{xx}^2\, dA + \frac{E}{2} \iint_\Omega \varepsilon_{yy}^2\, dA + \frac{E}{2} \iint_\Omega 2\,\varepsilon_{xy}^2\, dA \\ &= \frac{E}{2} \int_{-H/2}^{+H/2} \int_{-L/2}^{+L/2} \frac{4\,c^2}{L^2}\, y^2\, dx\, dy + 0 + 0 = \frac{E}{2}\,\frac{4\,c^2}{L^2}\,L\,\frac{2\,H^3}{3\,8} = \frac{125}{3} \approx 41.667 \end{aligned}$$

Determine approximate solutions of this plane stress problem with NH=NL layers in either direction and using either linear or quadratic elements. Then use these solutions $\vec{u}_1$ and $\vec{u}_2$ and the function `FEMgriddata()` to evaluate

[18] E.g. use [Stah08, §5] with $\nu = 0$ and $u_1 = x\,y$ and $u_2 = -\frac{x^2}{2}$ to arrive at

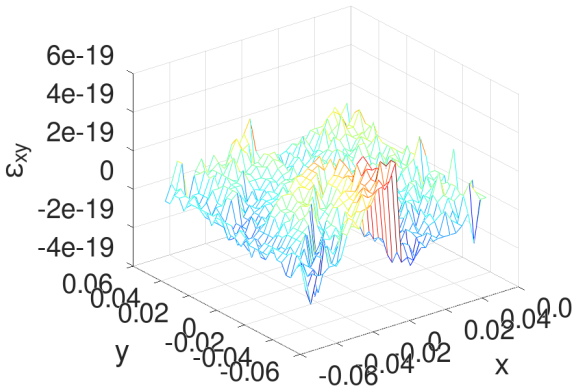
$$0 \stackrel{?}{=} \frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} + \frac{\partial}{\partial x}\left(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}\right) = +0 + 0 + \frac{\partial}{\partial x}\,(y + 0) \qquad \text{OK}$$

$$0 \stackrel{?}{=} \frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} + \frac{\partial}{\partial y}\left(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}\right) = -1 + 0 + \frac{\partial}{\partial y}\,(y + 0) \qquad \text{OK}$$

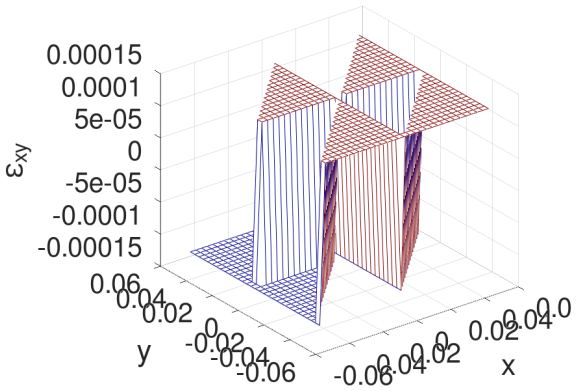


(a) ε_{xx} with quadratic elements

(b) ε_{xx} with linear elements



(c) ε_{xy} with quadratic elements



(d) ε_{xy} with linear elements

Figure 80: The strains ε_{xx} and ε_{xy} with two layers in each direction for linear and quadratic elements

the strains ε_{xx}, ε_{yy} and ε_{xy} on a fine xy–grid. Find the results for two layers (NL=NH=2) in Figure 80. Observe that the strains obtained by quadratic elements are very close to the strains of the exact solution. The strains based on linear elements show some surprising features:

- The strains are piecewise constant! This should be no surprise, since a partial derivative of order one of a piecewise linear function leads to a piecewise constant strain function. For this reason first order triangular elements are also called **Constant Strain Triangles**, or short CST elements.
- The piecewise constant approximation of the normal strain ε_{xx} is as good as can be, since only 8 triangular elements are used with this mesh.
- The piecewise approximation of the shearing strain ε_{xy} is drastically different from the exact value 0 . This is caused by the two contributions to $\varepsilon_{xy} = \frac{1}{2}(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x})$, which do not cancel out on the piecewise constant sections. The approximation based on second order elements is quite good, since $10^{-19} \approx 0$.

To examine the stiffness of the deformed body compute the elastic energy put into the body by the deformation. To arrive at reliable values a couple of steps are performed:

1. Generate a rather fine grid on the domain Ω, using the command `meshgrid()`.
2. Evaluate the partial derivatives of u_1 and u_2 on the grid with the help of `FEMgriddata()`. Then compute the three strains on the fine grid.
3. Use the command `mesh()` to visualize a few strains, leading to Figure 80.
4. With the strains use the expressions for the elastic energy density to evaluate the different contributions.
5. Use an iterated trapezoidal rule (`trapz()`) to perform the numerical integration for the three contributions to the elastic energy.

The above is performed for different numbers of layers of first and second order elements. Find the results in Table 15. This table shows a few, possibly surprising, results.

element type	# of layers	U_{elast}	$U_{\varepsilon_{xx}}$	$U_{\varepsilon_{yy}}$	$U_{\varepsilon_{xy}}$
exact		41.666	41.666	0	0
quadratic	NL=NH=1	41.759	41.759	0	0
quadratic	NL=NH=2	41.759	41.759	0	0
quadratic	NL=NH=5	41.759	41.759	0	0
linear	NL=NH=1	187.5	125	0	62.5
linear	NL=NH=2	78.472	62.847	0	15.625
linear	NL=NH=5	48.122	45.622	0	2.500
linear	NL=NH=10	43.850	43.225	0	0.625

Table 15: Elastic energy contributions for shearing

- The results generated with second order elements are very accurate, even for one layer only. This is caused by the fact that the exact solution is a polynomial of degree 2 and thus can be represented exactly by second order elements. The remaining, small difference can be made smaller by using a better integration scheme. See the remarks below (page 149), where an exact result is obtained.
- The results based on linear elements are severely different. The elastic energy is considerably too high and thus the solid considered to be much stiffer than it actually is. There are two contributions to this no-desirable effect:
  1. The piecewise constant patches lead to larger integrals.
  2. The shearing contribution by ε_{xy} does not vanish. The effect is often called **shear locking**.

  For a small number of layers the effect is drastic, for larger number of layers the effect becomes smaller.

ShearLocking.m

```
L = 0.1; H = 0.1; E = 100e9; nu = 0;
%% shearing of elements by applied displacement
NL = 2;    %% elements along length L
NH = NL;   %% elements along height H
Order = 1; %% order of elements, either 1 or 2

FEMmesh = CreateMeshRect([-L/2:L/NL:L/2],[-H/2:H/NH:+H/2],-22,-22,-11,-11);
if Order==2
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
endif

function res = gD1(xy)
  Disp = 0.01;
  res = Disp*xy(:,1).*xy(:,2);
endfunction

[u1,u2] = PlaneStress(FEMmesh,E,nu,{0,0},{'gD1',0},{0,0});
figure(2); FEMtrimesh(FEMmesh,u1); xlabel('x'); ylabel('y'); zlabel('u1')
figure(3); FEMtrimesh(FEMmesh,u2); xlabel('x'); ylabel('y'); zlabel('u2')

figure(1);   factor = 4e2;
trimesh(FEMmesh.elem,FEMmesh.nodes(:,1)+factor*u1,FEMmesh.nodes(:,2)+factor*u2,...
       'color','red','linewidth',2);
hold on ;
trimesh(FEMmesh.elem,FEMmesh.nodes(:,1),FEMmesh.nodes(:,2),...
       'color','green','linewidth',1);
plot(FEMmesh.GP(:,1),FEMmesh.GP(:,2),'b*');
hold off; xlabel('x'); ylabel('y'); xlim([-0.06,+0.06]); ylim([-0.06,+0.06]); axis equal

%% generate the data on the grid
x = linspace(-L/2,L/2,31); y = linspace(-H/2,+H/2,31); [xx,yy] = meshgrid(x,y);
[u1i,eps_xxi,eps_xy1i] = FEMgriddata(FEMmesh,u1,xx,yy);
[u2i,eps_xy2i,eps_yyi] = FEMgriddata(FEMmesh,u2,xx,yy);
eps_xyi = (eps_xy1i+eps_xy2i)/2;

figure(12); mesh(xx,yy,eps_xxi); xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(13); mesh(xx,yy,eps_yyi); xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')
figure(14); mesh(xx,yy,eps_xyi); xlabel('x'); ylabel('y'); zlabel('\epsilon_{xy}')

Wi = 0.5*E/(1-nu^2)*(eps_xxi.^2 + eps_yyi.^2+2*nu*eps_xxi.*eps_yyi+2*(1-nu)*eps_xyi.^2);
Wxxi = 0.5*E/(1-nu^2)*(eps_xxi.^2);
Wyyi = 0.5*E/(1-nu^2)*(eps_yyi.^2);
Wxxyyi = 0.5*E/(1-nu^2)*(2*nu*eps_xxi.*eps_yyi);
Wxyi = 0.5*E/(1-nu^2)*(2*(1-nu)*eps_xyi.^2);

figure(15); mesh(xx,yy,Wi);xlabel('x'); ylabel('y'); title('energy density')

EnergiesGrid = [trapz(x,trapz(y,Wi)),trapz(x,trapz(y,Wxxi)),...
                trapz(x,trapz(y,Wyyi)),trapz(x,trapz(y,Wxyi))]
```

The evaluation on a fine grid might seems unnecessary, since FEMoctave provides `EvaluateStrain()` to determine the values of the strains at the nodes. Then determine the contributions to the energy densities and integrate using `FEMIntegrate()`.

- The results for second order meshes seem reasonable.
- The results based on linear meshes are off, values and graphics. This is caused by the algorithms used:
  1. `EvaluateStrain()` returns values at the nodes. For the derivatives the average value of the neighboring elements are used, not the values inside the elements.

2. `FEMIntegrate()` will then take those values at the nodes and (for linear elements) apply a piecewise linear interpolation, followed by a Gauss integration. Thus the values used for the integration are drastically different form the values used when the equation was solved.

ShearLocking.m

```
%%% evaluate at the nodes
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(FEMmesh,u1,u2);

W = 0.5*E/(1-nu^2)*(eps_xx.^2 + eps_yy.^2+2*nu*eps_xx.*eps_yy+2*(1-nu)*eps_xy.^2);
Wxx = 0.5*E/(1-nu^2)*(eps_xx.^2);
Wyy = 0.5*E/(1-nu^2)*(eps_yy.^2);
Wxy = 0.5*E/(1-nu^2)*(2*(1-nu)*eps_xy.^2);

%% integration results are not reliable
EnergiesFEMIntegrate = [FEMIntegrate(FEMmesh,W),FEMIntegrate(FEMmesh,Wxx),...
                        FEMIntegrate(FEMmesh,Wyy),FEMIntegrate(FEMmesh,Wxy)]
figure(4); FEMtrimesh(FEMmesh,W);
           xlabel('x'); ylabel('y'); title('energy density, on nodes'); view([-50,20])
```

The above problem can be removed by evaluating the partial derivatives at the Gauss points, instead of the nodes. Use `FEMEvaluateGP()` to determine the contributions to the elastic energy density. Then integrate with `FEMIntegrate()`.

ShearLocking.m

```
%% integrate by evaluation at the Gauss points
[u1G,gradU1] = FEMEvaluateGP(FEMmesh,u1);
[u2G,gradU2] = FEMEvaluateGP(FEMmesh,u2);
eps_xxG = gradU1(:,1); eps_yyG = gradU2(:,2); eps_xyG = (gradU1(:,2)+gradU2(:,1))/2;
W = 0.5*E/(1-nu^2)*(eps_xxG.^2 + eps_yyG.^2+2*nu*eps_xxG.*eps_yyG+2*(1-nu)*eps_xyG.^2);
Wxx = 0.5*E/(1-nu^2)*(eps_xxG.^2);
Wyy = 0.5*E/(1-nu^2)*(eps_yyG.^2);
Wxxyy = 0.5*E/(1-nu^2)*(2*nu*eps_xxG.*eps_yyG);
Wxy = 0.5*E/(1-nu^2)*(2*(1-nu)*eps_xyG.^2);
EnergiesFEMIntegrateGauss = [FEMIntegrate(FEMmesh,W),FEMIntegrate(FEMmesh,Wxx),...
                             FEMIntegrate(FEMmesh,Wyy),FEMIntegrate(FEMmesh,Wxy)]
```

Below find the results for two layers NL=NH=2 and first and second order elements. Shown are in that order

$$
\begin{aligned}
\iint_\Omega \mathrm{W} &= \iint_\Omega \mathrm{Wxx} + \mathrm{Wyy} + \mathrm{Wxy} + \mathrm{Wxxyy} \\
\iint_\Omega \mathrm{Wxx} &= \frac{E}{2\left(1-\nu^2\right)} \iint_\Omega \varepsilon_{xx}^2 \, dA \\
\iint_\Omega \mathrm{Wyy} &= \frac{E}{2\left(1-\nu^2\right)} \iint_\Omega \varepsilon_{yy}^2 \, dA \\
\iint_\Omega \mathrm{Wxy} &= \frac{E}{2\left(1-\nu^2\right)} \iint_\Omega 2\left(1-\nu\right) \varepsilon_{xy}^2 \, dA
\end{aligned}
$$

- first order elements

```
EnergiesGrid             =   78.4722   62.8472  0   15.6250
EnergiesFEMIntegrate     =   67.4913   58.5938  0    8.8976
EnergiesFEMIntegrateGauss =  78.1250   62.5000  0   15.6250
```

- second order elements

```
EnergiesGrid              =  4.1759e+01   4.1759e+01   6.8171e-30   8.2941e-30
EnergiesFEMIntegrate      =  4.1667e+01   4.1667e+01   6.6145e-30   2.7413e-30
EnergiesFEMIntegrateGauss =  4.1667e+01   4.1667e+01   6.5378e-30   8.5890e-30
```

Observe that the results based on the integration with the Gauss points yields the same numbers as the exact formula.

5.14 Bending of an Euler beam

A plate of length $L = 1$, width $W = 1$ and height $H = 0.1$ is attached at the left edge and an upward force of $F = 100$ is applied on the right side. Use the material parameters $E = 100 \cdot 10^9$ and $\nu = 0$. Based on the Euler beam theory conclude

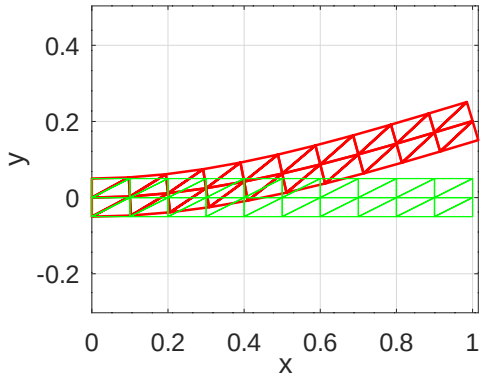


Figure 81: The original shape of the a beam and its (exaggerated) deformed shape, using two layers of elements

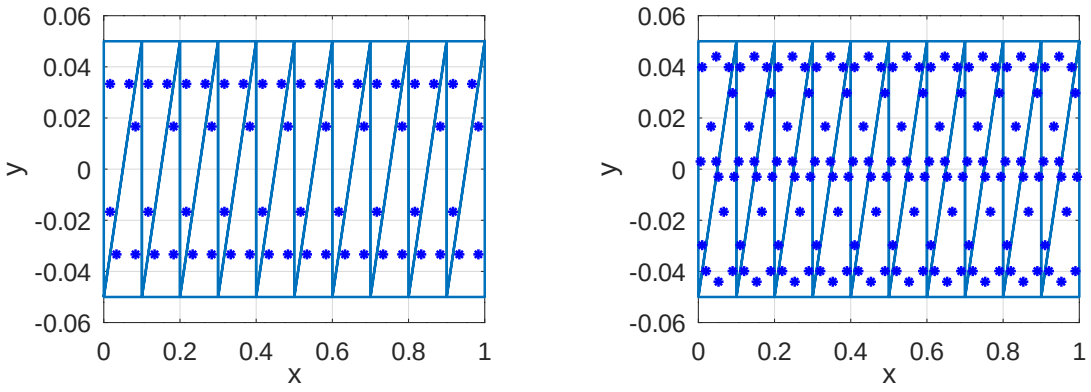
$$
\begin{aligned}
\frac{\partial^2}{\partial x^2}\, u_2(x,y) &= \frac{M}{E\,I} = \frac{F}{E\,I}\,(L-x) \quad , \quad \frac{\partial}{\partial x}\, u_2(x,y) = \frac{F}{E\,I}\,(L\,x - \frac{1}{2}\,x^2) \\
u_2(x,y) &= \frac{F}{E\,I}\,(\frac{L}{2}\,x^2 - \frac{1}{6}\,x^3) \\
u_1(x,y) &= -y\,\frac{\partial}{\partial x}\, u_2(x,y) = -\frac{F}{E\,I}\,(L\,x - \frac{1}{2}\,x^2)\,y \\
\varepsilon_{xx}(x,y) &= \frac{\partial\, u_1(x,y)}{\partial x} = -\frac{F}{E\,I}\,(L-x)\,y \quad , \quad \varepsilon_{yy}(x,y) = \frac{\partial\, u_2(x,y)}{\partial y} = 0 \\
\varepsilon_{xy}(x,y) &= \frac{1}{2}\,(\frac{\partial\, u_1}{\partial y} + \frac{\partial\, u_2}{\partial x}) = \frac{F}{E\,I}\,\left(-(L\,x - \frac{1}{2}\,x^2) + (\frac{L}{2}\,2\,x - \frac{1}{2}\,x^2)\right) = 0
\end{aligned}
$$

For the above parameters with the second moment $I = \frac{W\,H^3}{12}$ of the cross section obtain the following maximal values.

$$
\begin{aligned}
u_2(L,y) &= \frac{F}{3\,E\,I}\,L^3 = 4\,\frac{F}{E\,W\,H^3}\,L^3 = 4 \cdot 10^{-6} \\
u_1(L,-H/2) &= \frac{F}{4\,E\,I}\,L\,H = 3\,\frac{F}{E\,W\,H^2}\,L = 3 \cdot 10^{-7} \\
\varepsilon_{xx}(0,-H/2) &= \frac{F}{2\,E\,I}\,L\,H = 6\,\frac{F}{E\,W\,H^2}\,L = 6 \cdot 10^{-7}
\end{aligned}
$$

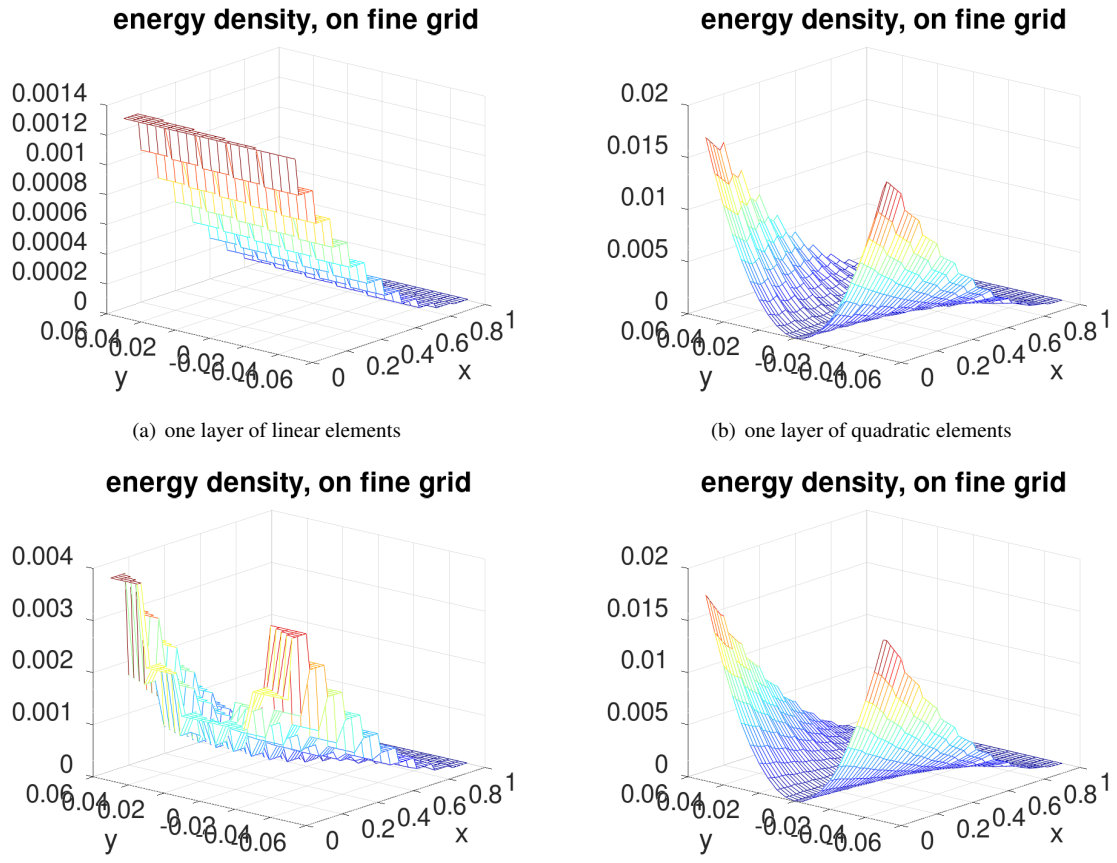
Use these results to verify the accuracy of the numerical approximations.

To examine the performance of the FEM algorithms use a rectangular mesh with NL sections along the horizontal x–axis and NH layers in the vertical y–direction. The code is using first, second or third order elements. In Figure 82 find the mesh and the corresponding integration points for meshes with NL=10 and just one layer, i.e. NH=1 . Observe that the figure uses different scaling, all triangles have height and width 0.1, which is usually recommended for good quality meshes. The code was run with NL=10 horizontal sections and NH=1 or 5 vertical sections. The elastic energy density W_{stress} is computed and displayed in Figure 83. Observe the piecewise constant energy density for linear elements, i.e. CST elements.



(a) linear elements (b) quadratic elements

Figure 82: Meshes for linear and quadratic elements with one layer, with the integration points



(a) one layer of linear elements (b) one layer of quadratic elements

(c) five layers of linear elements (d) five layers of quadratic elements

Figure 83: The elastic energy density of the bending beam with one or five layers

Multiple runs of the code `BendingBeam.m` lead to the results in Table 16. The values for the elastic energy are computed with the help of the strain values at the Gauss points. Observe that second and third order elements generate rather accurate results, even for a very coarse grid. With a coarse grid of linear elements the effect of shear locking is clearly visible. But even for a 80×8 grid the results are not very accurate.

element order	NL	NH	$\max\{u_2\}$	$\max\{\varepsilon_{xx}\}$	energy
Euler beam			$4 \cdot 10^{-6}$	$6 \cdot 10^{-7}$	$2 \cdot 10^{-4}$
first	80	8	$3.8159 \cdot 10^{-6}$	$5.7295 \cdot 10^{-7}$	$1.9079 \cdot 10^{-4}$
first	40	4	$3.3036 \cdot 10^{-6}$	$4.8900 \cdot 10^{-7}$	$1.6517 \cdot 10^{-4}$
first	20	2	$2.1525 \cdot 10^{-6}$	$3.1248 \cdot 10^{-7}$	$1.0762. \cdot 10^{-4}$
first	10	1	$0.9079 \cdot 10^{-6}$	$1.2718 \cdot 10^{-7}$	$0.4539 \cdot 10^{-4}$
second	80	8	$4.0243 \cdot 10^{-6}$	$6.1660 \cdot 10^{-7}$	$2.0120 \cdot 10^{-4}$
second	40	4	$4.0242 \cdot 10^{-6}$	$6.1101 \cdot 10^{-7}$	$2.0120 \cdot 10^{-4}$
second	20	2	$4.0235 \cdot 10^{-6}$	$6.0326 \cdot 10^{-7}$	$2.0117 \cdot 10^{-4}$
second	10	1	$4.0162 \cdot 10^{-6}$	$5.8832 \cdot 10^{-7}$	$2.0081 \cdot 10^{-4}$
third	80	8	$4.0244 \cdot 10^{-6}$	$6.2131 \cdot 10^{-7}$	$2.0122 \cdot 10^{-4}$
third	40	4	$4.0243 \cdot 10^{-6}$	$6.1708 \cdot 10^{-7}$	$2.0122 \cdot 10^{-4}$
third	20	2	$4.0243 \cdot 10^{-6}$	$6.1176 \cdot 10^{-7}$	$2.0121 \cdot 10^{-4}$
third	10	1	$4.0241 \cdot 10^{-6}$	$6.0592 \cdot 10^{-7}$	$2.0120 \cdot 10^{-4}$

Table 16: Different values for the deformation of a bending beam, depending on the size of the grid

BendingBeam.m

```
%% bending of beam by applied force
L = 1; H = 0.1; E = 100e9; nu = 0; Force = 100;

NL = 20;   %% number of elements along length L
NH =  NL/10;   %% number of elements along height H
Order = 2; %% order of elements, either 1, 2 or 3
FEMmesh = CreateMeshRect([0:L/NL:L],[-H/2:H/NH:+H/2],-22,-22,-11,-33);

figure(1); FEMtrimesh(FEMmesh);%% axis equal;
           hold on; plot(FEMmesh.GP(:,1),FEMmesh.GP(:,2),'b*'); hold off
           xlabel('x'); ylabel('y')
switch Order
  case 2
    FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
  case 3
    FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endswitch

[u1,u2] = PlaneStress(FEMmesh,E,nu,{0,0},{0,0},{0,Force/H});
figure(2); FEMtrimesh(FEMmesh,u1); xlabel('x'); ylabel('y'); zlabel('u1')
figure(3); FEMtrimesh(FEMmesh,u2); xlabel('x'); ylabel('y'); zlabel('u2')

FEMoctave_u2Max = max(u2);
EulerBeam = 4*Force*L^3/(E*H^3);
MaximalDisplacements = [EulerBeam, FEMoctave_u2Max]
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(FEMmesh,u1,u2);
figure(12); FEMtrimesh(FEMmesh,eps_xx); xlabel('x'); ylabel('y'); zlabel('eps_{xx}')
Results_Maxu1_Maxeps_xx = [max(abs(u1)), max(abs(eps_xx))]
W = 0.5*E/(1-nu^2)*(eps_xx.^2 + eps_yy.^2+2*nu*eps_xx.*eps_yy+2*(1-nu)*eps_xy.^2);

EnergyByForce = [Force*EulerBeam/2, Force*max(u2)/2]

figure(4);FEMtrimesh(FEMmesh,W); xlabel('x'); ylabel('y');
```

```
          title('energy density, on nodes'); view([-50,20])
figure(5);clf;FEMtricontour(FEMmesh,W); xlabel('x'); title('energy density')

%% integrate by evaluation at the Gauss points
W = EvaluateEnergyDensity(FEMmesh,eps_xx,eps_yy,eps_xy,E,nu);
EnergyIntegration = FEMIntegrate(FEMmesh,W)

[xx,yy] = meshgrid(linspace(0,L,101),linspace(-H/2,+H/2,51));
[u1i,eps_xxi,eps_xy1i] = FEMgriddata(FEMmesh,u1,xx,yy);
[u2i,eps_xy2i,eps_yyi] = FEMgriddata(FEMmesh,u2,xx,yy);
eps_xyi = (eps_xy1i+eps_xy2i)/2;

Wi = 0.5*E/(1-nu^2)*(eps_xxi.^2+eps_yyi.^2+2*nu*eps_xxi.*eps_yyi+2*(1-nu)*eps_xyi.^2);

figure(14); mesh(xx,yy,Wi);xlabel('x'); ylabel('y');
            title('energy density, on fine grid'); view([-50,20])

%% show deformed domain
factor = 1e5/2;
figure(100); ShowDeformation(FEMmesh,u1,u2,factor); xlabel('x'); ylabel('y'); axis equal
```

5.15 Eigenvalues and eigenmodes of a slender beam

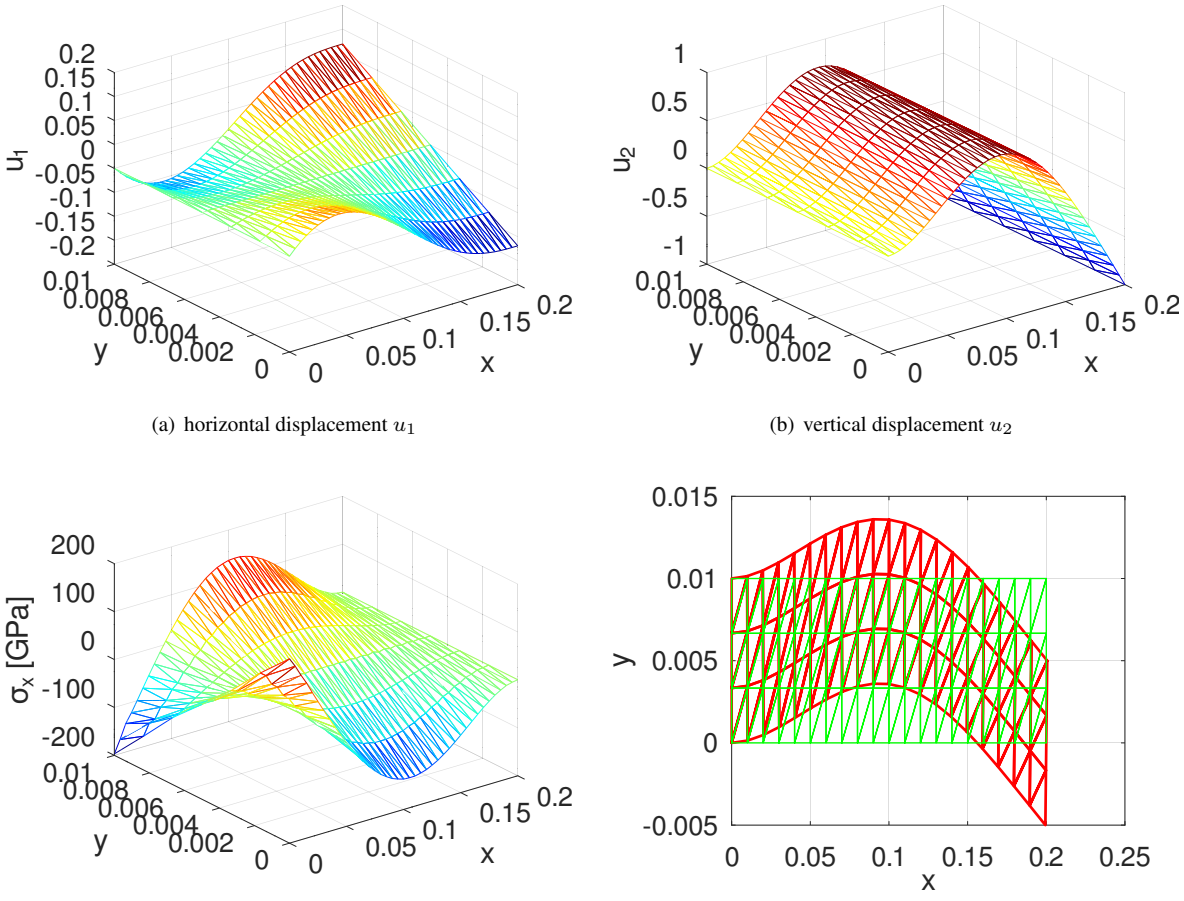
For the beam in Section 3.13.4 (page 56) more information can be obtained by `PlaneStressEig()`. The code in the file `EulerBeamModes.m` allows to examine multiple aspects of the eigenmodes of a bending beam. The code will generate figures similar to Figure 84. The Aluminum beam of length $L = 0.2$, height $H = 0.01$ and width $W = 0.01$ is clamped at the left edge at $x = 0$.

- Change the value of `Mode` to evaluate different modes, e.g. `Mode = 4`. The result might be surprising at first sight.
- Change the height H of the beam and observe the effect of the frequencies and maybe even the shape of the modes.
- The code allows to use linear, quadratic or cubic elements by selecting `MeshType`. Observe that the frequencies obtained by quadratic elements are smaller then the ones by linear elements, Cubic elements lead to the smallest frequencies.
- Modify the size of the mesh and observe that for finer meshes the frequencies are slightly lower.
- For a coarse mesh and linear elements the frequencies are considerably to high. This is caused by shear locking, see Section 5.13.

EulerBeamModes.m

```
clear *
L = 0.20; H = 0.01; W = 0.01; rho = 2.7e3;
E = 70e9; nu = 0.33; %% Aluminum
I2 = 1/12*H^3*W;
Mode = 2
Nx = 20; Ny = 3;
MeshType = 'linear';
MeshType = 'quadratic';
%MeshType = 'cubic';

f = @(z) 1+cos(z).*cosh(z);  %% clamped at x=0, free at x=L
% z = linspace(0,Mode*pi,100);
% figure(101); plot(z,f(z)); xlabel('z'); ylabel('f(z)')
z0 = fsolve(f,Mode*pi-pi/2);
freqEuler = z0^2*sqrt(E*I2/(rho*H*W))/(2*pi*L^2)
% p = k*lambda^0.25*L;
% C = (sin(p)-sinh(p))/(cos(p)+cosh(p));
% x = linspace(0,L);  x_p = k*lambda^0.25*x;
```



(a) horizontal displacement u_1

(b) vertical displacement u_2

(c) horizontal stress σ_x

(d) original and deformed beam

Figure 84: The second eigenmode of a bending beam

```
% y = cos(x_p)-cosh(x_p) + C*(sin(x_p)-sinh(x_p)); y = y/y(end);
% figure(102); plot(x,y); xlabel('x'); ylabel('height y(x)');
Mesh = CreateMeshRect(linspace(0,L,Nx+1),linspace(0,+H,Ny+1),-22,-22,-11,-22);
switch MeshType
 case 'quadratic'
   Mesh = MeshUpgrade(Mesh,'quadratic');
 case 'cubic'
   Mesh = MeshUpgrade(Mesh,'cubic');
endswitch
[la,u1,u2] = PlaneStressEig(Mesh,E,nu,rho,max(4,Mode));
freqFEM = sqrt(la')/(2*pi)
u1_disp = u1(:,Mode); u2_disp = u2(:,Mode);
MaxDisp = max(max(abs(u1_disp)),max(abs(u2_disp)));
u1_disp = u1_disp/MaxDisp; u2_disp = u2_disp/MaxDisp;
figure(1); FEMtrimesh(Mesh,u1_disp); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(2); FEMtrimesh(Mesh,u2_disp); xlabel('x'); ylabel('y'); zlabel('u_2')
[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1_disp,u2_disp,E,nu);
figure(11); FEMtrimesh(Mesh,sigma_x*1e-9);
            xlabel('x'); ylabel('y'); zlabel('\sigma_x [GPa]')
figure(12); FEMtrimesh(Mesh,sigma_y*1e-9);
            xlabel('x'); ylabel('y'); zlabel('\sigma_y [GPa]')
figure(13); FEMtrimesh(Mesh,tau_xy*1e-9);
            xlabel('x'); ylabel('y'); zlabel('\tau_{xy} [GPa]')

figure(20);clf; factor = 0.005;
trimesh(Mesh.elem,Mesh.nodes(:,1)+factor*u1_disp,Mesh.nodes(:,2)+factor*u2_disp,...
'color','red','linewidth',2);
hold on ;
trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),'color','green','linewidth',1);
xlabel('x'); ylabel('y'); %%xlim([0,L*1.1])
-->
Mode = 2
freqEuler =  1288.7
freqFEM   =   205.66   1274.56   3508.03   6372.09
```

5.16 Stress concentration on a plate with a slit

As an academic example examine a plate with dimensions 1×1 and vertical slit in the middle. Apply a fixed displacement of ± 0.1 on the left and right edge. Without the slit expect an approximate horizontal displacement of $u_x(x,y) \approx 0.1\,x$ and a resulting horizontal strain $\varepsilon_{xx} \approx 0.1$ and vertical strain of $\varepsilon_{yy}(x,y) \approx -\nu\,0.1 = -0.025$ in the middle section. The slit will change the situation drastically. For a vertical slit at $-W < y < W = 0.33$ and $x = 0$ find the resulting displacments in Figure 85.

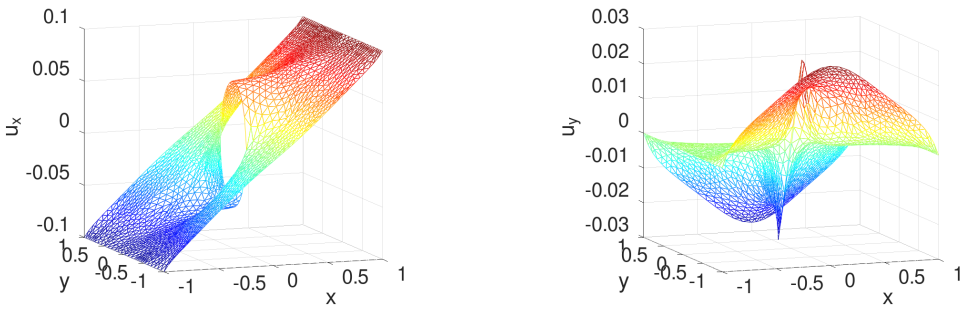


Figure 85: The displacements u_x and u_y for a plate with a slit

Since the elastic behaviour at the endpoints of the slit is rather special, examine the sitiation there more carefully by using a locally finer mesh, visible in Figure 86. The mesh at the upper end of the slit is refined, but the mesh at

the lower end of the slit is left coarse. The mesh visible in Figure 86 is upgraded to a mesh with cubic (or quadratic) elements by using `MeshUpgrade()`.

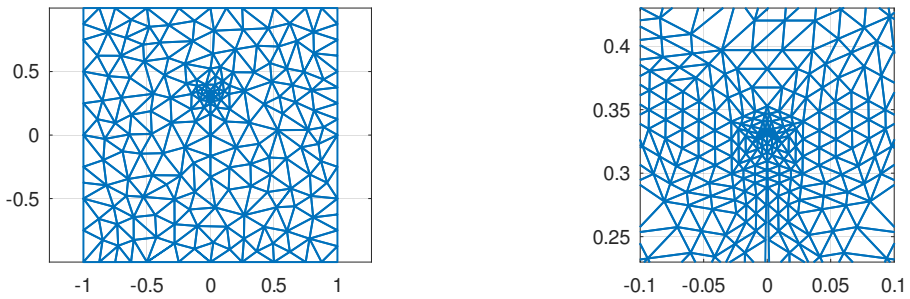


Figure 86: The mesh of a plate with a slit, the full domain and the area with a finer mesh. On the right the narrow slit is visible in the center of the lower part.

Slit.m

```
Hole.name = "Hole";  W = 0.33;
Hole.border = [0,-W,-22;-5e-3,0,-22;0,+W,-22;+5e-3,0,-22]; Hole.point = [0,0];
Segment.name= "Segment"; Segment.border = [-5e-3,W+5e-3,0;+5e-3,W+5e-3,0];
xy = [-1,-1,-22; 1,-1,-11;+1,+1,-22;-1,+1,-11];
Mesh = CreateMeshTriangle("Slit",xy,2e-2,Hole,Segment);
figure(1); FEMtrimesh(Mesh); axis equal
Mesh = MeshUpgrade(Mesh,'cubic');
figure(2); FEMtrimesh(Mesh); xlim([-0.1, +0.1]); ylim([W-0.1,W+0.1]); axis equal
```

The use `PlaneStress()` to determine the displacements and resulting strains and stresses. In Figure 87 find the graph of the von Mises stress.

- On the left in Figure 87 a strong peak of the von Mises stress at the upper end of the slit is visible. This is a clear sign of stress concentration.
- On the left in Figure 87 a weaker peak of the von Mises stress at the lower end of the slit is visible. The configuration is symmetric, thus the peaks should have the same height. Thus the difference of the height of the peaks indicates that the result strongly depends on the mesh.
- The contour plot on the right in Figure 87 illustrates that away from the peaks the expected symmetry is visible.
- The height of the peaks change for finer or coarser meshes and also depend on the type of elements used, i.e. linear, quadratic or cubic.

Standard FEM codes can not correctly analyze the propagation of cracks. Advanced models and FEM codes have to be used.

This illustrates that FEM results in situations with stress concentration can be very delicate for FEM computations. This setup is an extreme case of stress concentration and one has to be very careful to not draw wrong conclusions from FEM computations.

Find other examples of stress concentration in Sections 9.42, 9.45 and 9.48

Slit.m

```
Displacement = @(xy) 0.1*xy(:,1);
E = 1; nu = 0.25;  %% solve the plane stress problem
[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{Displacement,0},{0,0});

figure(11); FEMtrimesh(Mesh,u1);
            xlabel('x'); ylabel('y'); zlabel('u_x'); view([-20,10])
figure(12); FEMtrimesh(Mesh,u2);
            xlabel('x'); ylabel('y'); zlabel('u_y'); view([-20,10])
```

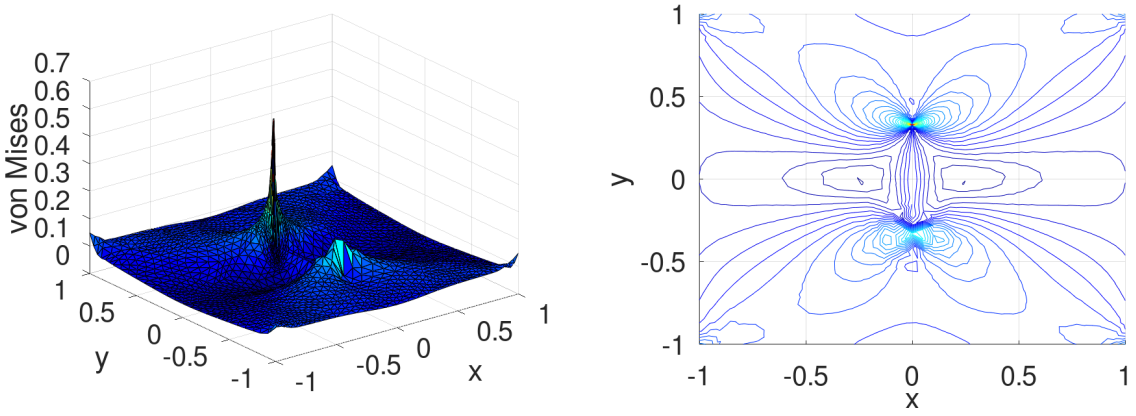


Figure 87: The von Mises stress for a plate with a slit

```
[eps_xx,eps_yy,eps_xy]   = EvaluateStrain(Mesh,u1,u2);
figure(13); FEMtrimesh(Mesh,eps_xx);
            xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(14); FEMtrimesh(Mesh,eps_yy);
            xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')

[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu);
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);
figure(21); FEMtrisurf(Mesh,vonMises);
            xlabel('x'); ylabel('y'); zlabel('von Mises')
figure(22); FEMtricontour(Mesh,vonMises,51); xlabel('x'); ylabel('y');
figure(23); FEMtrimesh(Mesh,sigma_x);
            xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(24); FEMtrimesh(Mesh,sigma_y);
            xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(25); FEMtrimesh(Mesh,tau_xy);
            xlabel('x'); ylabel('y'); zlabel('\tau_{xy}')
Max_vonMises = max(vonMises)
```

5.17 Adding missing constraints

5.17.1 Adding a constraint for a steady state heat problem

Examine the boundary value problem

$$\begin{aligned} -\Delta u(x,y) &= \sin(\pi\, x) && \text{for } -1 \le x,y \le +1 \\ \tfrac{\partial}{\partial n}\, u(x,y) &= 0 && \text{on the boundary} \end{aligned}$$

This BVP with Neumann boundary conditions only has solutions if the integral of the RHS over the domain vanishes. This is the case for the given RHS. Use $\iint_\Omega \sin(\pi\, x)\ dA = 0$ to conclude that this problem has solutions, but infinitely many. The solutions differ by a constant. To obtain a unique solution the value at one point can be selected, e.g. at $(x,y) = (0,0)$ select $u(0,0) = 7$. With this additional constraint the solution is unique. In the code `AdditionalConstraint.m` below the command `MeshAddConstraint()` is used to

- select the node closest to the origin by `Position = [0,0]`
- add the constraint $u = 7$ at this node by using the function $g_D(x,y) = 7$

The two numbers generated by the code illustrate that

- the first solution u_1 without the constraint has a huge mean value. The condition number of the matrix generated by the FEM code is huge. But *Octave* still shows a solution. It is the users responsibility to realize that the solution is not the desired one.
- the average value of the solution u_2 at the nodes a close to 7 and Figure 88 confirms the result. Replacing the average values at the nodes by a numerical integration leads to a result even closer to 7.

AdditionalConstraintHeat.m

```
Mesh = CreateMeshTriangle('square',[-1,-1,-2;1,-1,-2;1 1 -2; -1 1 -2],0.05);
figure(1); FEMtrimesh(Mesh);
Mesh = MeshUpgrade(Mesh,'quadratic');
function res = f(xy)
  res = sin(pi*xy(:,1));
endfunction
u1 = BVP2D(Mesh,1,0,0,0,'f',7,0,0);
MeanSolution1 = mean(u1)

Mesh = MeshAddConstraint(Mesh,[0,0],-1);
u2 = BVP2D(Mesh,1,0,0,0,'f',7,0,0);
figure(1); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u')
MeanSolution2 = mean(u2)
MeanSolution2Intgeration = FEMIntegrate(Mesh,u2)/4
-->
MeanSolution1 = -6.6466e+07
MeanSolution2 =  6.9988
MeanSolution2Intgeration = 7.0000
```

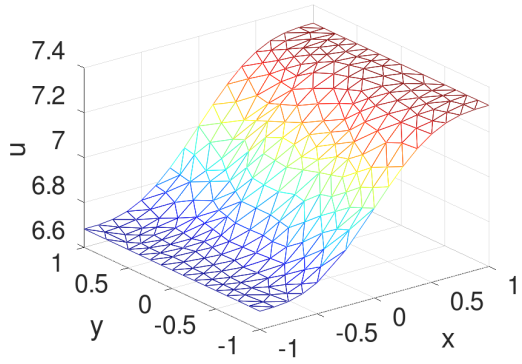


Figure 88: The solution of the boundary value problem with an additional constraint

5.17.2 Adding constraints for an elasticity problem

A similar setup can be considered for a plane stress elasticity problem. On a square $-1 \le x, y \le +1$ a set of boundary forces is applied.

$$\begin{aligned} f_x(+1,y) &= +\cos(y) \quad \text{for } -1 \le y \le +1 \\ f_x(-1,y) &= -\cos(y) \quad \text{for } -1 \le y \le +1 \\ f_y(x,+1) &= +\cos(x) \quad \text{for } -1 \le x \le +1 \\ f_y(x,-1) &= -\cos(x) \quad \text{for } -1 \le x \le +1 \end{aligned}$$

The net force on the plate is zero, i.e. there is a solution. But the plate is free to move and rotate. With `PlaneStress()` a solution can be determined, leading to Figure 89(a) for the total displacement $u = \sqrt{u_1^2 + u_2^2}$. It is clearly visible that the origin $(0,0)$ is displaced. Modifying the setup slightly[19] can lead to very different solutions.

Adding two constraints

[19] Add a tiny contribution of 10^{-10} to one of the bounadry forces and observe the results.

- zero displacement in x and y–direction at the origin by using `Pos = [0.0]; Mode = [-1,-1];`
- and zero dispacement in y–direction at the point $(1,0)$ by using `Pos = [1,0]; Mode = [-2,-1];`
- each followed by a call of `Mesh = MeshAddConstraint(Mesh,Pos,Mode);`

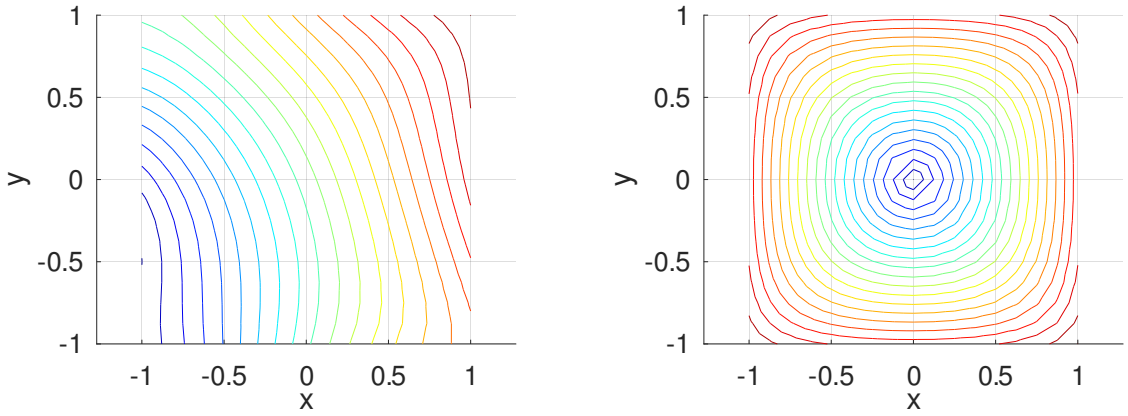
prevents the plate from moving and rotating freely. A call of `PlaneStress()` then leads to Figure 89(b). It is clearly visible that the origin $(0,0)$ is not moved. Slight changes in the setup will lead to slightly different solutions.

AdditionalConstraintPlate.m

```
Mesh = CreateMeshRect(linspace(-1,1,9),linspace(-1,1,9),-23, -23, -32, -32);
Mesh = MeshUpgrade(Mesh,'quadratic');
function res = fx(xy)
 res = 1*xy(:,1).*cos(xy(:,2));
endfunction
function res = fy(xy)
 res = 1*xy(:,2).*cos(xy(:,1));
endfunction
[u1,u2] = PlaneStress(Mesh,1,0,{0,0},{0,0},{'fx','fy'});
figure(2); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(3); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')
figure(4); FEMtrimesh(Mesh,sqrt(u1.^2+u2.^2)); xlabel('x'); ylabel('y'); zlabel('|u|')
figure(5); clf; FEMtricontour(Mesh,sqrt(u1.^2+u2.^2)); xlabel('x'); ylabel('y'); axis equal

Pos = [0.0]; Mode = [-1,-1];              %% fix the origin
Mesh = MeshAddConstraint(Mesh,Pos,Mode);  %% remove rotations
Pos = [1,0]; Mode = [-2,-1];
Mesh = MeshAddConstraint(Mesh,Pos,Mode);
[u1m,u2m] = PlaneStress(Mesh,1,0,{0,0},{0,0},{'fx','fy'});

figure(12); FEMtrimesh(Mesh,u1m); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(13); FEMtrimesh(Mesh,u2m); xlabel('x'); ylabel('y'); zlabel('u_2')
figure(14); FEMtrimesh(Mesh,sqrt(u1m.^2+u2m.^2)); xlabel('x'); ylabel('y'); zlabel('|u|')
figure(15); clf; FEMtricontour(Mesh,sqrt(u1m.^2+u2m.^2)); xlabel('x');
ylabel('y'); axis equal
```



(a) without contraints (b) with contraints

Figure 89: The contours for the displacement $u = \sqrt{u_1^2 + u_2^2}$, without and with constraints

5.18 Missing boundary constraints and null spaces

Examine a domain $\Omega \subset \mathbb{R}^2$ and minimize the elastic energy given by equation (29) with $\vec{f} = \vec{g}_N = \vec{0}$

$$U(\vec{u}) = \iint_\Omega \frac{1}{2} \frac{E}{(1-\nu^2)} \left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dA \,,$$

i.e. no external forces and all of the boundary is free to move. Since the strains depend on derivatives of the displacement, the energy $U(\vec{u})$ will not change for constant displacement vectors $\vec{u}$. In addition the domain can be rotated without deformation, i.e. without adding elastic energy. This leads to a three dimensional subspace on which the elastic energy vanishes. No reliable FEM algorithm will be able to solve the corresponding problem of minimizing the energy, since there is no unique minimum. To obtain a solution constraints have to be introduced, preventing the solid from moving in the x and y direction and preventing rotations.

As a consequence the global stiffness matrix $\mathbf{A}$ should have a three dimensional null space, describing constant displacements and rotations. Thus expect three very small eigenvalues. Constant displacement vectors $\vec{u}$ in x or y direction and rotations should satisfy

$$\mathbf{A}\,\vec{u} = \vec{0} \quad \text{and} \quad \frac{1}{2}\langle \vec{u}, \mathbf{A}\vec{u} \rangle = 0 \,,$$

where the second expression corresponds to the elastic energy. These expressions are examined in the code below. The code verifies this on a trapezoidal domain with corners at $(0,0)$, $(1,0)$, $(1,2)$ and $(0,1)$.

- There are three eigenvalues very close to zero. Due to the finite accuracy of the arithmetic on the CPU the values are not exactly zero[20]. The fourth eigenvalue is considerably larger. This confirms the three dimensional null space of the matrix $\mathbf{A}$.
- The vector `shift_x` implements a translation of the solid in x direction. Since `A*shift_x` is approximately 0 the vector is in the null space of $\mathbf{A}$.
- With the vector `shift_y` the behavior in y direction is examined.
- The displacement vector `rot_vec` examines a rotation of the solid, verifying that the energy is not increased by this rotation.
- The null space of the matrix $\mathbf{A}$ is spanned by the above three vectors.
- With the vector `rand_vec` examine an arbitrary displacement and observe that this vector is not in the null space and the energy is clearly increased.

TestNullSpace.m

```
Mesh = CreateMeshTriangle('test',[1 0 -22;2 0 -22;2 2 -22; 1 1 -22],0.01);

E = 1e9; nu = 0.3; f = {0,0}; gD = {0,0}; gN = {0,0};  %% set the parameters
if 0  %% plane stress
  [A,g] = PStressEquationM(Mesh,E,nu,f,gD,gD);         %% determine matrix A
else  %% axially symmetric
  [A,g] = AxiStressEquationM(Mesh,E,nu,f,gD,gD);       %% determine matrix A
endif
A = (A+A')/2;  %% assure that matrix is symmetric, it should be, but rounding errors
EigenValues = eigs(A,6,'sa')           %% find the smallest eigenvalues

n = size(A,1)/2;
shift_x = [ones(n,1);zeros(n,1)];      %% constant shift in x direction
shift_x = shift_x/norm(shift_x);
Norm_Shift_x  = [norm(A*shift_x),shift_x'*A*shift_x/2]

shift_y = [zeros(n,1);ones(n,1)];      %% constant shift in y direction
shift_y = shift_y/norm(shift_y);
Norm_Shift_y  = [norm(A*shift_y),shift_y'*A*shift_y/2]
```

[20] The operation $\mathbf{A} = (\mathbf{A} + \mathbf{A}')/2$ assures that the matrix is symmetric, to get around a problem in the implementation of `eigs()` in *Octave*.

```
x = Mesh.nodes(:,1); y = Mesh.nodes(:,2); %% at point [x,y] add displacement [-y,x]
rot_vec = [-y;x];                         %% a rotation
rot_vec = rot_vec/norm(rot_vec);
Norm_Rotation = [norm(A*rot_vec),rot_vec'*A*rot_vec/2]

rand_vec = [x.*y;y];    %% an arbitrary displacement vector
rand_vec = rand_vec/norm(rand_vec);
Norm_Random = [norm(A*rand_vec),rand_vec'*A*rand_vec/2]
-->
EigenValues = -5.9793e-07 -3.8610e-07 4.6269e-07 8.6494e+06 2.7384e+07 3.1225e+07

Norm_Shift_x  =   3.1718e-07   7.8932e-10
Norm_Shift_y  =   2.8492e-07  -1.9276e-08
Norm_Rotation =   3.3145e-07   2.2532e-09
Norm_Random   =   5.1146e+07   5.3317e+06
```

Another option is the (newer) command `PlaneStressEig()`, which shows that the first 3 eigenvalues are (approximately) zero and the other eigenvalues are clearly positive.

```
lambda = PlaneStressEig(Mesh,W,nu,1,6)'
-->
lambda =  -1.2637e-17   8.7877e-18   5.2417e-17   1.2456e-02   3.2735e-02   3.7392e-02
```

The zero eigenvalues could be removed by using additional constraints, as shown in the previous Section 5.17.2. The example of vibrations of a ring in Section 9.51 illustrates the same effect.

The situation changes for axially symmetric problems, i.e. the domain in the xz–plane is rotated about the z–axis to obtain the object in the space $\mathbb{R}^3$. Instead of `PStressEquationM()` use `AxiStressEquationM()` to generate the stiffness matrix $\mathbf{A}$. In the above code change the switch in the third line to obtain the results below.

- For axially symmetric problems moving the object up (in z–direction) does not lead to a deformation, thus there is at least a one-dimensional nullspace.
- Moving the intersection of the object with the plane $y = 0$ in radial direction (in the x–direction) does deform the 3D object and thus increases the elastic energy.
- Rotating the intersection of the object with the plane $y = 0$ does deform the 3D object and thus increases the elastic energy.

As a consequence there is only one eigenvalue (close to) zero, which is confirmed by the results below.

```
EigenValues = 1.7398e-06 4.0145e+06 7.3985e+06 1.8776e+07 5.0384e+07 5.4062e+07

Norm_Shift_x  =   4.0504e+07   4.8423e+06
Norm_Shift_y  =   6.0413e-07  -4.3348e-09
Norm_Rotation =   2.3078e+07   1.0892e+06
Norm_Random   =   1.5364e+08   1.8131e+07
```

6 The Mathematics of the Algorithms for 2D FEM

In this section the mathematical background for the FEM method applied to the problems in Section 2 is explained. Most of the theory is used to solve the second order elliptic boundary value problem (1). The explanations are certainly not complete, but should provide enough information to ease the understanding of the code. For in-depth coverage consult one of the many books on FEM and/or numerical analysis. The starting point for this presentation are the lecture notes [Stah08]. Find a list of books on FEM in [Stah08, §0].

The organization of this section is as follows:

6.1 The definition of classical and weak solutions of boundary value problems is given and the connection to calculus of variations is shown.

6.2 The most often used triangular elements are presented.

6.3 The method of interpolation on triangles and Gauss integration is explained.

6.4 Element stiffness matrices for triangular elements of order 1 are carefully derived. The construction of the global stiffness matrix is explained. The integration of the different contributions is performed.

6.5 Element stiffness matrices for triangular elements of order 2 are carefully derived. The integration of the different contributions is performed.

6.6 Element stiffness matrices for triangular elements of order 3 are carefully derived. The integration of the different contributions is performed.

6.7 Element stiffness matrices for triangular Hermite elements of are carefully derived. The integration of the different contributions is performed.

6.8 The theoretical convergence result is shown.

6.9 An algorithm to solve some nonlinear boundary value problems in domains $\Omega \subset \mathbb{R}^2$ is explained.

6.10 The algorithms to solve dynamic initial boundary value problems are presented.

- In Section 6.10.1 dynamic heat equations are examined, using the Crank–Nicolson approach.
- In Section 6.10.2 using eigenvalues to solve dynamic heat equations is explained.
- In Section 6.10.3 semilinear heat conduction problems are examined.
- In Section 6.10.4 dynamic wave equations are examined, using an implicit approximation.
- In Section 6.10.5 using eigenvalues to solve dynamic wave equations is explained.

6.11 A few remarks on using the inverse power iteration to determine eigenvalues.

6.1 Classical solutions and weak solutions

A function $u = u(x, y)$ is called a **classical solution** of the BVP (1) iff it is twice differentiable and

$$\begin{aligned} -\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y) \in \Omega \\ u &= g_1 && \text{for} \quad (x,y) \in \Gamma_1 \quad . \\ \vec{n} \cdot (a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y) \in \Gamma_2 \end{aligned}$$

Multiply this equation with a smooth function ϕ, vanishing on Γ_1, and integrate over the domain Ω. Then use integration by parts to arrive at

$$\begin{aligned} 0 &= -\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\,u - f \\ 0 &= \iint\limits_\Omega \Big(-\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\,u - f \Big)\ \phi\ dA \\ &= \iint\limits_\Omega (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\ \phi\ dA - \int_\Gamma \phi\ \Big(a\,\nabla u - u\,\vec{b} \Big) \cdot \vec{n}\ ds \\ 0 &= \iint\limits_\Omega (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\ \phi\ dA - \int_{\Gamma_2} \phi\ (g_2 + g_3\,u)\ ds\,. \end{aligned} \tag{46}$$

If a function u satisfies (46) it is called a **weak solution** of the above BVP. If there is no convection term ($\vec{b} = \vec{0}$) and some sign conditions for a and b_0 are satisfied, the above is equivalent to minimizing the functional

$$F(u) = \iint_\Omega \frac{1}{2}\, a\, (\nabla u)^2 + \frac{1}{2}\, b_0\, u^2 + f \cdot u \; dA - \int_{\Gamma_2} g_2\, u + \frac{1}{2}\, g_3\, u^2 \; ds$$

among all functions u satisfying the boundary condition $u = g_1$ on Γ_1. Figure 90 shows connections between classical solutions, weak solutions and the resulting system of (linear) equations for the finite element approach. The left branch in Figure 90 illustrates the usage of minimization and calculus of variations in the context of FEM algorithms. This approach is often called Ritz method, named after the Swiss mathematician Walter Ritz (1878–1909). The right branch in Figure 90 shows a Galerkin method, named after the Russian mathematician Boris Galerkin (1871–1945).

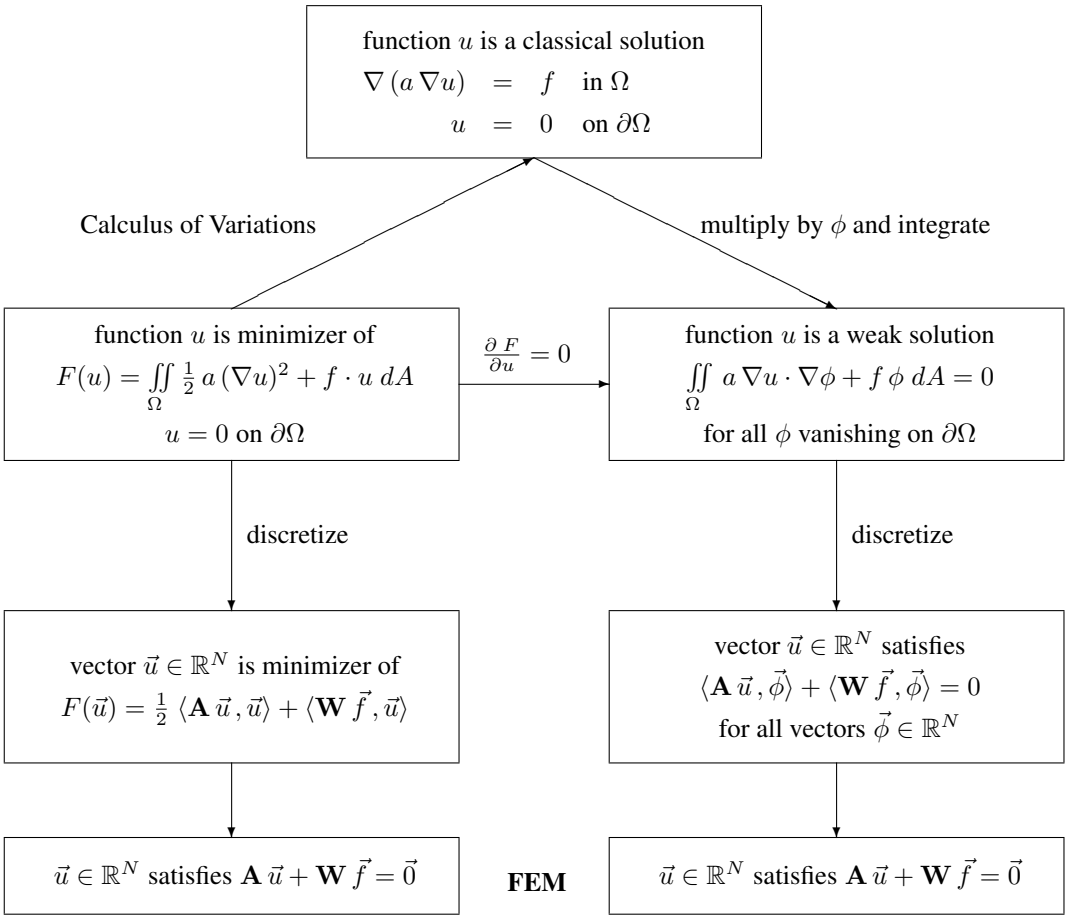
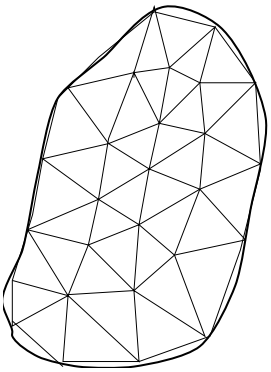


Figure 90: Classical and weak solutions, minimizers and FEM

In the above equation integrals over the domain $\Omega \subset \mathbb{R}^2$ have to be computed. To discretize this process use a triangularization of the domain, based on grid points $(x_i, y_i) \in \Omega$, $1 \leq i \leq n$. On each triangle T_k replace the function u by polynomials of degree 1 (or 2, or 3). These polynomials are completely determined by their values at the three corners of the triangle (or corners and some points on the edges). Integrals over the full domain Ω are split up into integrals over each triangle and then a summation over all triangles, i.e.



$$\iint_{\Omega} \ldots dA = \sum_k \iint_{T_k} \ldots dA \,.$$

The gradients of u and ϕ are replaced by the gradients of the piecewise polynomials. Each contribution has to be written in the form

$$\iint_{T_k} \ldots dA = \langle \mathbf{A}_k \vec{u}_k \,,\, \vec{\phi}_k \rangle + \langle \mathbf{W}_k \vec{f}_k \,,\, \vec{\phi}_k \rangle \,,$$

where $\mathbf{A}_k$ is the **element stiffness matrix**.

The above integral will be rewritten as sum of the above integrations of the triangles, leading to the condition

$$\langle \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\,,\, \vec{\phi} \rangle = 0 \quad \text{for all} \quad \vec{\phi} \in \mathbb{R}^N \,.$$

This condition is satisfied if $\vec{u}$ solves the linear system $\mathbf{A}\vec{u} = -\mathbf{W}\vec{f}$. The matrix $\mathbf{A}$ is called **global stiffness matrix**. It is this system of linear equations that will be solved to obtain an approximate solution of the boundary value problem (1).

6.2 A few triangular elements

There are different methods to construct finite elements on triangles. In Figure 91 find a graphical representation of a few commonly used triangular elements.

- A solid dot at a position indicates that the value at this point is used as a DOF (Degree Of Freedom).
- A circle around a solid dot at a position indicates that the values of the first order partial derivatives are used as DOFs, e.g. in the Hermite elements.
- A double circle around a solid dot at a position indicates that the values, first and second order partial derivatives are used as DOFs, e.g. in the Argyris elements.
- A short line at a position indicates that the value of the normal derivative at this point is used as a DOF, e.g. in the Morley and Argyris elements.

An element is called

- C^0 conforming if the resulting solutions are continuous across element boundaries.
- C^1 conforming if the resulting solutions and the first order derivatives are continuous across element boundaries.

The codes in FEMoctave only use linear, quadratic and cubic elements.

	linear	quadratic	Morley	cubic	Hermite	Argyris
degrees of freedom, DOF	3	6	6	10	10	21
polynomial basis	$\mathbb{P}_1$	$\mathbb{P}_2$	$\mathbb{P}_2$	$\mathbb{P}_3$	$\mathbb{P}_3$	$\mathbb{P}_5$
C^0 conforming	yes	yes	no	yes	yes	yes
C^1 conforming	no	no	quasi	no	quasi	yes

Table 17: A few properties of triangular elements

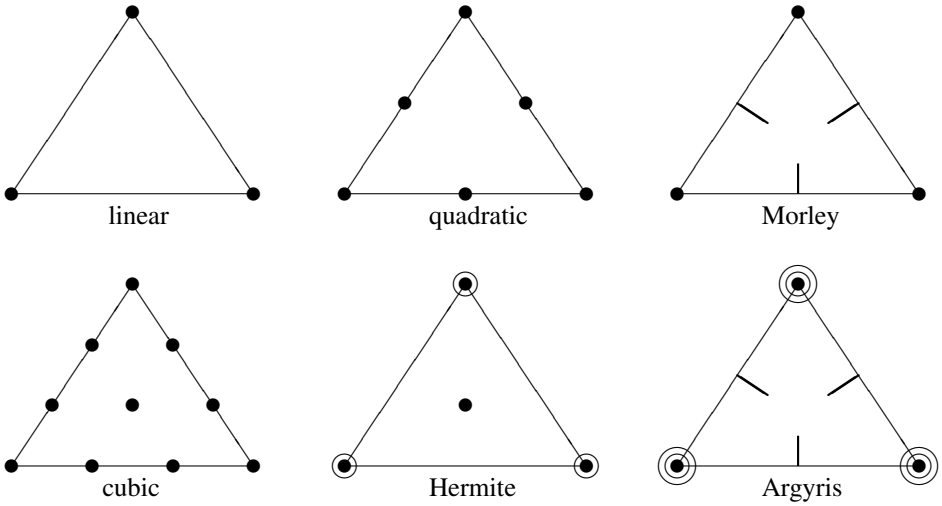


Figure 91: A few triangular elements

6.3 Transformation of coordinates, interpolation and Gauss integration

From the above it is obvious that integration over general triangles is important for the development of FEM algorithms. It turns out to be convenient to find integration methods for a standard triangle and then consider the general triangle by appropriate coordinate transformations.

6.3.1 Transformation of coordinates and integration over a general triangle

All of the necesssary integrals for the FEM method are integrals over general triangles E. These can be written as images of a standard triangle in a (ξ, ν)–plane, according to Figure 92. The transformation is given by

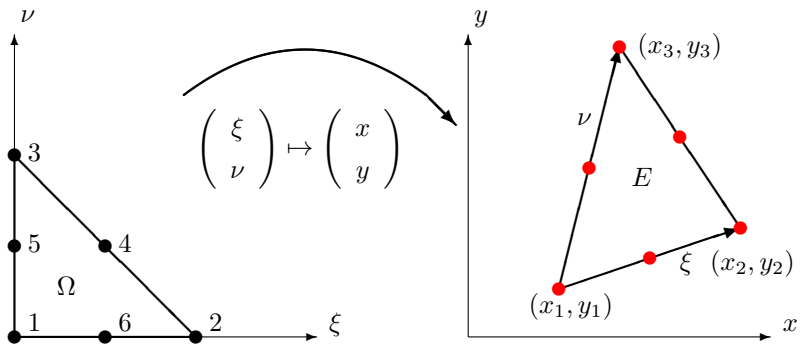


Figure 92: Transformation of the standard triangle Ω to a general triangle E

$$\begin{aligned}\begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \nu \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \\ &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix}\end{aligned}$$

with the transformation matrix

$$\mathbf{T} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}.$$

By using $0 < \xi, \nu < 1$ with $\xi + \nu < 1$ the standard triangle Ω is mapped onto the general triangle $E \subset \mathbb{R}^2$. If the coordinates (x, y) are given find the values of (ξ, ν) with the help of

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} .$$

If a function $f(x, y)$ is to be integrated over the general triangle E use the transformation rule

$$\iint\limits_E f \, dA = \iint\limits_\Omega f\left(\vec{x}\,(\xi, \nu)\right) \left| \det\left(\frac{\partial\,(x, y)}{\partial\,(\xi, \nu)} \right) \right| \, d\xi \, d\nu = |\det(\mathbf{T})| \int_0^1 \left(\int_0^\nu f\left(\vec{x}\,(\xi, \nu)\right) \, d\xi \right) d\nu \, . \tag{47}$$

The Jaccobi determinant is given by

$$\left| \det\left(\frac{\partial\,(x, y)}{\partial\,(u, v)} \right) \right| = |\det(\mathbf{T})| = |(x_2 - x_1)\,(y_3 - y_1) - (x_3 - x_1)\,(y_2 - y_1)|$$

If the orientation of the triangle is positive, then $\det(\mathbf{T})$ will be positive. Since the area of the standard triangle Ω equals $1/2$ find

$$\text{area of } E = \frac{1}{2} \, |\det \mathbf{T}| \, .$$

For an efficient numerical integration over the standard triangle Ω choose integration points $\vec{g}_j \in \Omega$ and corresponding weights w_j for $j = 1, 2, \ldots, m$ and then work with the values of the function at those points, i.e. seek an approximation of the integral of the form

$$\iint\limits_\Omega f(\vec{\xi}) \, dA \approx \sum_{j=1}^{m} w_j \, f(\vec{g}_j) \, . \tag{48}$$

The integration points $\vec{g}_j$ and weights w_j have to be chosen, such that the approximation error is as small as possible. Required are three essential conditions for the integration method:

- If a sample point is used in a Gauss integration, then all other points obtainable by permuting the three corners of the triangle must appear and with identical weight.
- All sample points $\vec{g}_j$ must be inside the triangle, or on the triangle boundary.
- All weights w_j must be positive.

6.3.2 Gauss integration on the standard triangle with 3 Gauss points

In Figure 93 consider the three points at $\vec{g}_1 = \frac{1}{2}\,(\lambda, \lambda)$, $\vec{g}_2 = (1 - \lambda, \lambda/2)$ and $\vec{g}_1 = (\lambda/2, 1 - \lambda)$. Find optimal values for the parameters λ and w such that polynomials of degree as high as possible are integrated exactly by

$$\iint\limits_\Delta f \, dA \approx w \, \left(f(\vec{g}_1) + f(\vec{g}_2) + f(\vec{g}_3) \right) \, .$$

To find the optimal values determine a solution of a nonlinear system of 2 equations for the unknowns λ and w. Require that ξ^k for $0 \le k \le 2$ be integrated exactly. This leads to the solution $\lambda = 1/3$ and the weight $w = 1/6$. This approximate integration yields the exact results for polynomials f up to degree 2 . Thus for a single triangle with diameter h, i.e. an area of the order h^2, the integration error for smooth functions is of the order $h^3 \cdot h^2 = h^5$. When dividing a large domain in sub-triangles of size h this leads to a total integration error of the order h^3.

The Gauss points and weights are given by

$$\mathbf{G} = \begin{bmatrix} 1/6 & 1/6 \\ 2/3 & 1/6 \\ 1/2 & 2/3 \end{bmatrix} \quad \text{and} \quad w = \frac{1}{6} \, .$$

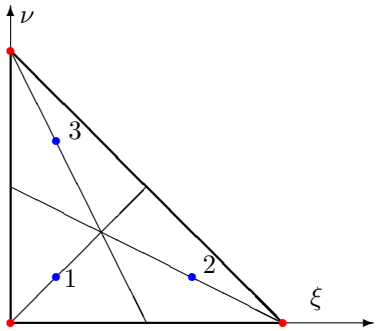


Figure 93: Gauss integration of order 2 on the standard triangle, using 3 integration points

For a general triangle the three Gauss points are located at

$$\begin{aligned}\mathbf{X}_G &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \frac{1}{6}\begin{bmatrix} (x_2-x_1)+(x_3-x_1) & 4\,(x_2-x_1)+(x_3-x_1) & 3\,(x_2-x_1)+4\,(x_3-x_1) \\ (y_2-y_1)+(y_3-y_1) & 4\,(y_2-y_1)+(y_3-y_1) & 3\,(y_2-y_1)+4\,(y_3-y_1) \end{bmatrix} \\ &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2-x_1 & x_3-x_1 \\ y_2-y_1 & y_3-y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T}\cdot\mathbf{G}^T\ .\end{aligned}$$

This integration scheme will be used for linear elements[21].

6.3.3 Gauss integration on the standard triangle with 7 Gauss points

As a second method use the points $g_1 = (\lambda_1, \lambda_1)$ and $g_4 = (\lambda_2, \lambda_2)$ along the diagonal $\xi = \nu$. Similarly use two more points along each connecting straight line from a corner of the triangle to the midpoint of the opposite edge. This leads to a total of six integration points where groups of three have the same weight. Finally add the midpoint with weight w_3. The integral is approximated by

$$\iint_\Delta f\ dA \approx \text{area}(\Delta)\ \left(w_1\,(f(\vec{g}_1)+f(\vec{g}_2)+f(\vec{g}_3)) + w_2\,(f(\vec{g}_4)+f(\vec{g}_5)+f(\vec{g}_6)) + w_3\,f(\vec{g}_7)\right)\ .$$

This is illustrated in Figure 94. The result is a 7×2 matrix $\mathbf{G}$ containing in each row the coordinates of one integration point $\vec{g}_j$ and a vector $\vec{w}\in\mathbb{R}^7$ with the corresponding integration weights. To determine the optimal values solve a

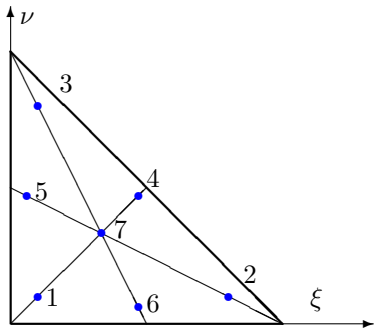


Figure 94: Gauss integration of order 5 on the standard triangle, using 7 integration points

nonlinear system of 5 equations for the unknowns λ_1, λ_2, w_1, w_2 and w_3. Require that ξ^k for $0\le k\le 5$ be integrated exactly. Find details in [Stah08]. Pick a solution of the resulting nonlinear system with $0<\lambda_1<\lambda_2<1$ (points should be inside the triangle) and positive weights w_1, w_2 and w_3.

This approximate integration yields the exact results for polynomials up to degree 5 . Thus for one triangle with diameter h and an area of the order h^2 the integration error for smooth functions is of the order $h^6\cdot h^2 = h^8$. When

[21] One might be temped to add the center of the triangle as a fourth point, but the resulting weight will be negative. This would lead to stiffness matrices that are not positive definite.

dividing a large domain in sub-triangles of size h this leads to a total integration error of the order h^6. For most problems this error will be considerably smaller than the approximation error of the FEM method and it is reasonably safe to ignore this contribution to the total error.

The optimal choice of Gauss points and integration weights is given by[22]

$$\mathbf{G} = \begin{bmatrix} \lambda_1/2 & \lambda_1/2 \\ 1-\lambda_1 & \lambda_1/2 \\ \lambda_1/2 & 1-\lambda_1 \\ \lambda_2/2 & \lambda_2/2 \\ 1-\lambda_2 & \lambda_2/2 \\ \lambda_2/2 & 1-\lambda_2 \\ 1/3 & 1/3 \end{bmatrix} \approx \begin{bmatrix} 0.101287 & 0.101287 \\ 0.797427 & 0.101287 \\ 0.101287 & 0.797427 \\ 0.470142 & 0.470142 \\ 0.059716 & 0.470142 \\ 0.470142 & 0.059716 \\ 0.333333 & 0.333333 \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{pmatrix} w_1 \\ w_1 \\ w_1 \\ w_2 \\ w_2 \\ w_2 \\ w_3 \end{pmatrix} \approx \begin{pmatrix} 0.0629696 \\ 0.0629696 \\ 0.0629696 \\ 0.0661971 \\ 0.0661971 \\ 0.0661971 \\ 0.1125000 \end{pmatrix} . \tag{49}$$

Using the general transformation results to compute the coordinates $\mathbf{X}_G$ for the Gauss integration in a general triangle by

$$\mathbf{X}_G = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \mathbf{G}^T . \tag{50}$$

This notation is used to compute the Gauss points for a given triangulation of the domain, i.e. for the mesh.

6.4 Construction of first order elements

Assume that the function u is linear on each triangle T_k, thus determined by the values at the three corners. Then all integrals in expression (46) have to be examined. For the linear elements use the integration with 3 Gauss nodes in the triangle, as described in Section 6.3.2. All contributions in (46)

$$0 = \iint_\Omega (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\;\phi\;dA - \int_{\Gamma_2} \phi\;(g_2 + g_3\,u)\;ds$$

have to be transformed into

$$0 = \langle \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\,,\,\vec{\phi}\rangle\,. \tag{51}$$

By integration over one triangle E find

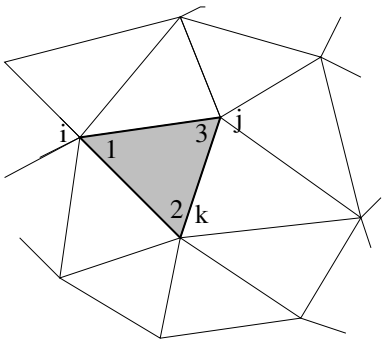
$$\iint_E (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\;\phi\;dA \approx \langle \mathbf{A}_E \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}\rangle + \langle \mathbf{W}_E \vec{f}_E\,, \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}\rangle\,.$$

The matrix $\mathbf{A}_E$ is the **element stiffness matrix** and $\mathbf{W}_E\vec{f}_E$ the corresponding vector. These entries have to be added in the correct rows and columns of the global stiffness matrix. For this examine the local and global numbering of nodes in Figure 95. In each triangle the three corners are numbered by 1,2 and 3, but in the global mesh (consisting of many triangles) they are numbered by i, k and j. Thus the entries in the element stiffness matrix $\mathbf{A}_E$ have to be added to rows/columns i,k and j in the global stiffness matrix $\mathbf{A}$.

$$\mathbf{A}_E = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \Longrightarrow \quad \mathbf{A} \longrightarrow \mathbf{A} + \begin{array}{c} \\ \text{row i} \\ \\ \text{row j} \\ \\ \text{row k} \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & \text{col i} & & \text{col j} & & \text{col k} & \end{array} \\ \begin{bmatrix} \ddots & \vdots & & \vdots & & \vdots & \\ \cdots & a_{11} & \cdots & a_{13} & \cdots\cdots & a_{12} & \cdots \\ & \vdots & \ddots & \vdots & & \vdots & \\ \cdots & a_{31} & \cdots & a_{33} & \cdots\cdots & a_{32} & \cdots \\ & \vdots & & \vdots & \ddots & \vdots & \\ \cdots & a_{21} & \cdots & a_{23} & \cdots\cdots & a_{22} & \cdots \\ & \vdots & & \vdots & & \vdots & \ddots \end{bmatrix} \end{array}$$

Similar procedures have to be appplied to the vectors.

[22] The exact values are $\lambda_1 = (12 - 2\sqrt{15})/21$, $\lambda_2 = (12 + 2\sqrt{15})/21$, $w_1 = (155 - \sqrt{15})/2400$, $w_4 = (155 + \sqrt{15})/2400$ and $w_7 = 9/80$.



local	$\longleftrightarrow$	global
triangle	$\longleftrightarrow$	mesh
1	$\longleftrightarrow$	i
2	$\longleftrightarrow$	k
3	$\longleftrightarrow$	j

Figure 95: Local and global numbering of nodes

6.4.1 Linear interpolation on a triangle

If the values of the function $\phi(x, y)$ at the three corners are given by ϕ_1, ϕ_2 and ϕ_3 then the values $\phi(\vec{g}_i)$ are given by

$$\begin{aligned}\phi(\vec{g}_1) &= \frac{2}{3}\,\phi_1 + \frac{1}{6}\,\phi_2 + \frac{1}{6}\,\phi_3\\ \phi(\vec{g}_2) &= \frac{1}{6}\,\phi_1 + \frac{2}{3}\,\phi_2 + \frac{1}{6}\,\phi_3\\ \phi(\vec{g}_3) &= \frac{1}{6}\,\phi_1 + \frac{1}{6}\,\phi_2 + \frac{2}{3}\,\phi_3\end{aligned}$$

or using a matrix notation

$$\begin{pmatrix}\phi(\vec{g}_1)\\ \phi(\vec{g}_3)\\ \phi(\vec{g}_3)\end{pmatrix} = \frac{1}{6}\begin{bmatrix}4 & 1 & 1\\ 1 & 4 & 1\\ 1 & 1 & 4\end{bmatrix}\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix} = \mathbf{M}\,\vec{\phi}\,.$$

This interpolation of the values from the nodes of the triangle to the Gauss points $\vec{g}_i$ is independent of shape and size of the triangle.

For second order elements the construction of this interpolation matrix is performed using the basis functions (see Section 6.5.1). For the linear case use the simpler basis functions

$$\vec{\Phi}(\xi,\nu) = \begin{pmatrix}\Phi_1(\xi,\nu)\\ \Phi_2(\xi,\nu)\\ \Phi_3(\xi,\nu)\end{pmatrix} = \begin{pmatrix}1-\xi-\nu\\ \xi\\ \nu\end{pmatrix}$$

and a linear interpolation of a function given at the nodes is given by

$$f(\xi,\nu) = \sum_{i=1}^{3} f_i\,\Phi_i(\xi,\nu)\,.$$

Since

$$\frac{\partial}{\partial\xi}\,\vec{\Phi}(\xi,\nu) = \begin{pmatrix}-1\\ 1\\ 0\end{pmatrix} \quad\text{and}\quad \frac{\partial}{\partial\nu}\,\vec{\Phi}(\xi,\nu) = \begin{pmatrix}-1\\ 0\\ 1\end{pmatrix}$$

observe that the gradient does not depend on the position within the triangle.

6.4.2 Integration of $f\,\phi$

Examine different methods to describe the function f: either by providing the values at the Gauss points, or by using the values at the nodes.

- If the values of the function f at the Gauss points $\vec{g}_i$ are denoted by f_i then this integral is approximated by

$$\begin{aligned}\iint\limits_E f\,\phi\,dA &\approx w\,2\,\text{area}(E)\,(f_1\,\phi(\vec{g}_1)+f_2\,\phi(\vec{g}_2)+f_3\,\phi(\vec{g}_3))\\ &= \frac{2\,\text{area}(E)}{6}\,\langle \mathbf{M}\,\vec{\phi}\,,\,\vec{f}\rangle = \frac{\text{area}(E)}{3}\,\langle\vec{\phi}\,,\,\mathbf{M}^T\,\vec{f}\rangle\,.\end{aligned}$$

  Thus find one contribution to (51).

- If the values of the function f at the nodes are denoted by f_i then first determine the values at the Gauss points by a linear interpolation. Then integrate as above, leading to the approximation

$$\iint\limits_E f\,\phi\,dA \approx \frac{2\,\text{area}(E)}{6}\,\langle \mathbf{M}\,\vec{\phi}\,,\,\mathbf{M}\,\vec{f}\rangle = \frac{\text{area}(E)}{3}\,\langle\vec{\phi}\,,\,\mathbf{M}^T\mathbf{M}\,\vec{f}\rangle\,.$$

  The matrix

$$\mathbf{M}^T\mathbf{M} = \frac{1}{36}\begin{bmatrix}18&9&9\\9&18&9\\9&9&18\end{bmatrix} = \frac{1}{4}\begin{bmatrix}2&1&1\\1&2&1\\1&1&2\end{bmatrix}$$

  is independent on the shape and size of the element (triangle). Thus find one contribution to (51).

6.4.3 Integration of $b_0\,u\,\phi$

Since the values of the functions u and ϕ are known at the nodes interpolate both functions and then use the values of the function $b_0(x,y)$ at the Gauss nodes to find

$$\begin{aligned}\iint\limits_E b_0\,u\,\phi\,dA &\approx w\,2\,\text{area}(E)\,\sum_{i=1}^{3} b_0(\vec{g}_i)\,u(\vec{g}_i)\,\phi(\vec{g}_1)\\ &= \frac{2\,\text{area}(E)}{6}\,\langle \mathbf{M}\,\vec{\phi}\,,\,\text{diag}(\vec{b})\,\mathbf{M}\,\vec{u}\rangle = \frac{\text{area}(E)}{3}\,\langle\vec{\phi}\,,\,\mathbf{M}^T\,\text{diag}(\vec{b}_0)\mathbf{M}\,\vec{u}\rangle\,,\end{aligned}$$

where

$$\text{diag}\,\vec{b}_0 = \begin{bmatrix}b_0(\vec{g}_1)&0&0\\0&b_0(\vec{g}_2)&0\\0&0&b_0(\vec{g}_3)\end{bmatrix}.$$

If $b_0(x,y)$ happens to be a constant, then the above simplifies to

$$\iint\limits_E b_0\,u\,\phi\,dA \approx b_0\,\frac{\text{area}(E)}{12}\,\langle\vec{\phi}\,,\,\begin{bmatrix}2&1&1\\1&2&1\\1&1&2\end{bmatrix}\vec{u}\rangle\,.$$

This is one of the contributions to (51).

6.4.4 Integration of $a\,\nabla u\cdot\nabla\phi$

Since the functions u and ϕ are linear on each triangle, we use the fact that the gradient is constant on each triangle. The gradient may be determined with the help of a normal vector of the plane passing through the three points

$$\begin{pmatrix}x_1\\y_1\\u_1\end{pmatrix}\,,\quad\begin{pmatrix}x_2\\y_2\\u_2\end{pmatrix}\quad\text{and}\quad\begin{pmatrix}x_3\\y_3\\u_3\end{pmatrix}.$$

A normal vector $\vec{n}$ is given by the vector product

$$\vec{n} = \begin{pmatrix}x_2-x_1\\y_2-y_1\\u_2-u_1\end{pmatrix}\times\begin{pmatrix}x_3-x_1\\y_3-y_1\\u_3-u_1\end{pmatrix} = \begin{pmatrix}+(y_2-y_1)\cdot(u_3-u_1)-(u_2-u_1)\cdot(y_3-y_1)\\-(x_2-x_1)\cdot(u_3-u_1)+(u_2-u_1)\cdot(x_3-x_1)\\+(x_2-x_1)\cdot(y_3-y_1)-(y_2-y_1)\cdot(x_3-x_1)\end{pmatrix}.$$

The third component of this vector equals twice the oriented[23] area of the triangle. To obtain the gradient in the first two components the vector has to be normalized, such that the third component equals -1, i.e.

$$\nabla u = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} = \frac{-1}{2\,\text{area}(E)} \begin{pmatrix} +(y_2-y_1)\cdot(u_3-u_1) - (u_2-u_1)\cdot(y_3-y_1) \\ -(x_2-x_1)\cdot(u_3-u_1) + (u_2-u_1)\cdot(x_3-x_1) \end{pmatrix}.$$

This formula can be written in the form

$$\nabla u = \frac{-1}{2\,\text{area}(E)} \begin{bmatrix} (y_3-y_2) & (y_1-y_3) & (y_2-y_1) \\ (x_2-x_3) & (x_3-x_1) & (x_1-x_2) \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{-1}{2\,\text{area}(E)}\, \mathbf{G} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \tag{52}$$

and thus

$$\langle \nabla\phi\,,\, \nabla u\rangle = \frac{1}{4\,\text{area}(E)^2} \langle \mathbf{G} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle = \frac{1}{4\,\text{area}(E)^2} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}^T\cdot\mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

If a_i are the values of the function $a(x,y)$ at the Gauss points $\vec{g}_i$ find

$$\iint\limits_E a\,\nabla\phi\cdot\nabla u\,dA \approx \frac{a_1+a_2+a_3}{3}\,\frac{1}{4\,\text{area}(E)} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}^T\cdot\mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

As an exercise one can verify that the matrix $\mathbf{G}^T\cdot\mathbf{G}$ is symmetric and positive semi-definite. The expression vanishes for constant vectors, i.e. for vanishing gradients.

6.4.5 Integration of $\mathbf{a}\cdot\nabla u\cdot\nabla\phi$ with a coefficient matrix $\mathbf{a}$

For a coefficient matrix $\mathbf{a}$ construct the matrix with the average over the values at the Gauss points.

$$\bar{\mathbf{a}} = \frac{1}{3} \begin{bmatrix} a_{11}(\vec{g}_1)+a_{11}(\vec{g}_2)+a_{11}(\vec{g}_3) & a_{12}(\vec{g}_1)+a_{12}(\vec{g}_2)+a_{12}(\vec{g}_3) \\ a_{12}(\vec{g}_1)+a_{12}(\vec{g}_2)+a_{12}(\vec{g}_3) & a_{22}(\vec{g}_1)+a_{22}(\vec{g}_2)+a_{22}(\vec{g}_3) \end{bmatrix} \in \mathbb{M}^{2\times 2}$$

Then use

$$\langle \nabla\phi\,,\, \bar{\mathbf{a}}\,\nabla u\rangle = \frac{1}{4\,\text{area}(E)^2} \langle \mathbf{G} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \bar{\mathbf{a}}\cdot\mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle = \frac{1}{4\,\text{area}(E)^2} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}^T\cdot\bar{\mathbf{a}}\cdot\mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

and

$$\iint\limits_E a\,\nabla\phi\cdot\nabla u\,dA \approx \frac{1}{4\,\text{area}(E)} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}^T\cdot\bar{\mathbf{a}}\cdot\mathbf{G} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

6.4.6 Integration of $u\,\vec{b}\cdot\nabla\phi$

Since the gradient of ϕ is constant on each of the triangles use

$$\begin{pmatrix} \phi_x \\ \phi_y \end{pmatrix} = \nabla\phi = \frac{-1}{2\,\text{area}(E)}\,\mathbf{G}\cdot\begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} = \frac{-1}{2\,\text{area}(E)} \begin{bmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{bmatrix} \cdot \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix},$$

[23] It is quietly assumed that the third component of $\vec{n}$ is positive. Since only the square of the gradient is used the influence of this ignorance will disappear. Generate meshes with triangles with a positive orientation also allows to assure $n_3 > 0$.

where

$$\mathbf{G_x} = \begin{bmatrix} y_3 - y_2 & y_1 - y_3 & y_2 - y_1 \end{bmatrix} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} x_2 - x_3 & x_3 - x_1 & x_1 - x_2 \end{bmatrix}.$$

Let $b_{1,i}$ be the values of the first component of $\vec{b}$ at the Gauss nodes and find

$$\begin{aligned}
\iint_E u\, b_1\, \phi_x \, dA &\approx \frac{\text{area}(E)}{3} \sum_{i=1}^{3} u(\vec{g}_i)\, b_{1,i}\, \phi_{x,i} \\
&= \frac{-\text{area}(E)}{3 \cdot 2\,\text{area}(E)} \left\langle \begin{bmatrix} \mathbf{G}_x \\ \mathbf{G}_x \\ \mathbf{G}_x \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{1,1} & 0 & 0 \\ 0 & b_{1,2} & 0 \\ 0 & 0 & b_{1,3} \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle \\
&= \frac{-1}{6} \left\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} \mathbf{G}_x^T & \mathbf{G}_x^T & \mathbf{G}_x^T \end{bmatrix} \begin{bmatrix} b_{1,1} & 0 & 0 \\ 0 & b_{1,2} & 0 \\ 0 & 0 & b_{1,3} \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle \\
&= \frac{-1}{6} \left\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}_x^T \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle \\
&= \frac{-1}{6} \left\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{1,1}(y_3 - y_2) & b_{1,2}(y_3 - y_2) & b_{1,3}(y_3 - y_2) \\ b_{1,1}(y_1 - y_3) & b_{1,2}(y_1 - y_3) & b_{1,3}(y_1 - y_3) \\ b_{1,1}(y_2 - y_1) & b_{1,2}(y_2 - y_1) & b_{1,3}(y_2 - y_1) \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle.
\end{aligned}$$

If the values of the second component of $\vec{b}$ at the Gauss nodes are given by $b_{2,i}$ find by similar computations

$$\begin{aligned}
\iint_E u\, b_2\, \phi_y \, dA &\approx \frac{-\text{area}(E)}{3} \sum_{i=1}^{3} u(\vec{g}_i)\, b_{2,i}\, \phi_{y,i} \\
&= \frac{-1}{6} \left\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}_y^T \begin{bmatrix} b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle \\
&= \frac{-1}{6} \left\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{2,1}(x_2 - x_3) & b_{2,2}(x_2 - x_3) & b_{2,3}(x_2 - x_3) \\ b_{2,1}(x_3 - x_1) & b_{2,2}(x_3 - x_1) & b_{2,3}(x_3 - x_1) \\ b_{2,1}(x_1 - x_2) & b_{2,2}(x_1 - x_2) & b_{2,3}(x_1 - x_2) \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle.
\end{aligned}$$

This leads to two more contributions to (51).

6.4.7 Integration over boundary segments

In expression (46) compute integrals over the boundary

$$\int_{\Gamma_2} \phi \ (g_2 + g_3\, u) \ ds\,.$$

For triangular domains the boundary consists of straight line segments. Replace the integral by a sum of line integrals and use a Gauss integration on each segment. Based on the two endpoints $\vec{x}_1$ and $\vec{x}_2$ use the values at the two Gauss

points[24]

$$\begin{aligned} \vec{p}_1 &= \tfrac{1}{2}\,(\vec{x}_1+\vec{x}_2) - \tfrac{1}{2\sqrt{3}}\,(\vec{x}_2-\vec{x}_1) \\ \vec{p}_2 &= \tfrac{1}{2}\,(\vec{x}_1+\vec{x}_2) + \tfrac{1}{2\sqrt{3}}\,(\vec{x}_2-\vec{x}_1) \end{aligned} \quad .$$

Polynomials up to degree 3 are integrated exactly, thus the error is proportional to h^4. By linear interpolation between the points $\vec{x}_1$ and $\vec{x}_2$ find the values of the function u at the Gauss points to be

$$\begin{aligned} u(\vec{p}_1) &= (1-\alpha)\,u_1 + \alpha\,u_2 \\ u(\vec{p}_2) &= \alpha\,u_1 + (1-\alpha)\,u_2 \end{aligned}$$

or

$$\begin{pmatrix} u(\vec{p}_1) \\ u(\vec{p}_2) \end{pmatrix} = \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} ,$$

where $\alpha = \frac{1-1/\sqrt{3}}{2} \approx 0.211325$. Using the length $L = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ this leads to the approximations

$$\begin{aligned} \int \phi\, g_2\, ds &\approx \frac{L}{2}\,\langle \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \end{pmatrix} \rangle \\ &= \frac{L}{2}\,\langle \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \end{pmatrix} \rangle \\ \int \phi\, g_3\, u\, ds &\approx \frac{L}{2}\,\langle \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \begin{bmatrix} g_3(\vec{p}_1) & 0 \\ 0 & g_3(\vec{p}_2) \end{bmatrix} \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \rangle \\ &= \frac{L}{2}\,\langle \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{bmatrix} (1-\alpha)\,g_3(\vec{p}_1) & \alpha\,g_3(\vec{p}_1) \\ \alpha\,g_3(\vec{p}_2) & (1-\alpha)\,g_3(\vec{p}_2) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \rangle \\ &= \frac{L}{2}\,\langle \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \begin{bmatrix} (1-\alpha)^2\,g_3(\vec{p}_1) + \alpha^2\,g_3(\vec{p}_2) & (1-\alpha)\,\alpha\,(g_3(\vec{p}_1)+g_3(\vec{p}_2)) \\ (1-\alpha)\,\alpha\,(g_3(\vec{p}_1)+g_3(\vec{p}_2)) & \alpha^2\,g_3(\vec{p}_1) + (1-\alpha)^2\,g_3(\vec{p}_2) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \rangle . \end{aligned}$$

The first expression will lead to a contribution to the RHS vector of the linear system to be solved, while the second expression will lead to entries in the matrix. These approximate integrations lead to the exact result if the function to be integrated is a polynomial of degree 3, or less. If h is the typical length of an edge then the error is of the order h^5 for one line segment and thus of order h^4 for the total boundary. This boundary integration is used for first order elements.

The second expression is of the form

$$\int \phi\, g_3\, u\, ds \approx \langle \vec{\phi}, \mathbf{B}\,\vec{u} \rangle = \langle \begin{pmatrix} \phi_2 \\ \phi_2 \end{pmatrix} , \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \rangle \tag{53}$$

and its effect on the linear system $\mathbf{A}\,\vec{u} + \mathbf{W}\,\vec{f} = \vec{0}$ to be solved depends on nodes being on the Dirichlet part of the boundary.

- If u_1 and u_2 are both free, i.e. not on the Dirichlet section, then all entries of the matrix $\mathbf{B}$ have to be added to the global stiffness matrix $\mathbf{A}$.
- If u_1 and u_2 are on the Dirichlet section, then nothing has to be added to $\mathbf{A}$ and $\vec{f}$.

[24] To derive the formula integrate 1, t, t^2 and t^3 over the interval $[-1,1]$.

$$\begin{array}{rcl} \int_{-1}^{+1} f(t)\,dt &=& w_1\,f(-\xi) + w_1\,f(+\xi) \\ \int_{-1}^{+1} 1\,dt = 2 &=& w_1\,1 + w_1\,1 \quad\Longrightarrow\quad w_1 = 1 \\ \int_{-1}^{+1} t\,dt = 0 &=& -\,w_1\,\xi + w_1\,\xi = 0 \\ \int_{-1}^{+1} t^2\,dt = \frac{2}{3} &=& +\,w_1\,\xi^2 + w_1\,\xi^2 \quad\Longrightarrow\quad \xi = \sqrt{1/3} \\ \int_{-1}^{+1} t^3\,dt = 0 &=& -\,w_1\,\xi^3 + w_1\,\xi^3 = 0 \end{array}$$

Thus t^4 is not integrated exactly and the error is proportional to h^4.

- If u_1 is free and u_2 is on the Dirichlet section, then only the first expression

$$b_{11}\,u_1 + b_{12}\,u_2 = b_{11}\,u_1 + b_{12}\,d_2$$

has to be added. d_2 is the Dirichlet value at the position of u_2. Then b_{11} has to be taken into account in $\mathbf{A}$ and $b_{12}\,d_2$ has to be added to $\mathbf{W}\,\vec{f}$.

- If u_2 is free and u_1 is on the Dirichlet section, then only the second expression $b_{21}\,u_1 + b_{22}\,u_2 = b_{21}\,d_1 + b_{22}\,u_2$ has to be added. d_1 is the Dirichlet value at the position of u_1. Then b_{22} has to be taken into account in $\mathbf{A}$ and $b_{12}\,d_1$ has to be added to $\mathbf{W}\,\vec{f}$.

6.5 Construction of second order elements

In this section the construction of the element stiffness matrix and vector for triangular elements or order 2 is examined. The ideas are very similar to Section 6.4 for linear basis functions, but using a bit more mathematics is required. Again all contributions in (46)

$$0 = \iint\limits_{\Omega} (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\;\phi\;dA - \int_{\Gamma_2} \phi\;\;(g_2 + g_3\,u)\;\;ds$$

have to be transformed into

$$0 = \langle \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\,,\;\vec{\phi}\rangle\,.$$

For second order element a general quadratic function is used on each of the triangles in the mesh. There are 6 linearly independent polynomials of degree 2 or less, namely 1, x, y, x^2, y^2 and $x \cdot y$.

6.5.1 The basis functions for a second order element and quadratic interpolation

Examine the standard triangle Ω in Figure 92 with the values of a function $f(\xi,\nu)$ at the corners and at the midpoints of the edges. Use the numbering as shown in Figure 92. The parameters ξ and ν at the nodes are given by Table 18. Construct polynomials $\phi_i(\xi,\nu)$ of degree 2, such that

$$\Phi_i(\xi_j,\nu_j) = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

i.e. each basis function is equal to 1 at one of the nodes and vanishes on all other nodes. These basis polynomials are given by

node i	1	2	3	4	5	6
ξ_i	0	1	0	$\frac{1}{2}$	0	$\frac{1}{2}$
ν_i	0	0	1	$\frac{1}{2}$	$\frac{1}{2}$	0

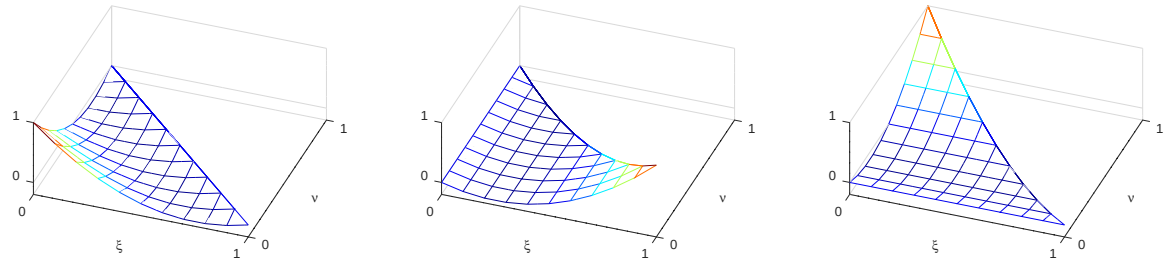
Table 18: Coordinates of the nodes in the standard quadratic triangle

$$\vec{\Phi}(\xi,\nu) = \begin{pmatrix} \Phi_1(\xi,\nu) \\ \Phi_2(\xi,\nu) \\ \Phi_3(\xi,\nu) \\ \Phi_4(\xi,\nu) \\ \Phi_5(\xi,\nu) \\ \Phi_6(\xi,\nu) \end{pmatrix} = \begin{pmatrix} (1-\xi-\nu)\,(1-2\,\xi-2\,\nu) \\ \xi\,(2\,\xi-1) \\ \nu\,(2\,\nu-1) \\ 4\,\xi\,\nu \\ 4\,\nu\,(1-\xi-\nu) \\ 4\,\xi\,(1-\xi-\nu) \end{pmatrix} \tag{54}$$

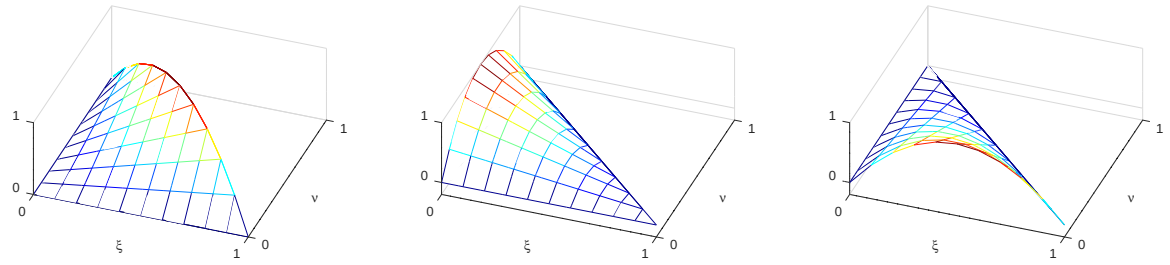
and find their graphs in Figure 96.

Any quadratic polynomial f on the standard triangle Ω can be written as linear combination of the basis functions by using

$$f(\xi,\nu) = \sum_{i=1}^{6} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu) = \sum_{i=1}^{6} f_i\,\Phi_i(\xi,\nu)\,. \tag{55}$$



(a) $\Phi_1(\xi,\nu) = (1-\xi-\nu)\,(1-2\,\xi-2\,\nu)$ (b) $\Phi_2(\xi,\nu) = \xi\,(2\,\xi-1)$ (c) $\Phi_3(\xi,\nu) = \nu\,(2\,\nu-1)$



(d) $\Phi_4(\xi,\nu) = 4\,\xi\,\nu$ (e) $\Phi_5(\xi,\nu) = 4\,\nu\,(1-\xi-\nu)$ (f) $\Phi_6(\xi,\nu) = 4\,\xi\,(1-\xi-\nu)$

Figure 96: Basis functions for second order triangular elements

This is the formula to apply a quadratic interpolation on the triangle, using the values f_i of the function at the nodes. To use this interpolation for a given point (x,y) in the triangle E in Figure 92 determine the correct values of the parameters ξ and ν, i.e. solve

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \nu \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix}.$$

This is equivalent to the linear system

$$\mathbf{T} \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

Since the 2×2 matrix $\mathbf{T}$ is invertible find

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

6.5.2 Determine values at the Gauss points and apply Gauss integration

Use equation (50) to determine the coordinates of the seven Gauss points. Then a function to be integrated can be evaluated at these Gauss points. Computing the values of the basis functions $\Phi_i(\xi,\nu)$ at the Gauss points $\vec{g}_j$ by $m_{j,i} = \Phi_i(\vec{g}_j)$ and write

$$f(\vec{g}_j) = \sum_{i=1}^{6} f_i\,\Phi_i(\vec{g}_j) = \sum_{i=1}^{6} m_{j,i}\,f_i$$

or using a matrix notation $\mathbf{M} \in \mathbb{R}^{7\times 6}$

$$\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,6} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,6} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_6 \end{pmatrix} = \mathbf{M} \cdot \vec{f} \tag{56}$$

$$\approx \begin{bmatrix} +0.474353 & -0.080769 & -0.080769 & 0.041036 & 0.323074 & 0.323074 \\ -0.080769 & +0.474353 & -0.080769 & 0.323074 & 0.041036 & 0.323074 \\ -0.080769 & -0.080769 & +0.474353 & 0.323074 & 0.323074 & 0.041036 \\ -0.052584 & -0.028075 & -0.028075 & 0.884134 & 0.112300 & 0.112300 \\ -0.028075 & -0.052584 & -0.028075 & 0.112300 & 0.884134 & 0.112300 \\ -0.028075 & -0.028075 & -0.052584 & 0.112300 & 0.112300 & 0.884134 \\ -0.111111 & -0.111111 & -0.111111 & 0.444444 & 0.444444 & 0.444444 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix}$$

The Gauss integration can be written in the form

$$\iint_{\Omega} f(\xi,\nu)\, dA \approx \sum_{j=1}^{7} w_j\, f(\vec{g}_j) = \langle \vec{w}\,,\, \mathbf{M}\cdot\vec{f}\,\rangle\,.$$

To integrate over the general triangle E use the transformation (47), i.e.

$$\iint_{E} f\, dA = \iint_{\Omega} f\left(\vec{x}\,(\xi,\nu)\right) \left|\det\left(\frac{\partial\,(x,y)}{\partial\,(\xi,\nu)}\right)\right| d\xi\, d\nu \approx |\det \mathbf{T}|\ \langle \vec{w}\,,\, \mathbf{M}\cdot\vec{f}\,\rangle\,.$$

Now all the tools to approximate the integrals required for the element stiffness matrix are available.

6.5.3 Integration of $f\,\phi$

The test function ϕ is given by its values $\vec{\phi}$ at the nodes, i.e. the corners of the triangle and the midpoints of the sides. Examine different methods to give the function f: either by providing the values at the Gauss points, or by using the values at the nodes.

- If the values of the function f at the Gauss points $\vec{g}_i$ are denoted by f_i then this integral is approximated by

$$\begin{aligned} \iint_{E} f\,\phi\, dA &\approx |\det(\mathbf{T})| \sum_{j=1}^{7} w_j\, f_j\, \phi(g_j) = |\det(\mathbf{T})|\, \langle \mathrm{diag}(\vec{w})\vec{f}\,,\, \mathbf{M}\,\vec{\phi}\rangle \\ &= |\det(\mathbf{T})|\, \langle \mathbf{M}^T\, \mathrm{diag}(\vec{w})\vec{f}\,,\, \vec{\phi}\rangle\,, \end{aligned}$$

  Thus find one contribution to (51).

- If the values of the function f at the nodes are denoted by f_i then first determine the values at the Gauss points by a quadratic interpolation. Then integrate as above, leading to the approximation

$$\iint_{E} f\,\phi\, dA \approx |\det(\mathbf{T})|\, \langle \mathrm{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\, \mathbf{M}\,\vec{\phi}\rangle = |\det(\mathbf{T})|\, \langle \mathbf{M}^T\, \mathrm{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\, \vec{\phi}\rangle\,.$$

The matrices $\mathbf{M}^T\,\mathrm{diag}(\vec{w})$ and $\mathbf{M}^T\,\mathrm{diag}(\vec{w})\,\mathbf{M}$ are independent on the triangle E.

6.5.4 Integration of $b_0\,u\,\phi$

Since the values of the functions u and ϕ are known at the nodes use an interpolation and then the function $b_0(x,y)$ at the Gauss nodes to find

$$\begin{aligned} \iint_{E} b_0\,u\,\phi\, dA &\approx |\det(\mathbf{T})| \sum_{j=1}^{7} w_j\, b_0(g_j)\, u(g_j)\, \phi(g_j) = |\det(\mathbf{T})|\, \langle \mathrm{diag}(\vec{w})\ \mathrm{diag}(\vec{b}_0)\mathbf{M}\,\vec{u}\,,\, \mathbf{M}\,\vec{\phi}\rangle \\ &= |\det(\mathbf{T})|\, \langle \mathbf{M}^T\,\mathrm{diag}(\vec{w})\ \mathrm{diag}(\vec{b}_0)\,\mathbf{M}\,\vec{u}\,,\, \vec{\phi}\rangle\,, \end{aligned}$$

where $\mathrm{diag}(\vec{b}_0) = \mathrm{diag}(b_0(\vec{g}_1), b_0(\vec{g}_2), b_0(\vec{g}_3), \ldots,\ b_0(\vec{g}_7))$.

6.5.5 Transformation of the gradient to the standard triangle

To examine the contributions containing ∇u or $\nabla \phi$ requires considerably more tools than the ones used in Section 6.4.4 for linear elements. For linear elements the gradients are constant on each of the triangles. For quadratic elements the gradients are linear functions and thus not constant. First examine how the gradient behave under the transformation to the standard triangle, only then use the above integration methods.

Using Section 6.3.1 the coordinates (ξ, ν) of the standard triangle are connected to the global coordinates (x, y) by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix}$$

or equivalently

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

If a function $f(x, y)$ is given on the general triangle E can pull it back to the standard triangle by

$$g(\xi, \nu) = f(x(\xi, \nu)\,,\, y(\xi, \nu))$$

and then compute the gradient of $g(\xi, \nu)$ with respect to its independent variables ξ and ν. The result will depend on the partial derivatives of f with respect to x and y. The standard chain rule implies

$$\begin{aligned} \frac{\partial}{\partial \xi} g(\xi, \nu) &= \frac{\partial}{\partial \xi} f(x(\xi, \nu)\,,\, y(\xi, \nu)) = \frac{\partial\, f(x,y)}{\partial x} \frac{\partial\, x}{\partial \xi} + \frac{\partial\, f(x,y)}{\partial y} \frac{\partial\, y}{\partial \xi} \\ &= \frac{\partial\, f(x,y)}{\partial x} (x_2 - x_1) + \frac{\partial\, f(x,y)}{\partial y} (y_2 - y_1) \\ \frac{\partial}{\partial \nu} g(\xi, \nu) &= \frac{\partial}{\partial \nu} f(x(\xi, \nu)\,,\, y(\xi, \nu)) = \frac{\partial\, f(x,y)}{\partial x} \frac{\partial\, x}{\partial \nu} + \frac{\partial\, f(x,y)}{\partial y} \frac{\partial\, y}{\partial \nu} \\ &= \frac{\partial\, f(x,y)}{\partial x} (x_3 - x_1) + \frac{\partial\, f(x,y)}{\partial y} (y_3 - y_1)\,. \end{aligned}$$

This can be written with the help of matrices in the form

$$\begin{pmatrix} \frac{\partial\, g}{\partial \xi} \\ \frac{\partial\, g}{\partial \nu} \end{pmatrix} = \begin{bmatrix} (x_2 - x_1) & (y_2 - y_1) \\ (x_3 - x_1) & (y_3 - y_1) \end{bmatrix} \cdot \begin{pmatrix} \frac{\partial\, f}{\partial x} \\ \frac{\partial\, f}{\partial y} \end{pmatrix} = \mathbf{T}^T \cdot \begin{pmatrix} \frac{\partial\, f}{\partial x} \\ \frac{\partial\, f}{\partial y} \end{pmatrix}$$

or equivalently

$$\left(\frac{\partial\, g}{\partial \xi}\,,\, \frac{\partial\, g}{\partial \nu} \right) = \left(\frac{\partial\, f}{\partial x}\,,\, \frac{\partial\, f}{\partial y} \right) \cdot \mathbf{T}\,. \tag{57}$$

This implies

$$\left(\frac{\partial\, f}{\partial x}\,,\, \frac{\partial\, f}{\partial y} \right) = \left(\frac{\partial\, g}{\partial \xi}\,,\, \frac{\partial\, g}{\partial \nu} \right) \cdot \mathbf{T}^{-1} = \frac{1}{\det \mathbf{T}} \left(\frac{\partial\, g}{\partial \xi}\,,\, \frac{\partial\, g}{\partial \nu} \right) \cdot \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix}$$

or by transposition

$$\begin{pmatrix} \frac{\partial\, f}{\partial x} \\ \frac{\partial\, f}{\partial y} \end{pmatrix} = \frac{1}{\det \mathbf{T}} \begin{bmatrix} y_3 - y_1 & -y_2 + y_1 \\ -x_3 + x_1 & x_2 - x_1 \end{bmatrix} \begin{pmatrix} \frac{\partial\, g}{\partial \xi} \\ \frac{\partial\, g}{\partial \nu} \end{pmatrix}. \tag{58}$$

Let g be a function on the standard triangle Ω given as a linear combination of the basis functions, i.e.

$$g(\xi, \nu) = \sum_{i=1}^{6} g_i\, \Phi_i(\xi, \nu)$$

where the basis function $\Phi_i(\xi,\nu)$ are given by (54). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (54). The result is

$$\operatorname{grad}\vec{\Phi} = \begin{bmatrix} -3+4\,\xi+4\,\nu & -3+4\,\xi+4\,\nu \\ 4\,\xi-1 & 0 \\ 0 & 4\,\nu-1 \\ 4\,\nu & 4\,\xi \\ -4\,\nu & 4-4\,\xi-8\,\nu \\ 4-8\,\xi-4\,\nu & -4\,\xi \end{bmatrix} = \begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix}. \tag{59}$$

Thus find on the standard triangle Ω

$$\left(\frac{\partial\, g}{\partial\xi}\,,\,\frac{\partial\, g}{\partial\nu}\right) = (g_1,g_2,g_3,g_4,g_5,g_6)\cdot\begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix} = \vec{g}^{\,T}\cdot\begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix}.$$

If the function $\varphi(x,y)$ is given on the general triangle E as linear combination of the basis functions on E find

$$\varphi(x,y) = \sum_{i=1}^{6} \varphi_i\;\Phi_i(\xi(x,y)\,,\,\nu(x,y))\,.$$

Now combine the results in this section to conclude

$$\left(\frac{\partial\,\varphi}{\partial x}\,,\,\frac{\partial\,\varphi}{\partial y}\right) = \left(\frac{\partial\,\varphi}{\partial\xi}\,,\,\frac{\partial\,\varphi}{\partial\nu}\right)\cdot\mathbf{T}^{-1} = \vec{\varphi}^{\,T}\cdot\begin{bmatrix} \vec{\Phi}_\xi & \vec{\Phi}_\nu \end{bmatrix}\cdot\mathbf{T}^{-1}$$

or by transposition

$$\begin{pmatrix} \frac{\partial\,\varphi}{\partial x} \\ \frac{\partial\,\varphi}{\partial y} \end{pmatrix} = \left(\mathbf{T}^{-1}\right)^T\cdot\begin{bmatrix} \vec{\Phi}_\xi^{\,T} \\ \vec{\Phi}_\nu^{\,T} \end{bmatrix}\cdot\vec{\varphi} = \frac{1}{\det(\mathbf{T})}\begin{bmatrix} +y_3-y_1 & -y_2+y_1 \\ -x_3+x_1 & +x_2-x_1 \end{bmatrix}\cdot\begin{bmatrix} \vec{\Phi}_\xi^{\,T} \\ \vec{\Phi}_\nu^{\,T} \end{bmatrix}\cdot\vec{\varphi}$$

and the same identities can be spelled out for the two components independently.

$$\frac{\partial\,\varphi}{\partial x} = \frac{1}{\det(\mathbf{T})}\left[\;(+y_3-y_1)\,\vec{\Phi}_\xi^{\,T}+(-y_2+y_1)\,\vec{\Phi}_\nu^{\,T}\;\right]\cdot\vec{\varphi}\,, \tag{60}$$

$$\frac{\partial\,\varphi}{\partial y} = \frac{1}{\det(\mathbf{T})}\left[\;(-x_3+x_1)\,\vec{\Phi}_\xi^{\,T}+(+x_2-x_1)\,\vec{\Phi}_\nu^{\,T}\;\right]\cdot\vec{\varphi}\,. \tag{61}$$

For the numerical integration use the values of the gradient at the Gauss integration points $\vec{g}_j = (\xi_j,\nu_j)$. The values of the function φ at the Gauss points can be computed with the help of the interpolation matrix $\mathbf{M}$ by

$$\begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M}\cdot\begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix}.$$

Similarly we define the interpolation matrices for the partial derivatives. Using

$$\mathbf{M}_\xi = \begin{bmatrix} -3+4\,\xi_1+4\,\nu_1 & 4\,\xi_1-1 & 0 & 4\,\nu_1 & -4\,\nu_1 & 4-8\,\xi_1-4\,\nu_1 \\ -3+4\,\xi_2+4\,\nu_2 & 4\,\xi_2-1 & 0 & 4\,\nu_2 & -4\,\nu_2 & 4-8\,\xi_2-4\,\nu_2 \\ \vdots & & & & & \vdots \\ -3+4\,\xi_7+4\,\nu_7 & 4\,\xi_7-1 & 0 & 4\,\nu_7 & -4\,\nu_7 & 4-8\,\xi_7-4\,\nu_7 \end{bmatrix}$$

$$\approx \begin{bmatrix} -2.18971 & -0.59485 & 0.00000 & 0.40515 & -0.40515 & 2.78456 \\ 0.59485 & 2.18971 & 0.00000 & 0.40515 & -0.40515 & -2.78456 \\ 0.59485 & -0.59485 & 0.00000 & 3.18971 & -3.18971 & 0.00000 \\ 0.76114 & 0.88057 & 0.00000 & 1.88057 & -1.88057 & -1.64170 \\ -0.88057 & -0.76114 & 0.00000 & 1.88057 & -1.88057 & 1.64170 \\ -0.88057 & 0.88057 & 0.00000 & 0.23886 & -0.23886 & 0.00000 \\ -0.33333 & 0.33333 & 0.00000 & 1.33333 & -1.33333 & 0.00000 \end{bmatrix}$$

find

$$\begin{pmatrix} \varphi_\xi(\vec{g}_1) \\ \varphi_\xi(\vec{g}_2) \\ \vdots \\ \varphi_\xi(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\xi \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix} .$$

Similarly write

$$\begin{aligned} \mathbf{M}_\nu &= \begin{bmatrix} -3+4\,\xi_1+4\,\nu_1 & 0 & 4\,\nu_1-1 & 4\,\xi_1 & 4-4\,\xi_1-8\,\nu_1 & -4\,\xi_1 \\ -3+4\,\xi_2+4\,\nu_2 & 0 & 4\,\nu_2-1 & 4\,\xi_2 & 4-4\,\xi_2-8\,\nu_2 & -4\,\xi_2 \\ \vdots & & & & & \vdots \\ -3+4\,\xi_7+4\,\nu_7 & 0 & 4\,\nu_7-1 & 4\,\xi_7 & 4-4\,\xi_7-8\,\nu_7 & -4\,\xi_7 \end{bmatrix} \\ &\approx \begin{bmatrix} -2.18971 & 0.00000 & -0.59485 & 0.40515 & 2.78456 & -0.40515 \\ 0.59485 & 0.00000 & -0.59485 & 3.18971 & 0.00000 & -3.18971 \\ 0.59485 & 0.00000 & 2.18971 & 0.40515 & -2.78456 & -0.40515 \\ 0.76114 & 0.00000 & 0.88057 & 1.88057 & -1.64170 & -1.88057 \\ -0.88057 & 0.00000 & 0.88057 & 0.23886 & 0.00000 & -0.23886 \\ -0.88057 & 0.00000 & -0.76114 & 1.88057 & 1.64170 & -1.88057 \\ -0.33333 & 0.00000 & 0.33333 & 1.33333 & 0.00000 & -1.33333 \end{bmatrix} \end{aligned}$$

and

$$\begin{pmatrix} \varphi_\nu(\vec{g}_1) \\ \varphi_\nu(\vec{g}_2) \\ \vdots \\ \varphi_\nu(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\nu \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix} .$$

The matrices $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ allow to compute the values of the partial derivatives at the Gauss points in the standard triangle Ω and they are independent on the general triangle E.

Combining the above two computations use the notation

$$\vec{x}_i = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi_i \\ \nu_i \end{pmatrix} \quad \text{for} \quad i = 1, 2, 3, \ldots, 7$$

and find for the first component $\varphi_x = \frac{\partial \varphi}{\partial x}$ of the gradient at the Gauss points

$$\begin{pmatrix} \varphi_x(\vec{x}_1) \\ \varphi_x(\vec{x}_2) \\ \vdots \\ \varphi_x(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\; (+y_3 - y_1)\, \mathbf{M}_\xi^T + (-y_2 + y_1)\, \mathbf{M}_\nu^T \; \right] \cdot \vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix} \varphi_y(\vec{x}_1) \\ \varphi_y(\vec{x}_2) \\ \vdots \\ \varphi_y(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\; (-x_3 + x_1)\, \mathbf{M}_\xi^T + (+x_2 - x_1)\, \mathbf{M}_\nu^T \; \right] \cdot \vec{\phi} .$$

The above results for $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ can be coded in *Octave* and then used to compute the element stiffness matrix.

6.5.6 Partial derivatives at the nodes

For post processing one also needs the partial derivatives of functions at the nodes. On the standard triangle Ω use the formulas for the partial derivatives of the basis functions in expression (59) to find them at the nodes, given by the (ξ, ν) coordinates in Table 18 for quadratic elements.

$$\begin{pmatrix} \varphi_\xi(\xi_1,\nu_1) \\ \varphi_\xi(\xi_2,\nu_2) \\ \varphi_\xi(\xi_3,\nu_3) \\ \varphi_\xi(\xi_4,\nu_4) \\ \varphi_\xi(\xi_5,\nu_5) \\ \varphi_\xi(\xi_6,\nu_6) \end{pmatrix} = \begin{bmatrix} -3 & 1 & 1 & 1 & -1 & -1 \\ -1 & 3 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 2 & 0 \\ 0 & 0 & -4 & -2 & -2 & 0 \\ 4 & -4 & 0 & -2 & 2 & 0 \end{bmatrix} \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix} = \mathbf{N}_\xi \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix}$$

and

$$\begin{pmatrix} \varphi_\nu(\xi_1,\nu_1) \\ \varphi_\nu(\xi_2,\nu_2) \\ \varphi_\nu(\xi_3,\nu_3) \\ \varphi_\nu(\xi_4,\nu_4) \\ \varphi_\nu(\xi_5,\nu_5) \\ \varphi_\nu(\xi_6,\nu_6) \end{pmatrix} = \begin{bmatrix} -3 & 1 & 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & 1 & 1 & -1 \\ 0 & 4 & 0 & 2 & 0 & 2 \\ 4 & 0 & -4 & -2 & 0 & 2 \\ 0 & -4 & 0 & -2 & 0 & -2 \end{bmatrix} \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix} = \mathbf{N}_\nu \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix} .$$

Then use the transformation formulas (60) and (61) to determine the gradient of a function on the general triangle

$$\varphi(x,y) = \sum_{i=1}^{6} \varphi_i \, \Phi_i(\xi(x,y), \nu(x,y))$$

at the nodes (x_i, y_i) in the general triangle E, leading to

$$\begin{pmatrix} \varphi_x(x_1,y_1) \\ \varphi_x(x_2,y_2) \\ \vdots \\ \varphi_x(x_6,y_6) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (+y_3 - y_1)\, \mathbf{N}_\xi^T + (-y_2 + y_1)\, \mathbf{N}_\nu^T \, \right] \cdot \vec{\varphi} \,,$$

$$\begin{pmatrix} \varphi_y(x_1,y_1) \\ \varphi_y(x_2,y_2) \\ \vdots \\ \varphi_y(x_6,y_6) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (-x_3 + x_1)\, \mathbf{N}_\xi^T + (+x_2 - x_1)\, \mathbf{N}_\nu^T \, \right] \cdot \vec{\varphi} \,.$$

These results are useful to evaluate the gradient at the nodes. Observe that the results depends on the triangle used for the interpolation and a node is typically member of more than one triangle.

6.5.7 Integration of $u\,\vec{b}\cdot\nabla\phi$ and $a\,\nabla u\cdot\nabla\phi$

The vector function $\vec{b}(\vec{x})$ has to be evaluated at the Gauss integration points $\vec{g}_j$. Then the integration of

$$\iint\limits_E u\,\vec{b}\,\nabla\phi\;dA = \iint\limits_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA + \iint\limits_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA$$

is approximated by

$$\iint\limits_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA \;\approx\; \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\big((y_3-y_1)\,\mathbf{M}_\xi^T + (-y_2+y_1)\,\mathbf{M}_\nu^T\big)\cdot\mathrm{diag}(\overrightarrow{wb}_1)\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle$$

$$\iint\limits_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA \;\approx\; \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\big((-x_3+x_1)\,\mathbf{M}_\xi^T + (x_2-x_1)\,\mathbf{M}_\nu^T\big)\cdot\mathrm{diag}(\overrightarrow{wb}_2)\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle\,.$$

The function $a\,\nabla u\cdot\nabla\phi = a\,(\frac{\partial u}{\partial x}\frac{\partial \phi}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial \phi}{\partial y})$ has to be evaluated at the Gauss integration points $\vec{g}_j$, then multiplied by the Gauss weights w_i and added up. Use the vector $\overrightarrow{wa}$ with the values of the function $a(x_i,y_i)$ and the weights w_i at the Gauss points to obtain

$$\begin{aligned}
\iint\limits_E a\,\frac{\partial u(\vec{x})}{\partial x}\,\frac{\partial \phi(\vec{x})}{\partial x}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial x}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial x}\,d\xi\,d\nu \\
&\approx \frac{|\det \mathbf{T}|}{(\det \mathbf{T})^2}\,\langle \mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle \\
\iint\limits_E a\,\frac{\partial u(\vec{x})}{\partial y}\,\frac{\partial \phi(\vec{x})}{\partial y}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial y}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial y}\,d\xi\,d\nu \\
&\approx \frac{|\det \mathbf{T}|}{(\det \mathbf{T})^2}\,\langle \mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_x &= \left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa})\cdot\left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right] \\
\mathbf{A}_y &= \left[\ (-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa})\cdot\left[\ (-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right].
\end{aligned}$$

6.5.8 Integration of $\mathbf{a}\cdot\nabla u\cdot\nabla\phi$ with a coefficient matrix $\mathbf{a}$

The integrals over elements to be examined are

$$\iint\limits_E \mathbf{a}\cdot\nabla u\cdot\nabla\phi\,dA = \iint\limits_E a_{11}\,\frac{\partial u}{\partial x}\,\frac{\partial \phi}{\partial x}+a_{12}\,\frac{\partial u}{\partial x}\,\frac{\partial \phi}{\partial y}+a_{12}\,\frac{\partial u}{\partial y}\,\frac{\partial \phi}{\partial x}+a_{22}\,\frac{\partial u}{\partial y}\,\frac{\partial \phi}{\partial y}\,dA\,.$$

Similar to the previous section this leads to

$$\begin{aligned}
\iint\limits_E a_{11}\,\frac{\partial u(\vec{x})}{\partial x}\,\frac{\partial \phi(\vec{x})}{\partial x}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a_{11}(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial x}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial x}\,d\xi\,d\nu \approx \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_{11}\vec{u}\,,\,\vec{\phi}\rangle \\
\iint\limits_E a_{12}\,\frac{\partial u(\vec{x})}{\partial x}\,\frac{\partial \phi(\vec{x})}{\partial y}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a_{12}(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial x}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial y}\,d\xi\,d\nu \approx \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_{12}\vec{u}\,,\,\vec{\phi}\rangle \\
\iint\limits_E a_{12}\,\frac{\partial u(\vec{x})}{\partial y}\,\frac{\partial \phi(\vec{x})}{\partial x}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a_{12}(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial y}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial x}\,d\xi\,d\nu \approx \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_{21}\vec{u}\,,\,\vec{\phi}\rangle \\
\iint\limits_E a_{22}\,\frac{\partial u(\vec{x})}{\partial y}\,\frac{\partial \phi(\vec{x})}{\partial y}\,dA &= |\det \mathbf{T}| \iint\limits_\Omega a_{22}(\vec{x}(\xi,\nu))\frac{\partial u(\vec{x}(\xi,\nu))}{\partial y}\,\frac{\partial \phi(\vec{x}(\xi,\nu))}{\partial y}\,d\xi\,d\nu \approx \frac{1}{|\det \mathbf{T}|}\,\langle \mathbf{A}_{22}\vec{u}\,,\,\vec{\phi}\rangle
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_{11} &= \left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa}_{11})\cdot\left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right] \\
\mathbf{A}_{12} &= \left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa}_{12})\cdot\left[\ (-x_3+x1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right] \\
\mathbf{A}_{21} &= \left[\ (-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa}_{12})\cdot\left[\ (+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\ \right] \\
\mathbf{A}_{22} &= \left[\ (-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right]^T\cdot \mathrm{diag}(\overrightarrow{wa}_{22})\cdot\left[\ (-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\ \right].
\end{aligned}$$

Since $\mathbf{A}_{12}^T=\mathbf{A}_{21}\in\mathbb{R}^{2\times 2}$ use[25] $\mathbf{A}_{12}+\mathbf{A}_{21}=2\,\mathbf{A}_{12}$ to conclude

$$\iint\limits_E \mathbf{a}\cdot\nabla u\cdot\nabla\phi\,dA \approx \frac{1}{|\det(T)|}\,\langle(\mathbf{A}_{11}+2\,\mathbf{A}_{12}+\mathbf{A}_{22})\cdot\vec{u}\,,\,\vec{\phi}\rangle\,.$$

[25] This is not correct if complex coefficients are used.

6.5.9 Integration over boundary segments

In expression (46) compute integrals over the boundary

$$\int_{\Gamma_2} \phi \ (g_2 + g_3\, u) \ ds\,.$$

For triangular domains the boundary consists of straight line segments. Thus replace the integral by a sum of line integrals and use a Gauss integration on each segment. Based on the two endpoints $\vec{x}_1$ and $\vec{x}_3$ and the midpoint $\vec{x}_2 = \frac{1}{2}\,(\vec{x}_1 + \vec{x}_3)$ use the values at three Gauss integration points. Based on[26]

$$\int_{-h/2}^{h/2} f(x)\ dx \approx \frac{h}{18}\ \left(5\, f(-\frac{\sqrt{3}}{2\sqrt{5}}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{3}}{2\sqrt{5}}\, h)\right)$$

polynomials up to degree 5 are integrated exactly, thus the error on one interval is proportional to h^7. To evaluate a function at the Gauss points

$$\begin{array}{rcl} \vec{p}_1 & = & \frac{1}{2}\ (\vec{x}_1 + \vec{x}_3) - \frac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_3 - \vec{x}_1) \\ \vec{p}_2 & = & \vec{x}_2 \ = \ \frac{1}{2}\ (\vec{x}_1 + \vec{x}_3) \\ \vec{p}_3 & = & \frac{1}{2}\ (\vec{x}_1 + \vec{x}_3) + \frac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_3 - \vec{x}_1) \end{array}$$

use a quadratic interpolation of a function with $f_- = f(-h/2)$, $f_0 = f(0)$ and $f_+ = f(+h/2)$. Since[27]

$$f(x) = f_0 + \frac{f_+ - f_-}{h}\, x + 2\, \frac{f_- - 2\, f_0 + f_+}{h^2}\, x^2$$

the quadratic interpolation result at $\pm\alpha\, h$ is

$$\begin{array}{rcl} f(\pm\alpha\, h) & = & f_0 \pm (f_+ - f_-)\, \alpha + 2\, (f_- - 2\, f_0 + f_+)\, \alpha^2 \\ & = & f_- \ (\pm\alpha + 2\, \alpha^2) + f_0\, (1 - 4\, \alpha^2) + f_+\ (\mp\alpha + 2\, \alpha^2) \end{array}$$

where $\alpha = \frac{\sqrt{3}}{2\sqrt{5}} = \frac{\sqrt{15}}{10} \approx 0.316$. If a function u is given at the two endpoints by u_1 and u_3 and at the midpoint by u_2 obtain

$$\begin{array}{rcl} \begin{pmatrix} u(\vec{p}_1) \\ u(\vec{p}_2) \\ u(\vec{p}_3) \end{pmatrix} & = & \begin{bmatrix} +\alpha + 2\,\alpha^2 & 1 - 4\,\alpha^2 & -\alpha + 2\,\alpha^2 \\ 0 & 1 & 0 \\ -\alpha + 2\,\alpha^2 & 1 - 4\,\alpha^2 & +\alpha + 2\,\alpha^2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \\ & = & \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \approx \begin{bmatrix} +0.68730 & 0.4 & -0.08730 \\ 0 & 1 & 0 \\ -0.08730 & 0.4 & +0.68730 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \end{array} \tag{62}$$

[26] To derive the 3 point Gauss integration scheme use

$$\begin{array}{rcl} \int_{-1}^{+1} f(t)\, dt & = & w_1\, f(-\xi) + w_0\, f(0) + w_1\, f(+\xi) \\ \int_{-1}^{+1} 1\, dt \ = \ 2 & = & w_1\, 1 + w_0\, 1 + w_1\, 1 \\ \int_{-1}^{+1} t\, dt \ = \ 0 & = & -\, w_1\, \xi + w_0\, 0 + w_1\, \xi \ = \ 0 \\ \int_{-1}^{+1} t^2\, dt \ = \ \frac{2}{3} & = & +\, w_1\, \xi^2 + w_1\, \xi^2 \\ \int_{-1}^{+1} t^3\, dt \ = \ 0 & = & -\, w_1\, \xi^3 + w_1\, \xi^3 \ = \ 0 \\ \int_{-1}^{+1} t^4\, dt \ = \ \frac{2}{5} & = & +\, w_1\, \xi^4 + w_1\, \xi^4 \\ \int_{-1}^{+1} t^5\, dt \ = \ 0 & = & -\, w_1\, \xi^5 + w_1\, \xi^5 \ = \ 0 \end{array}$$

Thus t^6 is not integrated exactly and the error is proportional to h^6. The system to be solved is

$$\left\{ \begin{array}{rcl} w_0 + 2\, w_1 & = & 2 \\ 2\, w_1\, \xi^2 & = & \frac{2}{3} \\ 2\, w_1\, \xi^4 & = & \frac{2}{5} \end{array} \right. \quad \Longrightarrow \quad \xi^2 = \frac{3}{5},\ w_1 = \frac{5}{9},\ w_0 = \frac{8}{9}\,.$$

[27] To verify use $f(0) = f_0$ and

$$f(\pm h/2) = f_0 \pm \frac{f_+ - f_-}{h}\, \frac{h}{2} + 2\, \frac{f_- - 2\, f_0 + f_+}{h^2}\, \frac{h^2}{4} = f_0 \pm \frac{1}{2}(f_+ - f_-) + \frac{1}{2}\,(f_- - 2\, f_0 + f_+)\,.$$

With the length $L = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$ of the segment this leads to the approximations

$$\begin{aligned}
\int_{\text{edge}} \phi\, g_2\, ds &\approx \frac{L}{18}\, \langle \mathbf{M}_B \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle = \frac{L}{18}\, \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{M}_B^T \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle \\
\int_{\text{edge}} \phi\, g_3\, u\, ds &\approx \frac{L}{18}\, \langle \mathbf{M}_B \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} 5\, g_3(\vec{p}_1) & 0 & 0 \\ 0 & 8\, g_3(\vec{p}_2) & 0 \\ 0 & 0 & 5\, g_3(\vec{p}_3) \end{bmatrix} \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle \\
&= \frac{L}{18}\, \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{M}_B^T \begin{bmatrix} 5\, g_3(\vec{p}_1) & 0 & 0 \\ 0 & 8\, g_3(\vec{p}_2) & 0 \\ 0 & 0 & 5\, g_3(\vec{p}_3) \end{bmatrix} \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle .
\end{aligned}$$

The first expression will lead to a contribution to the RHS vector of the linear system to be solved, while the second expression will lead to entries in the matrix. These approximate integrations lead to the exact result if the function to be integrated is a polynomial of degree 5, or less. If h is the typical length of an edge then the error is of the order h^7 for one line segment and thus of order h^6 for the total boundary. This boundary integration is used for the second order elements.

The second expression is of the form

$$\int \phi\, g_3\, u\, ds \approx \langle \vec{\phi}, \mathbf{B}\, \vec{u} \rangle = \langle \begin{pmatrix} \phi_2 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle$$

and its effect on the linear system $\mathbf{A}\, \vec{u} + \mathbf{W}\, \vec{f} = \vec{0}$ depends on nodes being on the Dirichlet part of the boundary.

- If u_1 and u_3 are both free, i.e. not on the Dirichlet section, then u_2 is free too. All entries of the matrix $\mathbf{B}$ have to be added to the global stiffness matrix $\mathbf{A}$.
- If u_1 and u_3 are on the Dirichlet section, then nothing has to be added to $\mathbf{A}$ and $\vec{f}$.
- If u_1 and u_2 are free and u_3 is on the Dirichlet section, then only the first two expressions

  $$\begin{aligned} b_{11}\, u_1 + b_{12}\, u_2 + b_{13}\, u_3 &= b_{11}\, u_1 + b_{12}\, u_2 + b_{13}\, d_3 \\ b_{21}\, u_1 + b_{22}\, u_2 + b_{23}\, u_3 &= b_{21}\, u_1 + b_{22}\, u_2 + b_{23}\, d_3 \end{aligned}$$

  have to be added. d_3 is the Dirichlet value at the position of u_3. $b_{13}\, g_3$ and $b_{23}\, d_3$ have to be added to $\mathbf{W}\, \vec{f}$, the other expression to $\mathbf{A}$.
- If u_2 and u_3 are free and u_1 is on the Dirichlet section, then only the second and third expressions

  $$\begin{aligned} b_{21}\, u_1 + b_{22}\, u_2 + b_{23}\, u_3 &= b_{21}\, d_1 + b_{22}\, u_2 + b_{23}\, u_3 \\ b_{31}\, u_1 + b_{32}\, u_2 + b_{33}\, u_3 &= b_{31}\, d_1 + b_{32}\, u_2 + b_{33}\, u_3 \end{aligned}$$

  have to be added. d_1 is the Dirichlet value at the position of u_1. $b_{21}\, g_1$ and $b_{31}\, d_1$ have to be added to $\mathbf{W}\, \vec{f}$, the other expression to $\mathbf{A}$.
- If u_1 and u_3 are free, then u_2 has to be free too, since it is the midpoint of a Neumann section of the boundary.

6.6 Construction of third order elements

In this section the construction of the element stiffness matrix and vector for triangular elements or order 3 is examined. The ideas are extremely similar to Section 6.5 for quadratic functions. Again all contributions in (46)

$$0 = \iint_{\Omega} (a\, \nabla u - u\, \vec{b}) \cdot \nabla \phi + (b_0\, u - f)\, \phi\, dA - \int_{\Gamma_2} \phi\, (g_2 + g_3\, u)\, ds$$

have to be transformed into

$$0 = \langle \mathbf{A}\vec{u} + \mathbf{W}\vec{f}, \vec{\phi} \rangle . \tag{63}$$

For third order elements a general cubic function is used on each of the triangles in the mesh. There are 10 linearly independent polynomials of degree 3 or less, namely 1, x, y, x^2, $x\, y$, y^2, x^3, $x^2 y$, $x\, y^2$ and y^3.

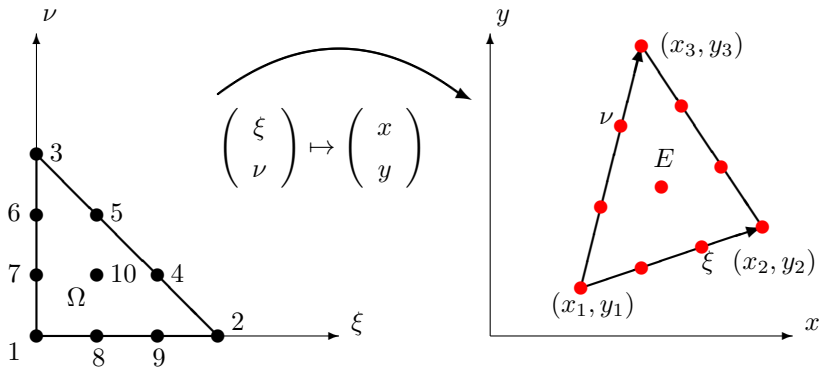


Figure 97: Transformation of the cubic standard triangle Ω to a general triangle E

6.6.1 The basis functions for a third order element and cubic interpolation

Examine the standard triangle Ω in Figure 97 with the values of a function $f(\xi, \nu)$ at the corners, the points on the edges and the mid point. Use the numbering as shown in Figure 97. The parameters ξ and ν at the nodes are given by Table 19. Construct polynomials $\phi_i(\xi, \nu)$ of degree 3, such that

$$\Phi_i(\xi_j, \nu_j) = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

i.e. each basis function is equal to 1 at one of the nodes and vanishes on all other nodes. These basis polynomials are given by[28]

node i	1	2	3	4	5	6	7	8	9	10
ξ_i	0	1	0	$\frac{2}{3}$	$\frac{1}{3}$	0	0	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$
ν_i	0	0	1	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	0	0	$\frac{1}{3}$

Table 19: Coordinates of the nodes in the standard cubic triangle

$$\vec{\Phi}(\xi, \nu) = \begin{pmatrix} \Phi_1(\xi, \nu) \\ \Phi_2(\xi, \nu) \\ \Phi_3(\xi, \nu) \\ \Phi_4(\xi, \nu) \\ \Phi_5(\xi, \nu) \\ \Phi_6(\xi, \nu) \\ \Phi_7(\xi, \nu) \\ \Phi_8(\xi, \nu) \\ \Phi_9(\xi, \nu) \\ \Phi_{10}(\xi, \nu) \end{pmatrix} = \begin{pmatrix} (1 - (\xi + \nu))\,(1 - 3\,(\xi + \nu))\,(1 - \frac{3}{2}\,(\xi + \nu)) \\ \xi\,(3\,\xi - 1)\,(\frac{3}{2}\,\xi - 1) \\ \nu\,(3\,\nu - 1)\,(\frac{3}{2}\,\nu - 1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\xi - 1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\nu - 1) \\ \frac{9}{2}\,\nu\,(1 - (\xi + \nu))\,(3\,\nu - 1) \\ 9\,\nu\,(1 - (\xi + \nu))\,(1 - \frac{3}{2}\,(\xi + \nu)) \\ 9\,\xi\,(1 - \frac{3}{2}\,(\xi + \nu))\,(1 - (\xi + \nu)) \\ \frac{9}{2}\,\xi\,(3\,\xi - 1)\,(1 - (\xi + \nu)) \\ 27\,\xi\,\nu\,(1 - (\xi + \nu)) \end{pmatrix} \tag{64}$$

[28] Use that the level curves of the functions ξ, ν and $1 - (\xi + \nu)$ at the levels 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1 are straight lines through the nodes. For each node use these functions to write down a polynomial vanishing at all other nodes, then choose the leading factor such that at the node the value equals 1.

$$= \begin{pmatrix} 1 - \frac{11}{2}\,\xi - \frac{11}{2}\,\nu + 9\,\xi^2 + 18\,\xi\,\nu + 9\,\nu^2 - \frac{9}{2}\,\xi^3 - \frac{27}{2}\,\xi^2\,\nu - \frac{27}{2}\,\xi\,\nu^2 - \frac{9}{2}\,\nu^3 \\ \xi - \frac{9}{2}\,\xi^2 + \frac{9}{2}\,\xi^3 \\ \nu - \frac{9}{2}\,\nu^2 + \frac{9}{2}\,\nu^3 \\ -\frac{9}{2}\,\xi\,\nu + \frac{27}{2}\,\xi^2\,\nu \\ -\frac{9}{2}\,\xi\,\nu + \frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\nu + \frac{9}{2}\,\xi\,\nu + 18\,\nu^2 - \frac{27}{2}\,\xi\,\nu^2 - \frac{27}{2}\,\nu^3 \\ 9\,\nu - \frac{45}{2}\,\xi\,\nu - \frac{45}{2}\,\nu^2 + \frac{27}{2}\,\xi^2\,\nu + 27\,\xi\,\nu^2 + \frac{27}{2}\,\nu^3 \\ 9\,\xi - \frac{45}{2}\,\xi^2 - \frac{45}{2}\,\xi\,\nu + \frac{27}{2}\,\xi^3 + 27\,\xi^2\,\nu + \frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\xi + 18\,\xi^2 + \frac{9}{2}\,\xi\,\nu - \frac{27}{2}\,\xi^3 - \frac{27}{2}\,\xi^2\,\nu \\ 27\,\xi\,\nu - 27\,\xi^2\,\nu - 27\,\xi\,\nu^2 \end{pmatrix} \tag{65}$$

and find their graphs in Figure 98.

Any cubic polynomial f on the standard triangle Ω can be written as linear combination of the 10 basis functions by using

$$f(\xi,\nu) = \sum_{i=1}^{10} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu) = \sum_{i=1}^{10} f_i\,\Phi_i(\xi,\nu)\,. \tag{66}$$

This is the formula to apply a cubic interpolation on the triangle, using the values $f_i = f(\xi_i,\nu_i)$ of the function at the nodes. To use this interpolation for a given point (x,y) in the triangle E in Figure 97. The transformation form the standard triangle Ω to the general triangle E is identical to the second order elements, i.e.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \begin{pmatrix} \xi \\ \nu \end{pmatrix}$$

and

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

6.6.2 Determine values at the Gauss points and apply Gauss integration

Use equation (50) (page 167) to determine the coordinates of the seven Gauss points. Then a function to be integrated can be evaluated at these Gauss points. Determine the values of the basis functions $\Phi_i(\xi,\nu)$ at the Gauss points $\vec{g}_j$ by $m_{j,i} = \Phi_i(\vec{g}_j)$ and write

$$f(\vec{g}_j) = \sum_{i=1}^{10} f_i\,\Phi_i(\vec{g}_j) = \sum_{i=1}^{10} m_{j,i}\,f_i$$

or using a matrix notation with $\mathbf{M} \in \mathbb{R}^{7\times 10}$

$$\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,10} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,10} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{10} \end{pmatrix} = \mathbf{M} \cdot \vec{f} \approx \tag{67}$$

$$\approx \begin{bmatrix} +0.22 & +0.06 & +0.06 & -0.03 & -0.03 & -0.25 & +0.51 & +0.51 & -0.25 & +0.22 \\ +0.06 & +0.22 & +0.06 & +0.51 & -0.25 & -0.03 & -0.03 & -0.25 & +0.51 & +0.22 \\ +0.06 & +0.06 & +0.22 & -0.25 & +0.51 & +0.51 & -0.25 & -0.03 & -0.03 & +0.22 \\ +0.04 & -0.06 & -0.06 & +0.41 & +0.41 & +0.05 & -0.10 & -0.10 & +0.05 & +0.36 \\ -0.06 & +0.04 & -0.06 & -0.10 & +0.05 & +0.41 & +0.41 & +0.05 & -0.10 & +0.36 \\ -0.06 & -0.06 & +0.04 & +0.05 & -0.10 & -0.10 & +0.05 & +0.41 & +0.41 & +0.36 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_9 \\ f_{10} \end{pmatrix}$$

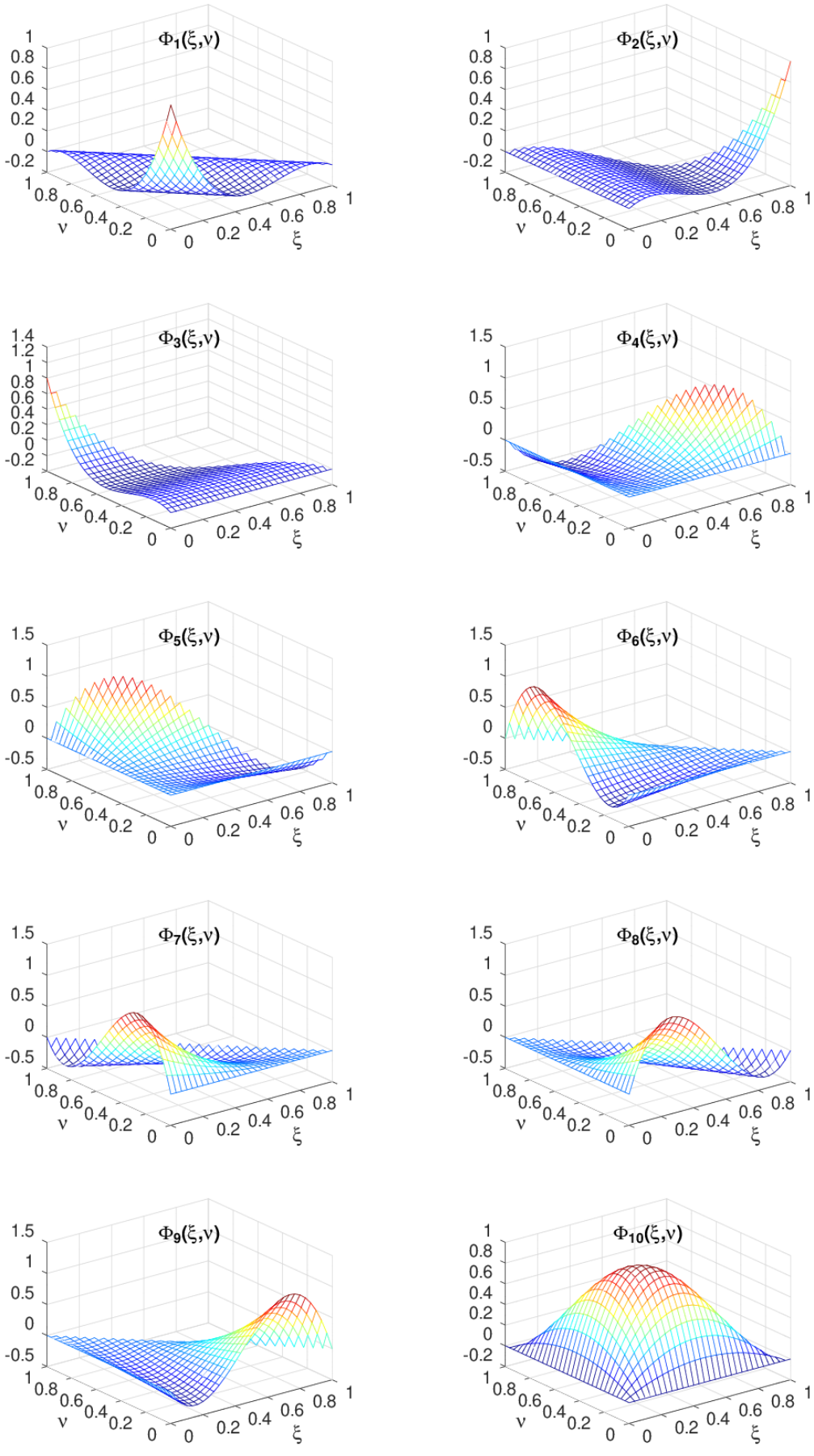


Figure 98: The 10 basis functions for third order triangular elements

The Gauss integration can be written in the form

$$\iint\limits_{\Omega} f(\xi,\nu)\; dA \approx \sum_{j=1}^{7} w_j\, f(\vec{g}_j) = \langle \vec{w}\,,\, \mathbf{M}\cdot\vec{f}\rangle\,.$$

To integrate over the general triangle E use the transformation (47), i.e.

$$\iint\limits_{E} f\; dA = \iint\limits_{\Omega} f\left(\vec{x}\,(\xi,\nu)\right)\; \left|\det\left(\frac{\partial\,(x,y)}{\partial\,(\xi,\nu)}\right)\right|\; d\xi\, d\nu \approx |\det \mathbf{T}|\;\; \langle \vec{w}\,,\, \mathbf{M}\cdot\vec{f}\rangle\,.$$

Now all the tools to approximate the integrals required for the element stiffness matrix are available.

6.6.3 Integration of $f\,\phi$ and $b_0\,u\,\phi$

These integrations are identical to the case of quadratic elements The test function ϕ is given by its values $\vec{\phi}$ at the nodes, i.e. the corners of the triangle and the two points on each side.

- If the values of the function f at the Gauss points $\vec{g}_i$ are denoted by f_i then this integral is approximated by

$$\iint\limits_{E} f\,\phi\; dA \approx |\det(\mathbf{T})| \sum_{j=1}^{7} w_j\, f_j\, \phi(g_j) \;=\; |\det(\mathbf{T})|\,\langle \mathrm{diag}(\vec{w})\vec{f}\,,\, \mathbf{M}\,\vec{\phi}\rangle = |\det(\mathbf{T})|\,\langle \mathbf{M}^T\,\mathrm{diag}(\vec{w})\vec{f}\,,\, \vec{\phi}\rangle\,.$$

  Thus find one contribution to (63).

- If the values of the function f at the nodes are denoted by f_i then first determine the values at the Gauss points by a cubic interpolation. Then integrate as above, leading to

$$\iint\limits_{E} f\,\phi\; dA \approx |\det(\mathbf{T})|\,\langle \mathrm{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\, \mathbf{M}\,\vec{\phi}\rangle = |\det(\mathbf{T})|\,\langle \mathbf{M}^T\,\mathrm{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\, \vec{\phi}\rangle\,.$$

- Since the values of the functions u and ϕ are known at the nodes use an interpolation and then the function $b_0(x,y)$ at the Gauss nodes to find

$$\begin{aligned}\iint\limits_{E} b_0\,u\,\phi\; dA \;&\approx\; |\det(\mathbf{T})| \sum_{j=1}^{7} w_j\, b_0(g_j)\, u(g_j)\, \phi(g_j) \;=\; |\det(\mathbf{T})|\,\langle \mathrm{diag}(\vec{w})\;\mathrm{diag}(\vec{b}_0)\mathbf{M}\,\vec{u}\,,\, \mathbf{M}\,\vec{\phi}\rangle \\ &=\; |\det(\mathbf{T})|\,\langle \mathbf{M}^T\,\mathrm{diag}(\vec{w})\;\mathrm{diag}(\vec{b}_0)\,\mathbf{M}\,\vec{u}\,,\, \vec{\phi}\rangle\,.\end{aligned}$$

The matrices $\mathbf{M}^T\,\mathrm{diag}(\vec{w})$ and $\mathbf{M}^T\,\mathrm{diag}(\vec{w})\,\mathbf{M}$ are again independent on the triangle E, but different from the case of quadratic elements.

6.6.4 Transformation of the gradient to the standard triangle

Computing the partial derivatives is again very similar to the case of quadratic elements. If a function $f(x,y)$ is given on the general triangle E can pull it back to the standard triangle by

$$g(\xi,\nu) = f(x(\xi,\nu)\,,\, y(\xi,\nu))$$

and then compute the gradient of $g(\xi,\nu)$ with respect to its independent variables ξ and ν. The result is This can be written with the help of matrices in the form

$$\begin{pmatrix} \frac{\partial\, g}{\partial \xi} \\ \frac{\partial\, g}{\partial \nu} \end{pmatrix} = \begin{bmatrix} (x_2-x_1) & (y_2-y_1) \\ (x_3-x_1) & (y_3-y_1) \end{bmatrix} \cdot \begin{pmatrix} \frac{\partial\, f}{\partial x} \\ \frac{\partial\, f}{\partial y} \end{pmatrix} = \mathbf{T}^T \cdot \begin{pmatrix} \frac{\partial\, f}{\partial x} \\ \frac{\partial\, f}{\partial y} \end{pmatrix}$$

or equivalently

$$\left(\frac{\partial\, g}{\partial \xi}\,,\, \frac{\partial\, g}{\partial \nu}\right) = \left(\frac{\partial\, f}{\partial x}\,,\, \frac{\partial\, f}{\partial y}\right)\cdot \mathbf{T}\,,$$

or

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \frac{1}{\det \mathbf{T}} \begin{bmatrix} y_3 - y_1 & -y_2 + y_1 \\ -x_3 + x_1 & x_2 - x_1 \end{bmatrix} \begin{pmatrix} \frac{\partial g}{\partial \xi} \\ \frac{\partial g}{\partial \nu} \end{pmatrix}.$$

Let u be a function on the standard triangle Ω given as a linear combination of the basis functions, i.e. $u(\xi,\nu) = \sum_{i=1}^{10} u_i\ \Phi_i(\xi,\nu)$, where the basis function $\Phi_i(\xi,\nu)$ are given by (65). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (65). The results are

$$\vec{\Phi}_\xi(\xi,\nu) = \frac{\partial}{\partial \xi}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 1 - 9\,\xi + \frac{27}{2}\,\xi^2 \\ 0 \\ -\frac{9}{2}\,\nu + 27\,\xi\,\nu \\ -\frac{9}{2}\,\nu + \frac{27}{2}\,\nu^2 \\ \frac{9}{2}\,\nu - \frac{27}{2}\,\nu^2 \\ -\frac{45}{2}\,\nu + 27\,\xi\,\nu + 27\,\nu^2 \\ 9 - 45\,\xi - \frac{45}{2}\,\nu + \frac{81}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2} + 36\,\xi + \frac{9}{2}\,\nu - \frac{81}{2}\,\xi^2 - 27\,\xi\,\nu \\ 27\,\nu - 54\,\xi\,\nu - 27\,\nu^2 \end{pmatrix} \tag{68}$$

and

$$\vec{\Phi}_\nu(\xi,\nu) = \frac{\partial}{\partial \nu}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 0 \\ 1 - 9\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2}\,\xi + \frac{27}{2}\,\xi^2 \\ -\frac{9}{2}\,\xi + 27\,\xi\,\nu \\ -\frac{9}{2} + \frac{9}{2}\,\xi + 36\,\nu - 27\,\xi\,\nu - \frac{81}{2}\,\nu^2 \\ 9 - \frac{45}{2}\,\xi - 45\,\nu + \frac{27}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{81}{2}\,\nu^2 \\ -\frac{45}{2}\,\xi + 27\,\xi^2 + 27\,\xi\,\nu \\ +\frac{9}{2}\,\xi - \frac{27}{2}\,\xi^2 \\ 27\,\xi - 27\xi^2 - 54\,\xi\,\nu \end{pmatrix}. \tag{69}$$

Thus find on the standard triangle Ω

$$\left(\frac{\partial u}{\partial \xi}\,,\,\frac{\partial u}{\partial \nu}\right) = (u_1, u_2, \ldots, u_{10}) \cdot \begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix} = \vec{u}^T \cdot \begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix}.$$

For a function $\varphi(x,y) = \sum_{i=1}^{10} \varphi_i\ \Phi_i(\xi(x,y)\,,\,\nu(x,y))$ use the above to conclude

$$\begin{pmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} +y_3 - y_1 & -y_2 + y_1 \\ -x_3 + x_1 & +x_2 - x_1 \end{bmatrix} \cdot \begin{bmatrix} \vec{\Phi}_\xi^T \\ \vec{\Phi}_\nu^T \end{bmatrix} \cdot \vec{\varphi}$$

or spelled out for the two components independently

$$\begin{aligned} \frac{\partial \varphi}{\partial x} &= \frac{1}{\det(\mathbf{T})} \left[\ (+y_3 - y_1)\,\vec{\Phi}_\xi^T + (-y_2 + y_1)\,\vec{\Phi}_\nu^T\ \right] \cdot \vec{\varphi}, \\ \frac{\partial \varphi}{\partial y} &= \frac{1}{\det(\mathbf{T})} \left[\ (-x_3 + x_1)\,\vec{\Phi}_\xi^T + (+x_2 - x_1)\,\vec{\Phi}_\nu^T\ \right] \cdot \vec{\varphi}. \end{aligned}$$

For the numerical integration use the values of the gradient at the Gauss integration points $\vec{g}_j = (\xi_j, \nu_j)$. Using expression (65) the values of the function φ at the Gauss points can be computed with the help of the interpolation matrix $\mathbf{M}$.

$$\begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{10} \end{pmatrix}$$

Similarly, using (68) and (69), define the interpolation matrices for the partial derivatives.

$$\frac{\partial}{\partial\xi}\begin{pmatrix}\varphi(\vec{g}_1)\\ \varphi(\vec{g}_2)\\ \vdots\\ \varphi(\vec{g}_7)\end{pmatrix} = \mathbf{M}_\xi \cdot \begin{pmatrix}\varphi_1\\ \varphi_2\\ \vdots\\ \varphi_{10}\end{pmatrix} \quad \text{and} \quad \frac{\partial}{\partial\nu}\begin{pmatrix}\varphi(\vec{g}_1)\\ \varphi(\vec{g}_2)\\ \vdots\\ \varphi(\vec{g}_7)\end{pmatrix} = \mathbf{M}_\nu \cdot \begin{pmatrix}\varphi_1\\ \varphi_2\\ \vdots\\ \varphi_{10}\end{pmatrix}. \tag{70}$$

Approximate values are

$$\mathbf{M}_\xi \approx \begin{bmatrix} -2.408 & 0.227 & 0 & -0.179 & -0.317 & 0.317 & -1.725 & 3.271 & -1.090 & 1.904\\ -0.227 & 2.408 & 0 & 1.725 & -0.317 & 0.317 & 0.179 & 1.090 & -3.271 & -1.904\\ -0.227 & 0.227 & 0 & -1.408 & 4.996 & -4.996 & 1.408 & -0.138 & 0.138 & 0\\ -0.511 & -0.247 & 0 & 3.852 & 0.868 & -0.868 & 1.358 & 1.137 & -0.379 & -5.210\\ 0.247 & 0.511 & 0 & -1.358 & 0.868 & -0.868 & -3.852 & 0.379 & -1.137 & 5.210\\ 0.247 & -0.247 & 0 & 0.489 & -0.221 & 0.221 & -0.489 & -2.984 & 2.984 & 0\\ 0.500 & -0.500 & 0 & 1.500 & 0 & 0 & -1.500 & -1.500 & 1.500 & 0 \end{bmatrix}$$

and

$$\mathbf{M}_\nu \approx \begin{bmatrix} -2.269 & 0 & 0.227 & -0.317 & -0.179 & -1.090 & 3.271 & -1.725 & 0.317 & 1.904\\ 0.863 & 0 & 0.227 & 4.996 & -1.408 & 0.138 & -0.138 & 1.408 & -4.996 & 0\\ 0.863 & 0 & 2.408 & -0.317 & 1.725 & -3.271 & 1.090 & 0.179 & 0.317 & -1.904\\ 2.473 & 0 & -0.247 & 0.868 & 3.852 & -0.379 & 1.137 & 1.358 & -0.868 & -5.210\\ 0.626 & 0 & -0.247 & -0.221 & 0.489 & 2.984 & -2.984 & -0.489 & 0.221 & 0\\ 0.626 & 0 & 0.511 & 0.868 & -1.358 & -1.137 & 0.379 & -3.852 & -0.868 & 5.210\\ 2.000 & 0 & -0.500 & 0 & 1.500 & 1.500 & -1.500 & -1.500 & 0 & 0 \end{bmatrix}.$$

The matrices $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ allow to compute the values of the partial derivatives at the Gauss points in the standard triangle Ω and they are independent on the general triangle E.

Combining the above two computations use the notation

$$\vec{x}_i = \begin{pmatrix}x_1\\ y_1\end{pmatrix} + \mathbf{T}\cdot\begin{pmatrix}\xi_i\\ \nu_i\end{pmatrix} \quad \text{for} \quad i = 1,2,3,\ldots,7$$

and find for the first component $\varphi_x = \frac{\partial\varphi}{\partial x}$ of the gradient at the Gauss points

$$\begin{pmatrix}\varphi_x(\vec{x}_1)\\ \varphi_x(\vec{x}_2)\\ \vdots\\ \varphi_x(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\left[\ (+y_3-y_1)\,\mathbf{M}_\xi^T + (-y_2+y_1)\,\mathbf{M}_\nu^T\ \right]\cdot\vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix}\varphi_y(\vec{x}_1)\\ \varphi_y(\vec{x}_2)\\ \vdots\\ \varphi_y(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\left[\ (-x_3+x_1)\,\mathbf{M}_\xi^T + (+x_2-x_1)\,\mathbf{M}_\nu^T\ \right]\cdot\vec{\phi}.$$

The above results for $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ can be coded in *Octave* and then used to compute the element stiffness matrix.

6.6.5 Integration of $u\,\vec{b}\cdot\nabla\phi$ and $a\,\nabla u\cdot\nabla\phi$

The vector function $\vec{b}(\vec{x})$ has to be evaluated at the Gauss integration points $\vec{g}_j$. Then the integration of

$$\iint\limits_E u\,\vec{b}\,\nabla\phi\;dA = \iint\limits_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA + \iint\limits_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA$$

is approximated by

$$\begin{aligned}
\iint\limits_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA &\approx \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\left((y_3-y_1)\,\mathbf{M}_\xi^T+(-y_2+y_1)\,\mathbf{M}_\nu^T\right)\cdot\mathrm{diag}(\overrightarrow{wb_1})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle\\
\iint\limits_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA &\approx \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\left((-x_3+x_1)\,\mathbf{M}_\xi^T+(x_2-x_1)\,\mathbf{M}_\nu^T\right)\cdot\mathrm{diag}(\overrightarrow{wb_2})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle\,.
\end{aligned}$$

The function $a\,\nabla u\cdot\nabla\phi = a\,(\frac{\partial\,u}{\partial x}\frac{\partial\,\phi}{\partial x}+\frac{\partial\,u}{\partial y}\frac{\partial\,\phi}{\partial y})$ has to be evaluated at the Gauss integration points $\vec{g}_j$, then multiplied by the Gauss weights w_i and added up. Use the vector $\overrightarrow{wa}$ with the values of the function $a(x_i,y_i)$ and the weights w_i at the Gauss points to obtain

$$\begin{aligned}
\iint\limits_E a\,\frac{\partial\,u(\vec{x})}{\partial x}\,\frac{\partial\,\phi(\vec{x})}{\partial x}\;dA &= |\det\mathbf{T}|\iint\limits_\Omega a(\vec{x}(\xi,\nu))\frac{\partial\,u(\vec{x}(\xi,\nu))}{\partial x}\,\frac{\partial\,\phi(\vec{x}(\xi,\nu))}{\partial x}\;d\xi\,d\nu\\
&\approx \frac{|\det\mathbf{T}|}{(\det\mathbf{T})^2}\,\langle\mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det\mathbf{T}|}\,\langle\mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle\\
\iint\limits_E a\,\frac{\partial\,u(\vec{x})}{\partial y}\,\frac{\partial\,\phi(\vec{x})}{\partial y}\;dA &= |\det\mathbf{T}|\iint\limits_\Omega a(\vec{x}(\xi,\nu))\frac{\partial\,u(\vec{x}(\xi,\nu))}{\partial y}\,\frac{\partial\,\phi(\vec{x}(\xi,\nu))}{\partial y}\;d\xi\,d\nu\\
&\approx \frac{|\det\mathbf{T}|}{(\det\mathbf{T})^2}\,\langle\mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det\mathbf{T}|}\,\langle\mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle\,,
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_x &= \left[\;(+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\;\right]^T\cdot\mathrm{diag}(\overrightarrow{wa})\cdot\left[\;(+y_3-y_1)\,\mathbf{M}_\xi+(-y_2+y_1)\,\mathbf{M}_\nu\;\right]\\
\mathbf{A}_y &= \left[\;(-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\;\right]^T\cdot\mathrm{diag}(\overrightarrow{wa})\cdot\left[\;(-x_3+x_1)\,\mathbf{M}_\xi+(+x_2-x_1)\,\mathbf{M}_\nu\;\right].
\end{aligned}$$

6.6.6 Partial derivatives at the nodes

For post processing one also needs the partial derivatives of the function at the nodes. On the standard triangle Ω use the formulas for the partial derivatives of the basis functions in expressions (68) and (69) to find them at the nodes, given by the (ξ,ν) coordinates in Table 19 for cubic elements.

$$\begin{pmatrix}
\varphi_\xi(\xi_1,\nu_1)\\ \varphi_\xi(\xi_2,\nu_2)\\ \varphi_\xi(\xi_3,\nu_3)\\ \varphi_\xi(\xi_4,\nu_4)\\ \varphi_\xi(\xi_5,\nu_5)\\ \varphi_\xi(\xi_6,\nu_6)\\ \varphi_\xi(\xi_7,\nu_7)\\ \varphi_\xi(\xi_8,\nu_8)\\ \varphi_\xi(\xi_9,\nu_9)\\ \varphi_\xi(\xi_{10},\nu_{10})
\end{pmatrix} =
\begin{bmatrix}
\frac{-11}{2} & 1 & 0 & 0 & 0 & 0 & 0 & 9 & \frac{-9}{2} & 0\\
-1 & \frac{11}{2} & 0 & 0 & 0 & 0 & 0 & \frac{9}{2} & -9 & 0\\
-1 & 1 & 0 & \frac{-9}{2} & 9 & -9 & \frac{9}{2} & 0 & 0 & 0\\
-1 & 1 & 0 & \frac{9}{2} & 0 & 0 & \frac{3}{2} & 3 & -3 & -6\\
-1 & \frac{-1}{2} & 0 & 3 & 3 & -3 & 3 & \frac{3}{2} & 0 & -6\\
\frac{1}{2} & 1 & 0 & -3 & 3 & -3 & -3 & 0 & \frac{-3}{2} & 6\\
-1 & 1 & 0 & \frac{-3}{2} & 0 & 0 & \frac{-9}{2} & 3 & -3 & 6\\
-1 & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{-3}{2} & 3 & 0\\
\frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 & -3 & \frac{3}{2} & 0\\
\frac{1}{2} & \frac{-1}{2} & 0 & \frac{3}{2} & 0 & 0 & \frac{-3}{2} & \frac{-3}{2} & \frac{3}{2} & 0
\end{bmatrix}
\begin{pmatrix}
\varphi_1\\ \varphi_2\\ \varphi_3\\ \varphi_4\\ \varphi_5\\ \varphi_6\\ \varphi_7\\ \varphi_8\\ \varphi_9\\ \varphi_{10}
\end{pmatrix} = \mathbf{N}_\xi
\begin{pmatrix}
\varphi_1\\ \varphi_2\\ \varphi_3\\ \varphi_4\\ \varphi_5\\ \varphi_6\\ \varphi_7\\ \varphi_8\\ \varphi_9\\ \varphi_{10}
\end{pmatrix}$$

and

$$\begin{pmatrix} \varphi_\nu(\xi_1,\nu_1) \\ \varphi_\nu(\xi_2,\nu_2) \\ \varphi_\nu(\xi_3,\nu_3) \\ \varphi_\nu(\xi_4,\nu_4) \\ \varphi_\nu(\xi_5,\nu_5) \\ \varphi_\nu(\xi_6,\nu_6) \\ \varphi_\nu(\xi_7,\nu_7) \\ \varphi_\nu(\xi_8,\nu_8) \\ \varphi_\nu(\xi_9,\nu_9) \\ \varphi_\nu(\xi_{10},\nu_{10}) \end{pmatrix} = \begin{bmatrix} \frac{-11}{2} & 0 & 1 & 0 & 0 & \frac{-9}{2} & 9 & 0 & 0 & 0 \\ -1 & 0 & 1 & 9 & \frac{-9}{2} & 0 & 0 & \frac{9}{2} & -9 & 0 \\ -1 & 0 & \frac{11}{2} & 0 & 0 & -9 & \frac{9}{2} & 0 & 0 & 0 \\ -1 & 0 & \frac{-1}{2} & 3 & 3 & 0 & \frac{3}{2} & 3 & -3 & -6 \\ -1 & 0 & 1 & 0 & \frac{9}{2} & -3 & 3 & \frac{3}{2} & 0 & -6 \\ \frac{1}{2} & 0 & 1 & 0 & 0 & \frac{3}{2} & -3 & 0 & 0 & 0 \\ -1 & 0 & \frac{-1}{2} & 0 & 0 & 3 & \frac{-3}{2} & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & \frac{-3}{2} & -3 & 3 & \frac{-9}{2} & 0 & 6 \\ \frac{1}{2} & 0 & 1 & 3 & -3 & \frac{-3}{2} & 0 & -3 & -3 & 6 \\ \frac{1}{2} & 0 & \frac{-1}{2} & 0 & \frac{3}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{-3}{2} & 0 & 0 \end{bmatrix} \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \\ \varphi_7 \\ \varphi_8 \\ \varphi_9 \\ \varphi_{10} \end{pmatrix} = \mathbf{N}_\nu \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \\ \varphi_7 \\ \varphi_8 \\ \varphi_9 \\ \varphi_{10} \end{pmatrix}$$

Now use the transformation formulas (60) and (61) to determine the gradient of a function on the general triangle

$$\varphi(x,y) = \sum_{i=1}^{10} \varphi_i \, \Phi_i(\xi(x,y),\nu(x,y))$$

at the nodes (x_i, y_i) in the general triangle E, leading to

$$\begin{pmatrix} \varphi_x(x_1,y_1) \\ \varphi_x(x_2,y_2) \\ \vdots \\ \varphi_x(x_{10},y_{10}) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (+y_3 - y_1)\, \mathbf{N}_\xi^T + (-y_2 + y_1)\, \mathbf{N}_\nu^T \, \right] \cdot \vec{\varphi}\,,$$

$$\begin{pmatrix} \varphi_y(x_1,y_1) \\ \varphi_y(x_2,y_2) \\ \vdots \\ \varphi_y(x_{10},y_{10}) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (-x_3 + x_1)\, \mathbf{N}_\xi^T + (+x_2 - x_1)\, \mathbf{N}_\nu^T \, \right] \cdot \vec{\varphi}\,.$$

These results are useful to evaluate the gradient at the nodes. Observe that the results depends on the triangle used for the interpolation and a node is typically member of more than one triangle.

6.6.7 Integration over boundary segments

In expression (46) integrals over the section Γ_2 of the boundary are required.

$$\int_{\Gamma_2} \phi \;\; (g_2 + g_3\, u) \;\; ds$$

For triangular domains the boundary consists of straight line segments. Thus replace the integral by a sum of line integrals and use a Gauss integration. Based on the two endpoints $\vec{x}_1$ and $\vec{x}_3$ and the midpoint $\vec{x}_2 = \frac{1}{2}\,(\vec{x}_1 + \vec{x}_3)$ use the values at three Gauss integration points.

Based on

$$\begin{aligned} \int_{-h/2}^{h/2} f(x)\, dx &\approx \frac{h}{18} \left(5\, f(-\frac{\sqrt{3}}{2\sqrt{5}}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{3}}{2\sqrt{5}}\, h) \right) \\ &= \frac{h}{18} \left(5\, f(-\frac{\sqrt{15}}{10}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{15}}{10}\, h) \right) \end{aligned}$$

polynomials up to degree 5 are integrated exactly, thus the error on one interval is proportional to h^7. To evaluate a function at the Gauss points

$$\begin{aligned} \vec{p}_1 &= \tfrac{1}{2}\,(\vec{x}_1 + \vec{x}_4) - \tfrac{\sqrt{3}}{2\sqrt{5}}\;(\vec{x}_4 - \vec{x}_1) \\ \vec{p}_2 &= \tfrac{1}{2}\,(\vec{x}_1 + \vec{x}_4) \\ \vec{p}_3 &= \tfrac{1}{2}\,(\vec{x}_1 + \vec{x}_4) + \tfrac{\sqrt{3}}{2\sqrt{5}}\;(\vec{x}_4 - \vec{x}_1) \end{aligned}$$

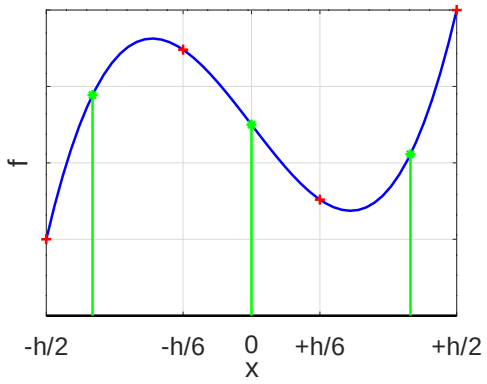


Figure 99: The interpolation from four nodes to three Gauss points on an interval $[-\frac{h}{2}, +\frac{h}{2}]$

use a cubic interpolation of a function with $f_{-2} = f(-h/2)$, $f_{-1} = f(-h/6)$, $f_{+1} = f(+h/6)$ and $f_{+2} = f(+h/2)$. Required are the values at $x = 0$ and $x = \pm\frac{\sqrt{15}}{10}\, h \approx \pm 0.387\, h$. This is illustrated in Figure 99 with the values of the function $f(x)$ indicated by red spots and the interpolation position and values in green. The computations are tedious shown at the end of this section in Subsection 6.6.8 and lead to

$$\begin{pmatrix} u(\vec{p}_1) \\ u(\vec{p}_2) \\ u(\vec{p}_3) \end{pmatrix} = \mathbf{M}_B \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

With the length $L = \sqrt{(x_4 - x_1)^2 + (y_4 - y_1)^2}$ of the segment this leads to the approximations

$$\begin{aligned}
\int_{\text{edge}} \phi\, g_2\, ds &\approx \frac{L}{18} \langle \mathbf{M}_B \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}, \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle = \frac{L}{18} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}, \mathbf{M}_B^T \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle \\
\int_{\text{edge}} \phi\, g_3\, u\, ds &\approx \frac{L}{18} \langle \mathbf{M}_B \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}, \begin{bmatrix} 5\, g_3(\vec{p}_1) & 0 & 0 \\ 0 & 8\, g_3(\vec{p}_2) & 0 \\ 0 & 0 & 5\, g_3(\vec{p}_3) \end{bmatrix} \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} \rangle \\
&= \frac{L}{18} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}, \mathbf{M}_B^T \begin{bmatrix} 5\, g_3(\vec{p}_1) & 0 & 0 \\ 0 & 8\, g_3(\vec{p}_2) & 0 \\ 0 & 0 & 5\, g_3(\vec{p}_3) \end{bmatrix} \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} \rangle .
\end{aligned}$$

The first expression will lead to a contribution to the RHS vector of the linear system to be solved, while the second expression will lead to entries in the matrix. These approximate integrations lead to the exact result if the function to be integrated is a polynomial of degree 5, or less. If h is the typical length of an edge then the error is of the order h^7 for one line segment and thus of order h^6 for the total boundary. This boundary integration is used for third order elements. The second expression is of the form

$$\int \phi\, g_3\, u\, ds \approx \langle \vec{\phi}, \mathbf{B}\, \vec{u} \rangle = \langle \begin{pmatrix} \phi_2 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}, \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{34} & b_{44} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} \rangle$$

and its effect on the linear system $\mathbf{A}\,\vec{u} + \mathbf{W}\,\vec{f} = \vec{0}$ to be solved depends on nodes being on the Dirichlet section of the boundary or the Neumann section.

- If u_1 and u_4 are free, i.e. not on the Dirichlet section, then u_2 and u_3 are free too. All entries of the matrix $\mathbf{B}$ have to be added to the global stiffness matrix $\mathbf{A}$.
- If u_1 and u_4 are on the Dirichlet section, then u_2 and u_3 are on the Dirichlet section too. Nothing has to be added to $\mathbf{A}$ and $\vec{f}$.
- If u_1, u_2 and u_3 are free and u_4 is on the Dirichlet section, then only the first three expressions

$$\begin{aligned} b_{11}\,u_1 + b_{12}\,u_2 + b_{13}\,u_3 + b_{14}\,u_4 &= b_{11}\,u_1 + b_{12}\,u_2 + b_{13}\,u_3 + b_{14}\,d_4 \\ b_{21}\,u_1 + b_{22}\,u_2 + b_{23}\,u_3 + b_{24}\,u_4 &= b_{21}\,u_1 + b_{22}\,u_2 + b_{23}\,u_3 + b_{24}\,d_4 \\ b_{31}\,u_1 + b_{32}\,u_2 + b_{33}\,u_3 + b_{34}\,u_4 &= b_{31}\,u_1 + b_{32}\,u_2 + b_{33}\,u_3 + b_{34}\,d_4 \end{aligned}$$

  have to be taken into account. d_4 is the Dirichlet value at the position of u_4. The contributions $b_{14}\,d_4$, $b_{24}\,d_4$ and $b_{34}\,d_4$ have to be added to $\mathbf{W}\,\vec{f}$, the other expressions to $\mathbf{A}$.
- If u_2, u_3 and u_4 are free and u_1 is on the Dirichlet section, then only the least three expressions

$$\begin{aligned} b_{21}\,u_1 + b_{22}\,u_2 + b_{23}\,u_3 + b_{24}\,u_4 &= b_{21}\,d_1 + b_{22}\,u_2 + b_{23}\,u_3 + b_{24}\,u_4 \\ b_{31}\,u_1 + b_{32}\,u_2 + b_{33}\,u_3 + b_{34}\,u_4 &= b_{31}\,d_1 + b_{32}\,u_2 + b_{33}\,u_3 + b_{34}\,u_4 \\ b_{41}\,u_1 + b_{42}\,u_2 + b_{43}\,u_3 + b_{44}\,u_4 &= b_{41}\,d_1 + b_{42}\,u_2 + b_{43}\,u_3 + b_{44}\,u_4 \end{aligned}$$

  have to be taken into account. d_1 is the Dirichlet value at the position of u_1. The contributions $b_{21}\,d_1$, $b_{31}\,d_1$ and $b_{41}\,d_1$ have to be added to $\mathbf{W}\,\vec{f}$, the other expressions to $\mathbf{A}$.

6.6.8 From a polynomial interpolation to the Gauss integration points on an interval

For an interval $[-h/2, +h/2]$ use a polynomial $p(x) = c_0 + c_1\,x + c_2\,x^2 + c_3\,x^3$, leading to

$$\begin{aligned} f_{-2} = p(-h/2) &= c_0 - \frac{1}{2}\,h\,c_1 + \frac{1}{4}\,h^2\,c_2 - \frac{1}{8}\,h^3\,c_3 \\ f_{-1} = p(-h/6) &= c_0 - \frac{1}{6}\,h\,c_1 + \frac{1}{36}\,h^2\,c_2 - \frac{1}{216}\,h^3\,c_3 \\ f_{+1} = p(+h/6) &= c_0 + \frac{1}{6}\,h\,c_1 + \frac{1}{36}\,h^2\,c_2 + \frac{1}{216}\,h^3\,c_3 \\ f_{+2} = p(+h/2) &= c_0 + \frac{1}{2}\,h\,c_1 + \frac{1}{4}\,h^2\,c_2 + \frac{1}{8}\,h^3\,c_3 \end{aligned}$$

or with a matrix notation

$$\begin{bmatrix} +1 & -\frac{1}{2} & +\frac{1}{4} & -\frac{1}{8} \\ +1 & -\frac{1}{6} & +\frac{1}{36} & -\frac{1}{216} \\ +1 & +\frac{1}{6} & +\frac{1}{36} & +\frac{1}{216} \\ +1 & +\frac{1}{2} & +\frac{1}{4} & +\frac{1}{8} \end{bmatrix} \begin{pmatrix} c_0 \\ h\,c_1 \\ h^2\,c_2 \\ h^3\,c_3 \end{pmatrix} = \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} .$$

The corresponding inverse matrix leads to

$$\begin{pmatrix} c_0 \\ h\,c_1 \\ h^2\,c_2 \\ h^3\,c_3 \end{pmatrix} = \frac{1}{16} \begin{bmatrix} -1 & +9 & +9 & -1 \\ +2 & -54 & +54 & -2 \\ +36 & -36 & -36 & +36 \\ -72 & +216 & -216 & +72 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} .$$

With $\lambda = \frac{\sqrt{15}}{10} \approx 0.3873$ and $p(\lambda\,h) = c_0 + \lambda\,c_1\,h + \lambda^2\,c_2\,h^2 + \lambda^3\,c_3\,h^3$ obtain

$$p(\lambda\,h) = \frac{1}{16} \begin{bmatrix} 1 & \lambda & \lambda^2 & \lambda^3 \end{bmatrix} \begin{bmatrix} -1 & +9 & +9 & -1 \\ +2 & -54 & +54 & -2 \\ +36 & -36 & -36 & +36 \\ -72 & +216 & -216 & +72 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

$$\begin{pmatrix} p(-\lambda\, h) \\ p(0) \\ p(+\lambda\, h) \end{pmatrix} = \frac{1}{16} \begin{bmatrix} 1 & -\lambda & \lambda^2 & -\lambda^3 \\ 1 & 0 & 0 & 0 \\ 1 & +\lambda & \lambda^2 & +\lambda^3 \end{bmatrix} \begin{bmatrix} -1 & +9 & +9 & -1 \\ +2 & -54 & +54 & -2 \\ +36 & -36 & -36 & +36 \\ -72 & +216 & -216 & +72 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

$$= \mathbf{M}_B \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}.$$

This matrix $\mathbf{M}_B$ is used on the segments of the boundary to determine the values at the Gauss integration points, given the values of the four nodes of a third order element.

6.7 Construction of Hermite elements

In this section a Hermite element on triangles will be constructed.

- The Hermite element is shown in Section 6.2. Among the elements use in these notes it is the only C^1 quasi conforming element.
- As basis all polynomial up to degree 3 are used, i.e. 10 DOF are required.
- As DOF use the values and first derivatives at the three corners, supplemented with the value at the centroid of the triangle. This is illustrated in Figure 100.
- The interpolated values will be continuous across element boundaries, but the first order derivatives may show jumps. At the corners of the triangles the first order derivatives coincide.

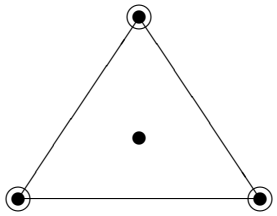


Figure 100: The Hermite triangle

Hermite elements are used in Sections 5.6 and 9.5. Due to the problematic implementation of the boundary conditions (see Section 6.7.7) Hermite elements are currently only used for elliptic problems, i.e. for the commands `BVP2D()` and `BVP2Dsym()`.

6.7.1 The Hermite triangle

The selection of DOF in these notes is:

- First the values at the three corners of the triangular element.
- Then the partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ at the three corners.
- Then the value at the centroid of the triangle.

This leads to the following conditions for the ten functions.

1. $u(x_1, y_1) = 1$ and $u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
2. $u(x_2, y_2) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$

3. $u(x_3, y_3) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
4. $u_x(x_1, y_1) = 1$ and $u(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
5. $u_y(x_1, y_1) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
6. $u_x(x_2, y_2) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
7. $u_y(x_2, y_2) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
8. $u_x(x_3, y_3) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_y(x_3, y_3) = u(x_c, y_c) = 0$
9. $u_y(x_3, y_3) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u(x_c, y_c) = 0$
10. $u(x_c, y_c) = 1$ and $u(x_1, y_1) = u_x(x_1, y_1) = u_y(x_1, y_1) = u(x_2, y_2) = u_x(x_2, y_2) = u_y(x_2, y_2) = u(x_3, y_3) = u_x(x_3, y_3) = u_y(x_3, y_3) = 0$

For each of those DOF there is exactly one polynomial of degree 3 satisfying the conditions. In Figure 101 find the graphs and contour plots for these ten functions on an equilateral triangle.

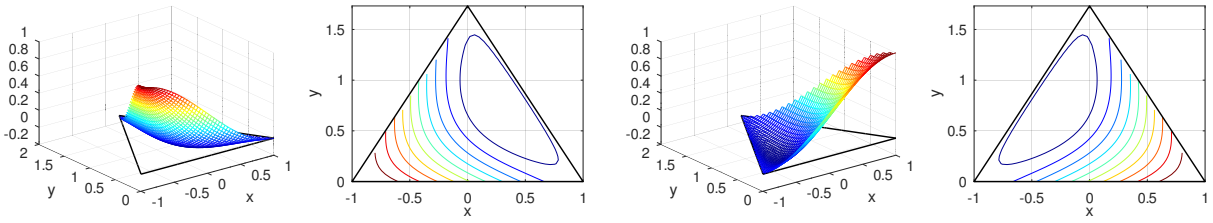
These functions have a few special properties:

- If a DOF is nonzero at a corner (value or one of the partial derivatives), then the values of the DOF function along the opposing edge vanish and both partial derivatives at the other corner of the triangle vanish. The normal derivative along the opposing edge does not necessarily vanish.
- For the tenth DOF function the values along all edges of the triangle vanish, as do the partial derivatives at the corners. The normal derivaties along the edges do not vanish.
- Observe that the partial derivatives are specified in the global xy coordinate system, i.e. not directional derivatives in the direction of the edge of the triangle.

6.7.2 The polynomial basis functions

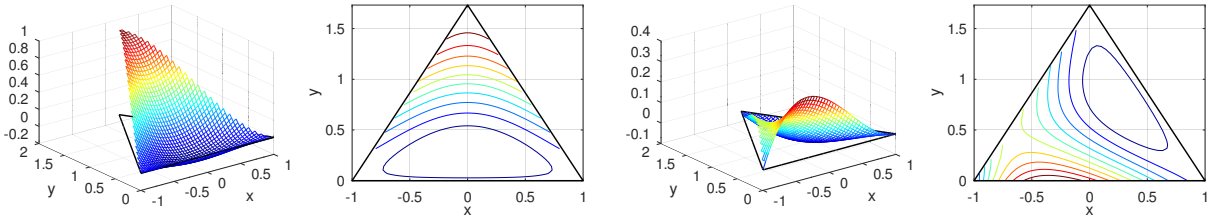
On a general triangle with the corners at (x_1, y_1), (x_2, y_2) and (x_3, y_3) use all polynomials up to degree 3 as basis functions. With this choice (instead of the above DOF functions) it is easier to assure matching values of the partial derivatives with respect to x and y at the nodes of the mesh. To avoid accuracy problems use the centroid $(x_c, y_c) = \frac{1}{3}(x_1 + x_2 + x_3\,,\; y_1 + y_2 + y_3)$ and then the shifted monomials 1, $(x - x_c)$, $(y - y_c)$, $(x - x_c)^2$, …,$(y - y_c)^3$. This leads to explicit expressions for the function u and the partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ on the given triangle.

$$\begin{aligned} u(x,y) &= \sum_{i=1}^{10} b_i\, p_i(x,y) \\ &= b_1\, 1 + b_2\,(x - x_c) + b_3\,(y - y_c) + b_4\,(x - x_c)^2 + b_5\,(x - x_c)\,(y - y_c) + b_6\,(y - y_c)^2 + \\ &\quad + b_7\,(x - x_c)^3 + b_8\,(x - x_c)^2\,(y - y_c) + b_9\,(x - x_c)\,(y - y_c)^2 + b_{10}\,(y - y_c)^3 \end{aligned}$$



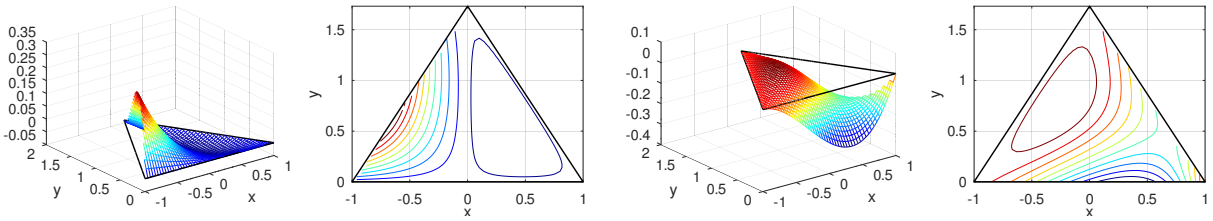
(a) first DOF function, $u(x_1, y_1) = 1$

(b) second DOF function, $u(x_2, y_2) = 1$



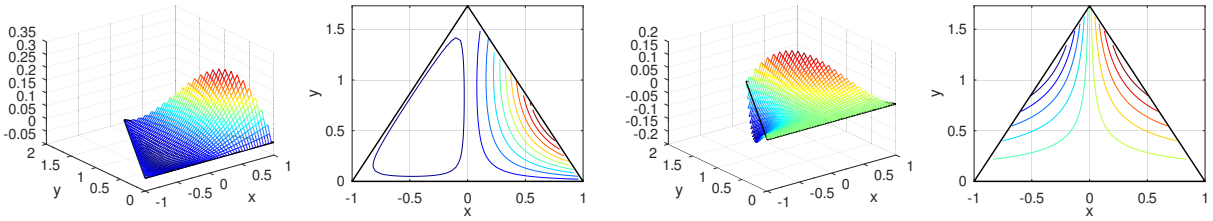
(c) third DOF function, $u(x_3, y_3) = 1$

(d) fourth DOF function, $u_x(x_1, y_1) = 1$



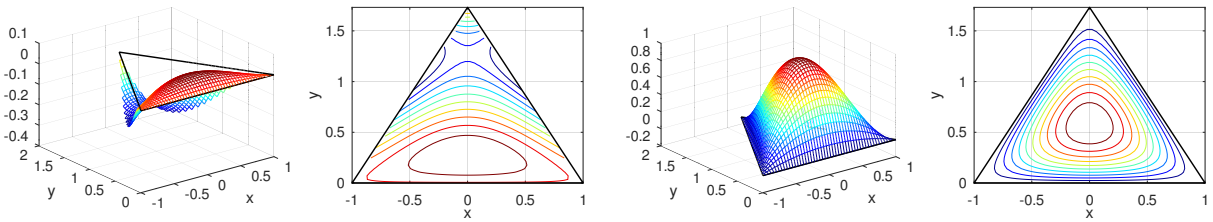
(e) fifth DOF function, $u_y(x_1, y_1) = 1$

(f) sixth DOF function, $u_x(x_2, y_2) = 1$



(g) seventh DOF function, $u_y(x_2, y_2) = 1$

(h) eighth DOF function, $u_x(x_3, y_3) = 1$



(i) nineth DOF function, $u_y(x_3, y_3) = 1$

(j) tenth DOF function, $u(x_c, y_c) = 1$

Figure 101: The ten functions related to the ten DOF of a Hermite element

$$= \left\langle \begin{pmatrix} 1 \\ (x-x_c) \\ (y-y_c) \\ (x-x_c)^2 \\ (x-x_c)(y-y_c) \\ (y-y_c)^2 \\ (x-x_c)^3 \\ (x-x_c)^2(y-y_c) \\ (x-x_c)(y-y_c)^2 \\ (y-y_c)^3 \end{pmatrix}, \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \end{pmatrix} \right\rangle =: \langle \vec{B}(x,y), \vec{b} \rangle \tag{71}$$

$$\frac{\partial}{\partial x} u(x,y) = \sum_{i=1}^{10} b_i \frac{\partial}{\partial x} p_i(x,y)$$

$$= b_2 + b_4\, 2\,(x-x_c) + b_5\,(y-y_c) + b_7\, 3\,(x-x_c)^2 + b_8\, 2\,(x-x_c)(y-y_c) + b_9\,(y-y_c)^2$$

$$= \left\langle \begin{pmatrix} 0 \\ 1 \\ 0 \\ 2\,(x-x_c) \\ (y-y_c) \\ 0 \\ 3\,(x-x_c)^2 \\ 2\,(x-x_c)(y-y_c) \\ (y-y_c)^2 \\ 0 \end{pmatrix}, \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \end{pmatrix} \right\rangle =: \langle \vec{B}_x(x,y), \vec{b} \rangle \tag{72}$$

$$\frac{\partial}{\partial y} u(x,y) = \sum_{i=1}^{10} b_i \frac{\partial}{\partial y} p_i(x,y)$$

$$= b_3 + b_5\,(x-x_c) + b_6\, 2\,(y-y_c) + b_8\,(x-x_c)^2 + b_9\, 2\,(x-x_c)(y-y_c) + b_{10}\, 3\,(y-y_c)^2 \tag{73}$$

$$= \left\langle \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ (x-x_c) \\ 2\,(y-y_c) \\ 0 \\ (x-x_c)^2 \\ 2\,(x-x_c)(y-y_c) \\ 3\,y^2 \end{pmatrix}, \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \end{pmatrix} \right\rangle =: \langle \vec{B}_y(x,y), \vec{b} \rangle \tag{74}$$

With the above functions there are two ways to represent the polynomial of degree 3 on the given triangle:

- The coefficient vector $\vec{b} \in \mathbb{R}^{10}$ and the sum of ten monomials $u(x,y) = \sum_{i=1}^{10} b_i\, p_i(x,y)$, shown in equation (71).
- The coefficient vector $\vec{c} \in \mathbb{R}^{10}$ related to the FEM basis function in Figure 101 and the corresponding sum.

In Table 20 find the expressions usefull to determine $\vec{c}$ (DOF) for given values of $\vec{b}$, the coefficients of the monomials. Use the notations $\Delta x_j = (x_j - x_c)$ and $\Delta y_j = (y_j - y_c)$. With this 10×10 matrix **BF2DOF** evaluate the 10 DOF $\vec{c}$ for the 10 basis functions. Thus for the coefficient vector $\vec{b} \in \mathbb{R}^{10}$ in equation (71) the product

$$\vec{c} = \mathbf{BF2DOF} \cdot \vec{b}$$

will determine the values $\vec{c}$ of the 10 DOFs. Thus the inverse matrix **DOF2BF** allows to determine the decomposition of the general function as the sum in equation (71) of the basic monomials.

$$\vec{b} = \mathbf{DOF2BF} \cdot \vec{c} = \mathbf{BF2DOF}^{-1} \cdot \vec{c}$$

This is implemented in a function `fDOF2BF()` with the corners of the triangle as input.

polynom	1	Δx	Δy	Δx^2	$\Delta x\,\Delta y$	Δy^2	Δx^3	$\Delta x^2\Delta y$	$\Delta x\,\Delta y^2$	Δy^3
$c_1 = u(x_1, y_1)$	1	Δx_1	Δy_1	Δx_1^2	$\Delta x_1\,\Delta y_1$	Δy_1^2	Δx_1^3	$\Delta x_1^2\Delta y_1$	$\Delta x_1\,\Delta y_1^2$	Δy_1^3
$c_2 = u(x_2, y_2)$	1	Δx_2	Δy_2	Δx_2^2	$\Delta x_2\,\Delta y_2$	Δy_2^2	Δx_2^3	$\Delta x_2^2\Delta y_2$	$\Delta x_2\,\Delta y_2^2$	Δy_2^3
$c_3 = u(x_3, y_3)$	1	Δx_3	Δy_3	Δx_3^2	$\Delta x_3\,\Delta y_3$	Δy_3^2	Δx_3^3	$\Delta x_3^2\Delta y_3$	$\Delta x_3\,\Delta y_3^2$	Δy_3^3
$c_4 = u_x(x_1, y_1)$	0	1	0	$2\,\Delta x_1$	Δy_1	0	$3\,\Delta x_1^2$	$2\,\Delta x_1\,\Delta y_1$	Δy_1^2	0
$c_5 = u_y(x_1, y_1)$	0	0	1	0	Δx_1	$2\,\Delta y_1$	0	Δx_1^2	$2\,\Delta x_1\,\Delta y_1$	$3\,\Delta y_1^2$
$c_6 = u_x(x_2, y_2)$	0	1	0	$2\,\Delta x_2$	Δy_2	0	$3\,\Delta x_2^2$	$2\,\Delta x_2\,\Delta y_2$	Δy_2^2	0
$c_7 = u_y(x_2, y_2)$	0	0	1	0	Δx_2	$2\,\Delta y_2$	0	Δx_2^2	$2\,\Delta x_2\,\Delta y_2$	$3\,\Delta y_2^2$
$c_8 = u_x(x_3, y_3)$	0	1	0	$2\,\Delta x_3$	Δy_3	0	$3\,\Delta x_3^2$	$2\,\Delta x_3\,\Delta y_3$	Δy_3^2	0
$c_9 = u_y(x_3, y_3)$	0	0	1	0	Δx_3	$2\,\Delta y_3$	0	Δx_3^2	$2\,\Delta x_3\,\Delta y_3$	$3\,\Delta y_3^2$
$c_{10} = u(x_c, y_c)$	1	0	0	0	0	0	0	0	0	0

Table 20: Entries in the matrix **BF2DOF** with the reduced polynomials in (71)

fDOF2BF.m

```
function DOF2BF = fDOF2BF(Tri)
  %% determine the matrix DOF2BF to convert the DOF (Degrees Of Freedom)
  %% to the coefficients for the BF (Basis Functions)
  x = Tri(:,1);   y = Tri(:,2);  mx = sum(x)/3; my = sum(y)/3;
  x = x-mx; y = y - my;
 BF2DOF = ...
    [1 x(1) y(1) x(1)^2 x(1)*y(1) y(1)^2 x(1)^3 x(1)^2*y(1) x(1)*y(1)^2 y(1)^3;
     1 x(2) y(2) x(2)^2 x(2)*y(2) y(2)^2 x(2)^3 x(2)^2*y(2) x(2)*y(2)^2 y(2)^3;
     1 x(3) y(3) x(3)^2 x(3)*y(3) y(3)^2 x(3)^3 x(3)^2*y(3) x(3)*y(3)^2 y(3)^3;
     0  1    0   2*x(1)   y(1)      0   3*x(1)^2 2*x(1)*y(1)    y(1)^2     0;
     0  0    1     0      x(1)    2*y(1)   0      x(1)^2   2*x(1)*y(1) 3*y(1)^2;
     0  1    0   2*x(2)   y(2)      0   3*x(2)^2 2*x(2)*y(2)    y(2)^2     0;
     0  0    1     0      x(2)    2*y(2)   0      x(2)^2   2*x(2)*y(2) 3*y(2)^2;
     0  1    0   2*x(3)   y(3)      0   3*x(3)^2 2*x(3)*y(3)    y(3)^2     0;
     0  0    1     0      x(3)    2*y(3)   0      x(3)^2   2*x(3)*y(3) 3*y(3)^2;
     1  0    0     0       0       0       0        0          0          0];
  DOF2BF = inv(BF2DOF);
endfunction
```

For the FEM algorithm the values of the function $u(\vec{g}_i)$ and the partial derivatives $\frac{\partial\, u(\vec{g}_i)}{\partial x}$ and $\frac{\partial\, u(\vec{g}_i)}{\partial x}$ at the Gauss integration points $\vec{g}_i$ are required. The function `fBF2uGP()` evaluates the location of the Gauss points and three matrices for the evaluation of the function. As input the cornes of the triangle have to be provided.

PG : The coordinates of the Gauss points.

uGPmat : Matrix to determine the values of the interpolated function at the Gauss points. If the polynomial is determined by the DOFs in $\vec{c} \in \mathbb{R}^{10}$ the values `uGP` of u at the Gauss points are computed by

```
uGP = uGPmat * DOF2BF * c;
```

duxGPmat : Matrix to determine the values of the partial derivative $\frac{\partial\, u}{\partial x}$ of the interpolated function at the Gauss points.

```
duxGP = duxGPmat * DOF2BF * c;
```

duyGPmat : Matrix to determine the values of the partial derivative $\frac{\partial u}{\partial y}$ of the interpolated function at the Gauss points.

```
duyGP = duyGPmat * DOF2BF * c;
```

fBF2uGP.m

```
function [GP,uGPmat,duxGPmat,duyGPmat] = fBF2uGPmat(Tri);
  %% determine the coordinates of the Gauss points
  %% and the matrices to determine u and its partial derivatives
  x = Tri(:,1); y = Tri(:,2);   mx = mean(x); my = mean(y);
  v1 = [(x(2)+x(3))/2-x(1)  (y(2)+y(3))/2-y(1)];
  v2 = [(x(3)+x(1))/2-x(2)  (y(3)+y(1))/2-y(2)];
  v3 = [(x(1)+x(2))/2-x(3)  (y(1)+y(2))/2-y(3)];
  l1 = (12 - 2*sqrt(15))/21;   l2 = (12 + 2*sqrt(15))/21;
  %% determine the Gauss points
  GP = [[x(1) y(1)] + l1*v1; [x(2) y(2)] + l1*v2; [x(3) y(3)] + l1*v3;
        [x(1) y(1)] + l2*v1; [x(2) y(2)] + l2*v2; [x(3) y(3)] + l2*v3;
        sum(x)/3 sum(y)/3 ];
  if nargout>1  %% evaluate the basis functions at the Gauss points
    x = GP(:,1)-mx; y = GP(:,2)-my;
    uGPmat = [ones(7,1),x,y,x.^2,x.*y,y.^2,x.^3,x.^2.*y,x.*y.^2,y.^3];
  endif
  if nargout>2  %% evaluate the first derivatives at the Gauss points
    duxGPmat = zeros(7,10);      duyGPmat = zeros(7,10);
    duxGPmat(:,[2 4 5 7 8 9])  = [ones(7,1),2*x,y,3*x.^2,2*x.*y,y.^2];
    duyGPmat(:,[3 5 6 8 9 10]) = [ones(7,1),x,2*y,x.^2,2*x.*y,3*y.^2];
  endif
endfunction
```

6.7.3 The expressions required for the element stiffness matrix

To find a FEM approximation to the solution of the BVP (1) use the weak fomulation in equation (46)

$$0 = \iint_{\Omega} (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\;\phi\;dA - \int_{\Gamma_2} \phi\;\;(g_2 + g_3\,u)\;\;ds\,.$$

and thus for one triangular element Δ evaluate

$$\iint_{\Delta} (a\,\nabla u - u\,\vec{b}) \cdot \nabla\phi + (b_0\,u - f)\;\phi\;dA = \langle \mathbf{ESM}_{\Delta}\,\vec{c},\,\vec{\phi}\rangle\;. \tag{75}$$

The vector $\vec{c} \in \mathbb{R}^{10}$ contains the coefficients for the DOF of the solution u on this triangular element. The vector $\vec{\phi} \in \mathbb{R}^{10}$ contains the coefficients for the DOF of the test function ϕ on this element. As numerical integration scheme use the standard Gauss integration with seven Gauss points, i.e.

$$\iint_{\Delta} f(x,y)\;dA \approx \text{area}(\Delta)\;\sum_{j=1}^{7} w_j\,f(\vec{g}_j)\;.$$

6.7.4 Integration of $f\,\phi$

The vector $\vec{f} \in \mathbb{R}^7$ contains the values of the function $f(x,y)$ at the seven Gauss points $\vec{g}_j$.

$$\begin{aligned}\iint_{\Delta} f\,\phi\;dA &\approx \text{area}(\Delta)\;\langle \text{diag}(\vec{w})\,\vec{f}\,,\,\mathbf{uGPmat}\cdot\mathbf{DOF2BF}\vec{\phi}\rangle \\ &= \text{area}(\Delta)\;\langle(\mathbf{uGPmat}\cdot\mathbf{DOF2BF})^T\,\text{diag}(\vec{w})\,\vec{f}\,,\,\vec{\phi}\rangle\end{aligned}$$

This leads to a contribution to the right hand side of the linear equations.

$$(\mathbf{uGPmat}\cdot\mathbf{DOF2BF})^T\,\text{diag}(\vec{w})\,\vec{f} = \mathbf{DOF2BF}^T\mathbf{uGPmat}^T\,\text{diag}(\vec{w})\,\vec{f} \in \mathbb{R}^{10}$$

6.7.5 Integration of $b_0\, u\, \phi$

The vector $\vec{b}_0 \in \mathbb{R}^7$ contains the values of the function $b_0(x, y)$ at the seven Gauss points $\vec{g}_j$.

$$\begin{aligned}\iint_\Delta b_0\, u\, \phi\, dA &\approx \text{area}(\Delta)\, \langle \text{diag}(\overrightarrow{w\, b_0}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \mathbf{uGPmat} \cdot \mathbf{DOF2BF} \vec{\phi} \rangle \\ &= \text{area}(\Delta)\, \langle (\mathbf{uGPmat} \cdot \mathbf{DOF2BF})^T \cdot \text{diag}(\overrightarrow{w\, b_0}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \vec{\phi} \rangle\end{aligned}$$

This leads to a contribution to the element stiffness matrix $\mathbf{ESM}_\Delta$.

$$\mathbf{DOF2BF}^T \cdot \mathbf{uGPmat}^T \cdot \text{diag}(\overrightarrow{w\, b_0}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF} \in \mathbb{R}^{10\times 10}$$

6.7.6 Integration of $u\, \vec{b} \cdot \nabla\phi$ and $a\, \nabla u \cdot \nabla\phi$

The vectors $\vec{b}_1, \vec{b}_2 \in \mathbb{R}^7$ contains the values of the functions $b_1(x, y)$ and $b_2(x, y)$ at the seven Gauss points $\vec{g}_j$.

$$\begin{aligned}\iint_\Delta u\, b_1\, \frac{\partial\, \phi}{\partial x}\, dA &\approx \text{area}(\Delta)\, \langle \text{diag}(\overrightarrow{w\, b_1}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \mathbf{duxGPmat} \cdot \mathbf{DOF2BF} \vec{\phi} \rangle \\ &= \text{area}(\Delta)\, \langle (\mathbf{duxGPmat} \cdot \mathbf{DOF2BF})^T \cdot \text{diag}(\overrightarrow{w\, b_1}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \vec{\phi} \rangle \\ \iint_\Delta u\, b_2\, \frac{\partial\, \phi}{\partial y}\, dA &\approx \text{area}(\Delta)\, \langle \text{diag}(\overrightarrow{w\, b_2}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \mathbf{duyGPmat} \cdot \mathbf{DOF2BF} \vec{\phi} \rangle \\ &= \text{area}(\Delta)\, \langle (\mathbf{duyGPmat} \cdot \mathbf{DOF2BF})^T \cdot \text{diag}(\overrightarrow{w\, b_2}) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \vec{\phi} \rangle\end{aligned}$$

This leads to a contribution to the element stiffness matrix $\mathbf{ESM}_\Delta$.

$$\mathbf{DOF2BF}^T \cdot \Big(\mathbf{duxGPmat}^T \cdot \text{diag}(\overrightarrow{w\, b_1}) + \mathbf{duyGPmat}^T \cdot \text{diag}(\overrightarrow{w\, b_2}) \Big) \cdot \mathbf{uGPmat} \cdot \mathbf{DOF2BF}$$

The vector $\vec{a} \in \mathbb{R}^7$ contains the values of the function $a(x, y)$ and $b_2(x, y)$ at the seven Gauss points $\vec{g}_j$.

$$\begin{aligned}\iint_\Delta a\, \frac{\partial\, u}{\partial x}\, \frac{\partial\, \phi}{\partial x}\, dA &\approx \text{area}(\Delta)\, \langle \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duxGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \mathbf{duxGPmat} \cdot \mathbf{DOF2BF} \vec{\phi} \rangle \\ &= \text{area}(\Delta)\, \langle (\mathbf{duxGPmat} \cdot \mathbf{DOF2BF})^T \cdot \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duxGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \vec{\phi} \rangle \\ \iint_\Delta a\, \frac{\partial\, u}{\partial y}\, \frac{\partial\, \phi}{\partial y}\, dA &\approx \text{area}(\Delta)\, \langle \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duyGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \mathbf{duyGPmat} \cdot \mathbf{DOF2BF} \vec{\phi} \rangle \\ &= \text{area}(\Delta)\, \langle (\mathbf{duyGPmat} \cdot \mathbf{DOF2BF})^T \cdot \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duyGPmat} \cdot \mathbf{DOF2BF}\, \vec{c},\, \vec{\phi} \rangle\end{aligned}$$

This leads to a contribution to the element stiffness matrix $\mathbf{ESM}_\Delta$.

$$\mathbf{DOF2BF}^T \cdot \Big(\mathbf{duxGPmat}^T\, \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duxGPmat} + \mathbf{duyGPmat}^T\, \text{diag}(\overrightarrow{w\, a}) \cdot \mathbf{duyGPmat} \Big) \cdot \mathbf{DOF2BF}$$

Observe that all contributions to the element stiffness matrix have a leading factor $\mathbf{DOF2BF}^T$ and a trailing factor $\mathbf{DOF2BF}$. This is used in the code `FEMEquationHermite.m` for a slightly shorter runtime.

6.7.7 Integration over boundary segments

- The used procedure is equivalent to the method used for liner elements, since only the two endpoints of the boundary segment are used.
- The applied method has a serious drawback. On a segment with Dirichlet conditions the slope in the direction of the segment is known. This is **not** taken into account, only the values at the nodes are prescribed by the Dirichlet condition. This overly simplistic approach might be the main reason for the lack of numerical accuracy with this implementation of Hermite elements. This is illustrated by two examples in Section 9.5, starting in page 249.
- The main application of Hermite elements is for fourth order problems, e.g. a bending plate problem. The above problematic aspect is of much lesser importance.

6.7.8 Upgrading a linear mesh to a Hermite mesh

In the function `MeshUpgrade()` a triangular mesh of linear elements is upgraded to a Hermite Mesh, i.e. the additional DOF are generated and numbered.

For each element (triangle):

- The first 3 nodes are identical to the linear mesh and contain the number of the nodes.
- The "nodes" 4, 6 and 8 contain the values of the partial derivatives $\frac{\partial u}{\partial x} = u_x$ at the three corners.
- The "nodes" 5, 7 and 9 contain values of the partial derivatives $\frac{\partial u}{\partial y} = u_y$ at the three corners.
- Node 10 contains the value at the midpoint $(x_c, y_c) = \frac{1}{3}\,(x_1 + x_2 + x_3\,,\; y_1 + y_2 + y_3)$.

Thus each entry `elem` in the structure `Mesh` contains the numbers pointing to the values of

$$(u(x_1,y_1), u(x_2,y_2), u(x_3,y_3), u_x(x_1,y_1), u_y(x_1,y_1), u_x(x_2,y_2), u_y(x_2,y_2), u_x(x_3,y_3), u_y(x_3,y_3), u(x_c,y_c))$$

Thus obtain the new number of "nodes" by `nNodesHermite = 3*nNodes + nElem`. All additional entries are new DOFs. Thus obtain the new number of DOF by `nDOFHermite = nDOF + 2*nNodes + nElem`.

Algorithm 2: Update the linear mesh to a Hermite mesh

```
begin
    Read the mesh with linear triangles
    for each triangle do
        use the first 3 nodes for the new Hermite element
        for nodes 1, 2, 3 do
            decide whether this node was already treated
            if treated then
                store the nodes for du/dx and dudy in the new element
            else
                new node with du/dx at node, type 21, add to element
                new node with du/dy at node, type 22, add to element
                store the new DOF for this node in the new element
                mark the node as treated
                increase the number of nodes by 2
            end
        end
        add the centroid as new node, type 23
        increase the number of nodes by 1
    end
end
```

6.7.9 Convert a Hermit mesh to a mesh with linear elements

A triangle with a Hermite element has 10 DOF. Thus it is possible to convert this triangle into a set of 9 triangles with linear elements, i.e. the DOF are the values at those 10 nodes. The solution u_{Herm} generated with the Hermite elements can be converted to a vector u_{Lin} with the values of the solution at the new nodes. The algorithm is spelled out below and implementd in the function `MeshHermit2Linear.m`.

6.8 Convergence of the approximate solutions u_h to the exact solution u

A key feature of a good FEM algorithm is a rapid convergence. As the diameter h of the triangles converges to 0, the approximate solutions $u_h(x,y)$ should converge to the exact solution $u(x,y)$. The statements below are correct for very smooth exact solutions and "nice" domains. Find more information in books on the mathematical background of FEM, e.g. [AxelBark84] or consult [Stah08].

It is convenient to state the approximation results using two norms on the function space $L_2(\Omega)$ and the Sobolev space $V = H^1(\Omega) = W^{1,2}(\Omega)$. The norms are given by

$$\|u\|_2^2 = \iint\limits_{\Omega} u^2(x,y)\; dA \quad \text{and} \quad \|u\|_V^2 = \iint\limits_{\Omega} u^2(x,y) + \|\nabla u(x,y)\|^2\; dA\,.$$

Algorithm 3: Convert a Hermite mesh to a linear mesh

```
begin
    Read the mesh with Hermite triangles
    Create variables to reserve the memory
    for each Hermite triangle do
        use the first 3 nodes and node 10 for the new linear elements
        for edges 1, 2, 3 do
            decide whether this edge was already treated
            if treated then
                store the nodes for the two additional nodes on this edge
            else
                evaluate the coordinates of the two new nodes
            end
        end
        evaluate at the new nodes using the cubic interpolation, using uHermite
        store the results in uLinear
        generate and store the numbers of the 9 new triangles
    end
    evaluate the areas of the new triangles
    evaluate the Gauss points for the new triangles
    return the resulting mesh and the values of the solution uLinear
end
```

The convergence results assume that the meshes are well constructed, e.g. satisfy a minimal angle condition. The scalar h is the typical (or maximal) diameter of the triangles used for the mesh.

- If the solutions u_h are generated by first order, triangular elements, i.e. piecewise linear functions, then

$$\|u_h - u\|_V \leq C\, h \quad \text{and} \quad \|u_h - u\|_2 \leq C_1\, h^2$$

  for some constants C and C_1 independent on h. A short formulation is

  - u_h converges to u with an error proportional to h^2 as $h \to 0$.
  - ∇u_h converges to ∇u with an error proportional to h as $h \to 0$.

- If the solutions u_h are generated by second order, triangular elements, i.e. piecewise quadratic functions, then

$$\|u_h - u\|_V \leq C\, h^2 \quad \text{and} \quad \|u_h - u\|_2 \leq C_1\, h^3$$

  for some constants C and C_1 independent on h. A short formulation is

  - u_h converges to u with an error proportional to h^3 as $h \to 0$.
  - ∇u_h converges to ∇u with an error proportional to h^2 as $h \to 0$.

- If the solutions u_h are generated by third order, triangular elements, i.e. piecewise cubic functions, then

$$\|u_h - u\|_V \leq C\, h^3 \quad \text{and} \quad \|u_h - u\|_2 \leq C_1\, h^4$$

  for some constants C and C_1 independent on h. A short formulation is

  - u_h converges to u with an error proportional to h^4 as $h \to 0$.
  - ∇u_h converges to ∇u with an error proportional to h^3 as $h \to 0$.

Observe that the convergence results are about the integral of differences, and not point-wise estimates. In addition the exact solution u is assumed to be smooth. Thus one has to be careful when using the estimates for problems with limited regularity of the type in Section 9.4.

6.9 Solving semilinear problems

For a smooth function $f(x,u)$ examine a nonlinear boundary value problem[29] of the form

$$\begin{aligned} -\nabla\,(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,u) && \text{for} \quad (x,y)\in\Omega \\ u &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned} \quad . \tag{76}$$

The essential tool is Newton's method, combined with a partial substitution. For this use a linear Taylor approximation of the nonlinear function $f(x,y,u)$

$$f(x,y,u+\phi) \approx f(x,y,u) + \frac{\partial\, f(x,y,u)}{\partial u}\,\phi = f(x,y,u) + f_u(x,y,u)\,\phi$$

For an approximate solution $u_n(x)$ search a solution of the form $u_n(x,y)+\phi(x,y)$. Examine the linear boundary value problem for the perturbation ϕ.

$$\begin{aligned} -\nabla\cdot(a\nabla(u_n+\phi) - (u_n+\phi)\,\vec{b}) + b_0(u_n+\phi) &= f(u_n) + f_u(u_n)\,\phi && \text{in } \Omega \\ u_n+\phi &= g_1 && \text{on } \Gamma_1 \\ \vec{n}\cdot(a\nabla(u_n+\phi) - (u_n+\phi)\,\vec{b}) &= g_2 + g_3(u_n+\phi) && \text{on } \Gamma_2 \end{aligned}$$

Solving for ϕ leads to the BVP

$$\begin{aligned} -\nabla\,(a\nabla\phi - \phi\,\vec{b}) + (b_0 - f_u(u_n))\,\phi &= +\nabla\,(a\nabla u_n - u_n\,\vec{b}) - b_0\,u_n + f(u_n) && \text{in } \Omega \\ \phi &= g_1 - u_n \;=\; 0 && \text{on } \Gamma_1 \\ \vec{n}\cdot(a\,\nabla\phi - \phi\,\vec{b}) &= -\vec{n}\cdot(a\,\nabla u_n - u_n\,\vec{b}) + g_2 + g_3\,u_n + g_3\,\phi \;=\; g_3\,\phi && \text{on } \Gamma_2 \end{aligned} \tag{77}$$

A tricky part is to evaluate the RHS $\nabla(a\nabla u_n - u_n\,\vec{b}) - b_0\,u_n$ in the above BVP (77) for ϕ. Solving the linear BVP

$$\begin{aligned} -\nabla\,(a\,\nabla u - u\,\vec{b}) + b_0\,\phi &= 0 && \text{in } \Omega \\ u &= g_1 && \text{on } \Gamma_1 \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{on } \Gamma_2 \end{aligned}$$

leads to a linear system $\mathbf{A}_0\vec{u} + \vec{g} = \vec{0}$. The missing $f(x,y)$ in the above BVP can be implemented by using the vector $\vec{f}$ with the values of the function f at the Gauss points. Construct a weight matrix $\mathbf{W}$ and solve the system $\mathbf{A}_0\vec{u} + \vec{g} + \mathbf{W}\vec{f} = \vec{0}$. Then the RHS vector for the linear system to solve the BVP (77) is $\mathbf{A}_0\vec{u} + \vec{g} + \mathbf{W}\vec{f}$, where $\vec{f}$ is generated using the current solution $f(x,y,u_n)$.

Then update the solution to $u_{n+1} = u_n + \phi$. For an initial guess $u_0(x,y)$ close enough to an isolated solution the Newton based algorithm will converge quadratically.

To implement the above algorithm in `FEMoctave` start out with an initial function $u(x,y) = u_0(x,y)$, hopefully close to the true solution.

- Start with the function $u_0(x,y)$ and evaluate (if necessary) $f_0 = f(x,y,u_0(x,y))$ at the Gauss points. Evaluate the coefficients at the Gauss points.

$$a_0 = a(x,y,u_0) \quad , \quad b_1 = b_1(x,y) \quad , \quad b_2 = b_2(x,y) \quad , \quad b_0 = b_0(x,y) \quad \text{and} \quad b_n = 0\,.$$

- Solve the boundary value problem (76) for u_n

$$\begin{aligned} -\nabla\,(a_0\,\nabla u_n - u_n\,\vec{b}_{1,2}) + b_0\,u &= f_0 && \text{for} \quad (x,y)\in\Omega \\ u_n &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\ \vec{n}\cdot(a_0\,\nabla u - u_n\,\vec{b}_{1,2}) &= g_2 + g_3\,u_n && \text{for} \quad (x,y)\in\Gamma_2 \end{aligned}$$

- Repeat

[29] Functions of the type $f(x,y,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial y})$ are not implemented, but it is possible to do so. Complex coefficients are not implemented yet either. I have not run across an application using these features. Let me know if you need this feature.

- Store the current solution $u_{old} = u_n$.
- If f depends on u evaluate

$$\begin{aligned} f_n &= f(x, y, u_n(x, y)) \quad \text{at the Gauss points} \\ f_u &= \frac{\partial}{\partial u} f(x, y, u_n(x, y)) \quad \text{at the Gauss points} \end{aligned}$$

- Evaluate at the nodes

$$\text{RHS}_n = -\nabla \left(a_0 \, \nabla u_n - u_n \, \vec{b} \right) + b_0 \, u - f_n$$

  This can be implemented with a matrix multiplication, see the above explanation.
- Solve the boundary value problem for the perturbation ϕ

$$-\nabla \left(a \nabla \phi - \phi \, \vec{b} \right) + \left(b_0 - f_u(u) \right) \phi = -\text{RHS}$$

  with homogeneous boundary conditions.
- Update $u_n \longrightarrow u_n + \phi$.

- until $\|\phi\| = \|u_n - u_{old}\|$ small enough or too many iterations.

For the convergence test absolute and relative values are used, if one of them is small enough the algorithm stops.

This is the algorithm used in the command `BVP2DNL()`.

6.10 Dynamic problems

There are two distinct classes of dynamic problems:

- Parabolic problems with the heat equation $\dot{u} = \Delta u$ as the typical example.
- Hyperbolic problems with the wave equation $\ddot{u} = \Delta u$ as the typical example.

For both types the following sections will present unconditionally stable, consistent time stepping algorithms.

6.10.1 Dynamic problems of the heat equation type

Examine an IBVP (6) of parabolic type.

$$\begin{aligned} \rho \, \tfrac{\partial}{\partial t} \, u - \nabla \cdot \left(a \, \nabla u - u \, \vec{b} \right) + b_0 \, u &= f && \text{for} \quad (x, y, t) \in \Omega \times (0, T] \\ u &= g_1 && \text{for} \quad (x, y, t) \in \Gamma_1 \times (0, T] \\ \vec{n} \cdot \left(a \, \nabla u - u \, \vec{b} \right) &= g_2 + g_3 \, u && \text{for} \quad (x, y, t) \in \Gamma_2 \times (0, T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t = 0 \end{aligned}$$

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$. Solve the static problem with nonhomogeneous boundary conditions.

$$\begin{aligned} -\nabla \cdot \left(a \, \nabla u_B - u_B \, \vec{b} \right) + b_0 \, u_B &= 0 && \text{for} \quad (x, y) \in \Omega \\ u_B &= g_1 && \text{for} \quad (x, y) \in \Gamma_1 \\ \vec{n} \cdot \left(a \, \nabla u_B + u_B \, \vec{b} \right) &= g_2 + g_3 \, u_B && \text{for} \quad (x, y) \in \Gamma_2 \end{aligned} \tag{78}$$

Then the new function $v(x, y, t) = u(x, y, t) - u_B(x, y)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. $g_1 = g_2 = 0$.

$$\begin{aligned} \rho \, \tfrac{\partial}{\partial t} \, v - \nabla \cdot \left(a \, \nabla u - v \, \vec{b} \right) + b_0 \, v &= f && \text{for} \quad (x, y, t) \in \Omega \times (0, T] \\ v &= 0 && \text{for} \quad (x, y, t) \in \Gamma_1 \times (0, T] \\ \vec{n} \cdot \left(a \, \nabla v + v \, \vec{b} \right) &= g_3 \, v && \text{for} \quad (x, y, t) \in \Gamma_2 \times (0, T] \\ v &= u_0 - u_B && \text{on} \quad \Omega \quad \text{at} \quad t = 0 \end{aligned}$$

This equation is transformed to a system of ordinary differential equations.

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{v}_0\,. \tag{79}$$

The implementation assumes that the coefficient functions ρ, a, b_0, $\vec{b}$ and g_i depend on (x, y), while f may depend on time t too and the position (x, y). There are four algorithms for the time stepping available, identical to the solvers for the 1D problem in Section 4.12 on page 88.

- `CN`: the standard Crank–Nicolson algorithm. This is the default algorithm.
- `implicit`: the standard implicit solver.
- `explicit`: the standard explicit solver.
- `RK`: an implicit Runge–Kutta algorithm.

Some documentation on these standard algorithms is shown in Section 7.6 on page 215.

For the command `IBVP2D()` the Crank–Nicolson approximation can be used to advance the solution of the ODE (79) from time t to $t + \Delta t$. Approximate the time derivative in (79) by a centered difference formula.

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t)}{\Delta t} &= -\mathbf{A}\,\frac{\vec{v}(t+\Delta t) + \vec{v}(t)}{2} + \vec{f}(t + \Delta t/2) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= +\left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}(t) + \Delta t\;\vec{f}(t + \Delta t/2)
\end{aligned}$$

For each time step a linear system has to be solved. Observe that the matrix on the left does not change as time advances. There are a few option available[30] to solve the linear system for each time step. Using a sparsity preserving LU factorization of the matrix on the left, these systems can be solved efficiently. The matrices $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices with $\mathbf{P}^{-1} = \mathbf{P}^T$. A substantial amount of time has to be used to perform the LU factorization, but then the time stepping is fast.

$$\begin{aligned}
\mathbf{P}\,\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q} &= \mathbf{L}\,\mathbf{U} && \text{LU factorization} \\
\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v} &= \vec{b} && \text{system to be solved} \\
\mathbf{P}\,\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q}\,\mathbf{Q}^{-1}\,\vec{v} &= \mathbf{P}\,\vec{b} && \\
\mathbf{L}\,\mathbf{U}\,\mathbf{Q}^{-1}\,\vec{v} &= \mathbf{P}\,\vec{b} && \\
\vec{v} &= \mathbf{Q}\,(\mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{P}\,\vec{b}))) && \text{in the } \textit{Octave} \text{ code}
\end{aligned}$$

With the computed $\vec{v}(t)$ then find the solution $\vec{u}(t) = \vec{v}(t) + \vec{u}_B$ of the original problem. This was the default algorithm used with the command `IBVP2D()` in `FEMoctave` for version up to 2.1.8 .

With the `lu()` command with 5 return arguments an additional scaling matrix $\mathbf{R}$ is used and should lead to sparser factorization and improved numerical stability. Then the above leads to

$$\begin{aligned}
\mathbf{P}\,(\mathbf{R}\backslash\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right))\,\mathbf{Q} &= \mathbf{L}\,\mathbf{U} && \text{LU factorization, with rescaling} \\
\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v} &= \vec{b} && \text{system to be solved} \\
\mathbf{P}\,(\mathbf{R}\backslash\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right))\,\mathbf{Q}\,\mathbf{Q}^{-1}\,\vec{v} &= \mathbf{P}\,\mathbf{R}\backslash\vec{b} && \\
\mathbf{L}\,\mathbf{U}\,\mathbf{Q}^{-1}\,\vec{v} &= \mathbf{P}\,\mathbf{R}\backslash\vec{b} && \\
\vec{v} &= \mathbf{Q}\,(\mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{P}\,(\mathbf{R}\backslash\vec{b})))) && \text{in the } \textit{Octave} \text{ code}
\end{aligned}$$

This the default algorithm used with the command `IBVP2D()` in `FEMoctave`.

If the matrix $\mathbf{A}$ is symmetric and positive definite one can use Cholesky factorization with row and column permutations to preserve the sparsity, as much as possible. This should be faster than a LU factorization. The Cholesky but its not!

[30] To solve the systems $\mathbf{A}\,\vec{x} = \vec{b}$ for many different vectors $\vec{b}$ one could use

- $\vec{x} = \mathbf{A}^{-1}\,\vec{b}$ for each time step: extremely inefficient.
- $\vec{x} = \mathbf{A}\backslash\vec{b}$ for each time step: inefficient.
- $[\mathbf{L}, \mathbf{U}] = \text{lu}(\mathbf{A})$ once, i.e. $\mathbf{A} = \mathbf{L}\,\mathbf{U}$. Then $\vec{x} = \mathbf{U}\backslash(\mathbf{L}\backslash\vec{b})$ at each time step: reasonably efficient.
- $[\mathbf{L}, \mathbf{U}, \mathbf{P}, \mathbf{Q}] = \text{lu}(\mathbf{A})$ once, i.e. $\mathbf{P}\,\mathbf{A}\,\mathbf{Q} = \mathbf{L}\,\mathbf{U}$. Then $\vec{x} = \mathbf{Q}\,(\mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{P}\,\vec{b})))$ at each time step: efficient.
- $[\mathbf{L}, \mathbf{U}, \mathbf{P}, \mathbf{Q}, \mathbf{R}] = \text{lu}(\mathbf{A})$ once, i.e. $\mathbf{P}\,(\mathbf{R}\backslash\mathbf{A})\,\mathbf{Q} = \mathbf{L}\,\mathbf{U}$. Then $\vec{x} = \mathbf{Q}\,(\mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{P}\,(\mathbf{R}\backslash\vec{b}))))$ at each time step: efficient and more numerical stability.

factorization is used with the command `IBVP2Dsym()`.

$$\begin{aligned}
\mathbf{Q}^T \left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \mathbf{Q} &= \mathbf{R}^T\,\mathbf{R} && \text{Cholesky factorization}\\
\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \vec{v} &= \vec{b} && \text{system to be solved}\\
\mathbf{Q}^T \left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \mathbf{Q}\,\mathbf{Q}^T \vec{v} &= \mathbf{Q}^T \vec{b} &&\\
\mathbf{R}^T\,\mathbf{R}\,\mathbf{Q}^T \vec{v} &= \mathbf{Q}^T \vec{b} &&\\
\vec{v} &= \mathbf{Q}\,(\mathbf{R}\backslash(\mathbf{R}^T\backslash(\mathbf{Q}^T\,\vec{b}))) && \text{in the } \textit{Octave} \text{ code}
\end{aligned}$$

The *Octave* manual claims that a lower Cholesky factorization is often faster.

$$\begin{aligned}
\mathbf{Q}^T \left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \mathbf{Q} &= \mathbf{L}\,\mathbf{L}^T && \text{lower Cholesky factorization}\\
\left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \vec{v} &= \vec{b} && \text{system to be solved}\\
\mathbf{Q}^T \left(\mathbf{W} + \tfrac{\Delta t}{2}\,\mathbf{A}\right) \mathbf{Q}\,\mathbf{Q}^T \vec{v} &= \mathbf{Q}^T \vec{b} &&\\
\mathbf{L}\,\mathbf{L}^T\,\mathbf{Q}^T \vec{v} &= \mathbf{Q}^T \vec{b} &&\\
\vec{v} &= \mathbf{Q}\,(\mathbf{L}^T\backslash(\mathbf{L}\backslash(\mathbf{Q}^T\,\vec{b}))) && \text{in the } \textit{Octave} \text{ code}
\end{aligned}$$

6.10.2 Using eigenvalues for dynamic problems of the heat equation type

With equation (79) for $\vec{f} = \vec{0}$

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{0} \quad \text{with} \quad \vec{v}(0) = \vec{v}_0$$

observe that a generalized eigenvalue λ with eigenvector $\vec{v}$, i.e.

$$\mathbf{A}\,\vec{v} = \lambda\,\mathbf{W}\,\vec{v}$$

leads to a solution $\vec{u}(t) = c\,\exp(-\lambda\,t)\,\vec{v}$, since

$$\begin{aligned}
\mathbf{W}\,\frac{d}{dt}\,\vec{u}(t) &= -\lambda\,\mathbf{W}\,\vec{v}\,\exp(-\lambda\,t)\\
\mathbf{A}\,\vec{u}(t) &= +\lambda\,\mathbf{W}\,\vec{v}\,\exp(-\lambda\,t)
\end{aligned}$$

Thus for $\lambda > 0$ find an exponentially decaying solution of the IBVP.

6.10.3 Semilinear dynamic problems of the heat equation type

The approach is very similar to the above section 6.10.1 for linear problems. For semilinear problems the RHS is given by $f(x, y, t, u)$. The general semilinear, parabolic problem (8) is of the form

$$\begin{aligned}
\rho\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f(x,y,t,u) && \text{for} \quad (x,y,t) \in \Omega \times (t_0, T]\\
u &= g_1 && \text{for} \quad (x,y,t) \in \Gamma_1 \times (t_0, T]\\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t) \in \Gamma_2 \times (t_0, T]\\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t = t_0
\end{aligned}$$

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$. Solve the static problem with nonhomogeneous boundary conditions.

$$\begin{aligned}
-\nabla\cdot(a\,\nabla u_B - u_B\,\vec{b}) + b_0\,u_B &= 0 && \text{for} \quad (x,y,t) \in \Omega\\
u_B &= g_1 && \text{for} \quad (x,y) \in \Gamma_1\\
\vec{n}\cdot(a\,\nabla u_B + u_B\,\vec{b}) &= g_2 + g_3\,u_B && \text{for} \quad (x,y,t) \in \Gamma_2
\end{aligned}$$

Then the new function $v(x, y, t) = u(x, y, t) - u_B(x, y)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. $g_1 = g_2 = 0$.

$$\begin{aligned}
\rho\,\tfrac{\partial}{\partial t}\,v - \nabla\cdot(a\,\nabla u - v\,\vec{b}) + b_0\,v &= f(x,y,t,v+u_B) && \text{for} \quad (x,y,t) \in \Omega \times (0, T]\\
v &= 0 && \text{for} \quad (x,y,t) \in \Gamma_1 \times (0, T]\\
\vec{n}\cdot(a\,\nabla v + v\,\vec{b}) &= g_3\,v && \text{for} \quad (x,y,t) \in \Gamma_2 \times (0, T]\\
v &= u_0 - u_B && \text{on} \quad \Omega \quad \text{at} \quad t = 0
\end{aligned}$$

This equation is transformed to a system of nonlinear ordinary differential equations.

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t, \vec{v}(t) + \vec{u}_B) \quad \text{with} \quad \vec{v}(0) = \vec{v}_0\,. \tag{80}$$

The time stepping algorithms for (80) are spelled out in Section 7.6.6 on page 218 for 1D problems.

- CNEXP: a standard Crank–Nicolson approach for the linear contribution and an explicit step for the nonlinear contribution.

$$\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_{n+1} = \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_n + \Delta t\,\mathbf{M}\,\vec{f}(t + \frac{1}{2}\,\Delta t, \vec{v}_n + \vec{u}_B)$$

- CNPC: a standard Crank–Nicolson approach for the linear contribution and an additional predictor–corrector step.

$$\begin{aligned}
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_{temp} &= \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_n + \Delta t\,\mathbf{M}\,\vec{f}(t, \vec{v}_n + \vec{u}_B) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_{n+1} &= \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}_n + \frac{\Delta t}{2}\,\mathbf{M}\,\left(\vec{f}(t, \vec{v}_n + \vec{u}_B) + \vec{f}(t + \Delta t, \vec{v}_{temp} + \vec{u}_B)\right)
\end{aligned}$$

6.10.4 Dynamic problems of the wave equation type

Examine an IBVP (9) of hyperbolic type.

$$\begin{aligned}
\rho\,\tfrac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\tfrac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\
u &= g_1 && \text{for} \quad (x,y,t) \in \Gamma_1 \times (0,T] \\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t) \in \Gamma_2 \times (0,T] \\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t = 0 \\
\tfrac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t = 0
\end{aligned}$$

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$, solving the boundary value problem (78). Then the new function $v(x,y,t) = u(x,y,t) - u_B(x,y)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. $g_1 = g_2 = 0$.

$$\begin{aligned}
\rho\,\tfrac{\partial^2}{\partial t^2}\,v + 2\,\alpha\,\tfrac{\partial}{\partial t}\,v(t) - \nabla\cdot(a\,\nabla u - v\,\vec{b}) + b_0\,v &= f && \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\
v &= 0 && \text{for} \quad (x,y,t) \in \Gamma_1 \times (0,T] \\
\vec{n}\cdot(a\,\nabla v - v\,\vec{b}) &= g_3\,v && \text{for} \quad (x,y,t) \in \Gamma_2 \times (0,T] \\
v &= u_0 - u_B && \text{on} \quad \Omega \quad \text{at} \quad t = 0 \\
\tfrac{\partial}{\partial t}\,v &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t = 0
\end{aligned}$$

This equation is transformed to a system of ordinary differential equations.

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{u}_0 - \vec{u}_B\,, \quad \frac{d}{dt}\,\vec{v}(0) = \vec{v}_0 \tag{81}$$

The implementation assumes that the coefficient functions ρ, α, a, b_0, $\vec{b}$ and g_i depend on (x,y), while f may depend on time t and the position (x,y). Then use an implicit approximation[31] to advance the solution from time $t - \Delta t$ and t to $t + \Delta t$.

$$\begin{aligned}
\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) - \mathbf{A}\,\vec{v}(t) + \vec{f}(t) \\
\mathbf{W}\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -2\,\mathbf{D}\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t-\Delta t)}{2\,\Delta t} - \\
&\quad -\mathbf{A}\,\frac{\vec{v}(t-\Delta t) + 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{4} + \vec{f}(t) \\
\left(+\mathbf{W} + \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= -\left(\mathbf{W} - \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t) + \\
&\quad + \left(2\,\mathbf{W} - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t) + (\Delta t)^2\,\vec{f}(t)
\end{aligned}$$

[31] This is a standard choice and unconditionally stable, see e.g. [Stah08, §4].

This scheme is unconditionally stable and consistent of order 2. Observe that the matrices do not change as time advances. Thus use again a sparsity preserving LU factorization for the time stepping, see the above Section 6.10.1. `FEMoctave` uses the sparsity preserving, rescaled version of `lu()`. The above scheme is unconditionally stable, at least for constant coefficients[32].

- To construct the solution at the initial time Δt one could use the initial value u_0 and initial velocity v_0 and a scheme with the same order of consistency with respect to time. An explicit scheme for the first step leads to

$$\begin{aligned}
\frac{d}{dt}\,\vec{v}(0) &= \vec{v}_0 \approx \frac{\vec{v}(\Delta t)-\vec{v}(-\Delta t)}{2\,\Delta t} \quad \Longrightarrow \quad \vec{v}(-\Delta t) \approx \vec{v}(\Delta t) - 2\,\Delta t\,\vec{v}_0 \\
\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) - \mathbf{A}\,\vec{v}(t) + \vec{f}(t) \\
\mathbf{W}\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -2\,\mathbf{D}\,\frac{\vec{v}(t+\Delta t)-\vec{v}(t-\Delta t)}{2\,\Delta t} - \mathbf{A}\,\vec{v}(t) + \vec{f}(t) \\
(\mathbf{W}+\Delta t\,\mathbf{D})\,\vec{v}(t+\Delta t) &= -(\mathbf{W}-\Delta t\,\mathbf{D})\,\vec{v}(t-\Delta t) + 2\,\mathbf{W}\,\vec{v}(t) + (\Delta t)^2\,(-\mathbf{A}\,\vec{v}(t) + \vec{f}(t)) \\
(\mathbf{W}+\Delta t\,\mathbf{D})\,\vec{v}(\Delta t) &= -(\mathbf{W}-\Delta t\,\mathbf{D})\,(\vec{v}(\Delta t) - 2\,\Delta t\,\vec{v}_0) + \\
&\quad +2\,\mathbf{W}\,(\vec{u}_0 - \vec{u}_B) + (\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0 - u_B) + \vec{f}(0)) \\
2\,\mathbf{W}\,\vec{v}(\Delta t) &= +2\,(\mathbf{W}-\Delta t\,\mathbf{D})\,\Delta t\,\vec{v}_0 + \\
&\quad +2\,\mathbf{W}\,(\vec{u}_0 - \vec{u}_B) + (\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0 - \vec{u}_B) + \vec{f}(0)) \\
\mathbf{W}\,\vec{v}(\Delta t) &= (\mathbf{W}-\Delta t\,\mathbf{D})\,\Delta t\,\vec{v}_0 + \\
&\quad +\mathbf{W}\,(\vec{u}_0 - \vec{u}_B) + \frac{1}{2}\,(\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0 - \vec{u}_B) + \vec{f}(0))\,.
\end{aligned}$$

  The conditional stability for this single step might cause numerical trouble.

- One could also use $\vec{v}(-\Delta t) \approx \vec{v}(\Delta t) - 2\,\Delta t\,\vec{v}_0$ in the implicit scheme at $t = 0$.

$$\begin{aligned}
\left(+\mathbf{W} + \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}(\Delta t) &= -\left(\mathbf{W} - \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)(\vec{v}(\Delta t) - 2\,\Delta t\,\vec{v}_0) + \\
&\quad +\left(2\,\mathbf{W} - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\vec{v}(0) + (\Delta t)^2\,\vec{f}(0) \\
\left(+2\,\mathbf{W} + 2\,\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}(\Delta t) &= +2\,\Delta t\left(\mathbf{W} - \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}_0 + \\
&\quad +\left(2\,\mathbf{W} - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\vec{v}(0) + (\Delta t)^2\,\vec{f}(0) \\
\left(+\mathbf{W} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}(\Delta t) &= +\Delta t\left(\mathbf{W} - \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}_0 + \\
&\quad +\left(\mathbf{W} - \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\vec{v}(0) + \frac{(\Delta t)^2}{2}\,\vec{f}(0)
\end{aligned}$$

  This initial step requires solving a new system of linear equations. If there is no damping term ($\mathbf{D} = 0$) it is the same system as for the time stepping, thus is used.

The above implicit solver is the default for the algorithm in `I2BVP2D()`. It is very similar to the algorithm in Section 7.7.1 for 1D problems. In addition an explicit solver is available in `I2BVP2D()` too, with the description in Section 7.7.2.

6.10.5 Using eigenvalues for dynamic problems of the wave equation type

With equation (81) for $\vec{f} = \vec{0}$ and a damping factor of the form $D\,\mathbf{W}$ with a constant $D \geq 0$ (instead of the matrix $\mathbf{D}$)

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,D\,\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{0} \tag{82}$$

[32] I have a proof in `WaveStability.tex`.

observe that a generalized eigenvalue $\lambda > 0$ with eigenvector $\vec{v}$, i.e. $\mathbf{A}\,\vec{v} = \lambda\,\mathbf{W}\,\vec{v}$ and weak damping $0 \le D < \sqrt{\lambda}$ leads to a solution $\vec{u}(t) = \exp(\mu\,t)\,\vec{v}$ with $\mu \in \mathbb{C}$, since

$$\begin{aligned} \vec{0} &= \mu^2\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t) + \mu\,2\,D\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t) + \lambda\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t) \\ 0 &= \mu^2 + \mu\,2\,D + \lambda \\ \mu_{1,2} &= \frac{1}{2}\left(-2\,D \pm \sqrt{4\,D^2 - 4\,\lambda}\right) \;=\; -D \pm i\,\sqrt{\lambda - D^2} \;\in\; \mathbb{C}\,. \end{aligned}$$

Thus the real solutions are of the form

$$\vec{u}(t) = \exp(-D\,t)\,\left(\vec{v}_1\,\cos(\sqrt{\lambda - D^2}\,t) + \vec{v}_2\,\sin(\sqrt{\lambda - D^2}\,t)\right)\,.$$

The angular velocity of the exponentially decaying oscillations is given by $\omega = \sqrt{\lambda - D^2}$.

- For the case of strong damping $D > \sqrt{\lambda}$ use

  $$\mu_{1,2} = -D \pm \sqrt{D^2 - \lambda} \in \mathbb{R}$$

  to find two exponentially decaying solutions

  $$\vec{u}(t) = c_1\,\exp(-D + \sqrt{D^2 - \lambda}\,t)\,\vec{v}_1 + c_2\,\exp(-D - \sqrt{D^2 - \lambda}\,t)\,\vec{v}_2\,.$$

- If the damping term is not in the special form $D\,\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t)$ the above, simple approach does not work. Instead replace equation (82) by the first order system

  $$\frac{d}{dt}\,\begin{pmatrix} v(t) \\ \mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) \end{pmatrix} = \begin{pmatrix} \frac{d}{dt}\,\vec{v}(t) \\ -2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) - \mathbf{A}\,\vec{v}(t) \end{pmatrix}$$

  or with a matrix notation

  $$\frac{d}{dt}\,\begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{bmatrix}\,\begin{pmatrix} v(t) \\ \frac{d}{dt}\,\vec{v}(t) \end{pmatrix} = \begin{bmatrix} \mathbf{0} & \mathbb{I} \\ -\mathbf{A} & -2\,\mathbf{D} \end{bmatrix}\,\begin{pmatrix} v(t) \\ \frac{d}{dt}\,\vec{v}(t) \end{pmatrix}\,.$$

  Thus the generalized eigenvalues of

  $$\begin{bmatrix} \mathbf{0} & \mathbb{I} \\ -\mathbf{A} & -2\,\mathbf{D} \end{bmatrix}\,\vec{x} = \lambda\,\begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{bmatrix}\,\vec{x}$$

  provide information on the behavior of the solutions of the wave equation. This is not implemented in FEMoctave.

6.11 Inverse power iteration or `eigs()` to determine small eigenvalues of positive definite matrices

The algorithm to solve the generalized eigenvalue problem

$$\mathbf{A}\,\vec{x} = \lambda\,\mathbf{B}\,\vec{x}$$

for given, positive definite matrices $\mathbf{A}$ and $\mathbf{B}$ is based on inverse power iteration. A small number of the smallest eigenvalues can be estimated with reasonable efficiency. This algorithm imposes some restrictions though:

- Both matrices $\mathbf{A}$ and $\mathbf{B}$ have to be symmetric and strictly positive definite.
- Only very few eigenvalues and eigenvectors should be computed. The convergence rate for too many eigenvalues is unacceptable.
- There are obvious improvements possible, but I hope for an *Octave* implementation of the command `eigs()`. **This is the case now, thus I use** `eigs()`. Thus some of the notes on eigenvalues do not apply any more.

The algorithm is presented in [GoluVanLoan96] and some more details are worked out in [VarFEM], available at andreasstahel.github.io/Notes/VarFEM.pdf.

To determine the first m eigenvalues proceed as follows.

- Create an $n \times m$ matrix $\mathbf{V}_0$ with the initial vectors $\vec{v}_{j,0}$ as its columns.
- repeat until desired precision is reached
  - solve the matrix equation $\mathbf{A} \cdot \mathbf{V}_k = \mathbf{B} \cdot \mathbf{V}_{k-1}$ or $\mathbf{V}_k = \mathbf{A}^{-1} \cdot \mathbf{B} \cdot \mathbf{V}_{k-1}$
  - ortho-normalize the columns of $\mathbf{V}_k$, using a generalized Gram-Schmidt algorithm. The resulting columns of $\mathbf{V}_k$ are orthonormal with respect to the scalar product $\langle \vec{x}\,,\, \mathbf{B}\vec{y} \rangle$.
- for $j = 1, 2 \ldots m$ compute $\beta_j = \langle \mathbf{V}(:,j)\,,\, \mathbf{A} \cdot \mathbf{V}(:,j) \rangle$. Then β_j should be good approximations to the eigenvalues.

The error estimates are based on results in [Demm97]. For a normalized, approximate eigenvector $\vec{v}_i$ and the corresponding approximate eigenvalue β_i compute the residual $\vec{r} = \mathbf{A}\,\vec{v}_i - \beta_i\,\mathbf{B}\,\vec{v}_i$. Then the estimates

$$\min_{\lambda_j \in \sigma(\mathbf{A})} |\beta_i - \lambda_j| \le \sqrt{\langle \vec{r}\,,\, \mathbf{B}^{-1}\vec{r} \rangle} \quad \text{and} \quad |\beta_i - \lambda_j| \le \frac{\langle \vec{r}\,,\, \mathbf{B}^{-1}\vec{r} \rangle}{\text{gap}} \tag{83}$$

are valid. The denominator gap measures the distance to the next eigenvalue.

$$\text{gap} = \min\{|\beta_i - \lambda_j| \;:\; \lambda_j \in \sigma(\mathbf{A}), j \neq i\}\,.$$

Without the exact values of the eigenvalues λ_i there is no way to compute the gap exactly. Thus use the approximate eigenvalues. Expect the error estimate to have its problems at multiple eigenvalues. For the largest computed eigenvalue one can not estimate the size of the gap reliably, since no information on the next eigenvalue is available.

7 The Algorithms for 1D FEM

In this section the algorithm for FEM algorithms for problems with one independent variable are presented, for sake of completeness. Only second order elements will be used. These notes are based on the presentation in the class room notes [Stah08, §6.9].

7.1 The 1D problems to be examined are shown.

7.2 The element stiffness matrix is constructed.

7.3 The boundary conditions are taken into account.

7.4 The resulting linear system of equations is solved.

7.5 The tools to evaluate the solutions of 1D problems are introduced.

7.6 Dynamic problems of order 1 with respect to time are examined. Four different time steppers are introduced and compared. The algorithm for nonlinear problems is spelled out.

7.7 Dynamic problems of order 2 with respect to time are examined. An implicit and an explicit time stepper are presented. The algorithm for nonlinear problems is spelled out.

7.8 An algorithm to solve a nonlinear boundary value problems is presented.

7.1 The problems to be examined

The ordinary differential equation to be examined is of the form

$$-\left(a(x)\,u'(x)\right)' + b(x)\,u'(x) + c(x)\,u(x) = d(x)\,f(x) \tag{84}$$

with some boundary conditions. Multiplying (84) by a smooth test function $\phi(x)$ and an integration by parts leads to

$$\begin{aligned} 0 \;&=\; \int_{x_0}^{x_n} \Big(-\left(a(x)\,u'(x)\right)' + b(x)\,u'(x) + c(x)\,u(x) - d(x)\,f(x)\Big)\;\phi(x)\;dx \;= \\ &=\; -a(x)\,u'(x)\,\phi(x)\Big|_{x=x_0}^{x_n} + \int_{x_0}^{x_n} a(x)\,u'(x)\,\phi'(x) + \left(b(x)\,u'(x) + c(x)\,u(x) - d(x)\,f(x)\right)\phi(x)\,dx \end{aligned} \tag{85}$$

The ODE (84) has to be supplemented with boundary conditions at the two endpoints $x = x_0$ and $x = x_n$.

$$\begin{array}{rcll} u(x_i) & = & g_D & \text{Dirichlet} \\ a(x_i)\,u'(x_i) & = & g_{N1} + g_{N2}\,u(x_i) & \text{Neumann} \end{array} \tag{86}$$

If the contribution $b(x)\,u'(x)$ vanishes in the ODE (84), solving the ODE is closely related to minimizing the "energy" expression

$$F(u) = \int_{x_0}^{x_n} \frac{1}{2}\,a(x)\,(u'(x))^2 + \frac{1}{2}\,c(x)\,u^2(x) - d(x)\,f(x)\,u(x)\;dx\;, \tag{87}$$

respecting the boundary conditions. For a Neumann condition at the right endpoint add the contribution

$$-g_{N1}\,u(x_n) - \frac{1}{2}\,g_{N2}\,u^2(x_n) \tag{88}$$

to the above functional $F(u)$ and similar at the left endpoint add

$$+g_{N1}\,u(x_0) + \frac{1}{2}\,g_{N2}\,u^2(x_0)\;.$$

Using FEM this equation will be discretized, leading to the global stiffness matrix $\mathbf{A}$ and the global weight matrix $\mathbf{M}$, such that $\langle \mathbf{A}\vec{u} - \mathbf{M}\vec{f}\,,\,\vec{\phi}\rangle = 0$ for all vectors $\vec{\phi}$. This then leads to the linear system $\mathbf{A}\vec{u} = \mathbf{M}\vec{f}$ to be solved for the vector $\vec{u}$.

The corresponding first order dynamic equation is given by

$$w(x)\,\frac{\partial\,u(x,t)}{\partial t} - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t) \tag{89}$$

with the initial condition $u(x,0) = u_0(x)$ and the boundary conditions, either Dirichlet of Neumann.

The initial value problem of order 2 is

$$w_2(x)\,\frac{\partial^2}{\partial t^2}\,u(x,t) + 2\,w_1(x)\,\frac{\partial}{\partial t}\,u(x,t) - \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,t) \tag{90}$$

again with the corresponding boundary and initial conditions.

A nonlinear boundary value problem is of the form

$$-\frac{\partial}{\partial x}\left(a(x)\,\frac{\partial\,u(x,t)}{\partial x}\right) + b(x)\,\frac{\partial\,u(x,t)}{\partial x} + c(x)\,u(x,t) = d(x)\,f(x,u(x),u'(x)) \tag{91}$$

with the corresponding linear boundary conditions.

7.2 Interpolation, Gauss integration and the element stiffness matrices

In a first step the code will extend the provided interval $[x_0, x_1, x_2, \ldots, x_n]$ and add the midpoints $x_{i+0.5} = \frac{x_i + x_{i+1}}{2}$ to the interval, i.e. the new discretization will consist of $2\,n + 1$ points. The nodes are at

$$\vec{x} = [x_0, x_{0.5}, x_1, x_{1.5}, \ldots, x_{n-1}, x_{n-0.5}, x_n] \in \mathbb{R}^{2n+1}\;.$$

To generate a finite element formulation first examine a subinterval $x_i \le x \le x_{i+1}$. The nodes for the FEM algorithm are the two endpoints x_i, x_{i+1} and the midpoint $x_{i+0.5} = \frac{x_i + x_{i+1}}{2}$. For given coefficient functions $a(x)$, $b(x)$, $c(x)$ and $d(x)$ and the values of the functions $u(x)$, $f(x)$ and $\phi(x)$ at the three nodes. Then use a quadratic interpolation to construct the functions $u(x)$, $f(x)$ and $\phi(x)$ on the interval. Four integrals have to be examined.

$$\begin{array}{rclcrcl} I_f & = & \int_{x_i}^{x_{i+1}} d(x)\,f(x)\,\phi(x)\;dr & , & I_0 & = & \int_{x_i}^{x_{i+1}} c(x)\,u(x)\,\phi(x)\;dr\;, \\ I_1 & = & \int_{x_i}^{x_{i+1}} b(x)\,u'(x)\,\phi(x)\;dr & \text{and} & I_2 & = & \int_{x_i}^{x_{i+1}} a(x)\,u'(x)\,\phi'(x)\;dr \end{array}$$

To compute these integrals use the very efficient 3–point Gauss integration on a standard interval $[\frac{-h}{2}\,,\,\frac{+h}{2}]$ of length h.

- On the interval $-\frac{h}{2} \le x \le +\frac{h}{2}$ the 3-point Gauss integration formula is given by

$$\int_{-h/2}^{h/2} u(x)\,dx \approx \frac{h}{18}\left(5\,u(-\sqrt{\frac{3}{5}}\frac{h}{2}) + 8\,u(0) + 5\,u(+\sqrt{\frac{3}{5}}\frac{h}{2})\right). \tag{92}$$

- The three values of a function $u(x)$ at $u(-h/2) = u_-$, $u(0) = u_0$ and $u(h/2) = u_+$ determine a quadratic interpolating polynomial[33]

$$u(x) = u_0 + \frac{u_+ - u_-}{h}\,x + \frac{u_+ - 2\,u_0 + u_-}{h^2}\,2\,x^2\,.$$

Use $x = 0$ and $x = \pm\sqrt{\frac{3}{5}}\,\frac{h}{2}$ to determine the values of $u(x)$ at the Gauss points by[34]

$$\begin{pmatrix} u(-\sqrt{\frac{3}{5}}\frac{h}{2}) \\ u(0) \\ u(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{pmatrix} = \begin{bmatrix} \frac{3}{10} + \frac{\sqrt{\frac{3}{5}}}{2} & \frac{4}{10} & \frac{3}{10} - \frac{\sqrt{\frac{3}{5}}}{2} \\ 0 & 1 & 0 \\ \frac{3}{10} - \frac{\sqrt{\frac{3}{5}}}{2} & \frac{4}{10} & \frac{3}{10} + \frac{\sqrt{\frac{3}{5}}}{2} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}$$

$$= \frac{1}{10}\begin{bmatrix} 3+\sqrt{15} & 4 & 3-\sqrt{15} \\ 0 & 10 & 0 \\ 3-\sqrt{15} & 4 & 3+\sqrt{15} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} = \mathbf{G}_0 \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}.$$

Use this Gaussian interpolation matrix to compute the values of the function at the Gauss integration points, using the values at the nodes.

- The matrix $\mathbf{G}_0$ is basis to construct a matrix to interpolate from the nodes of a FEM grid to the Gauss points. As example consider a grid with three elements of order 2 and thus $3 \cdot 2 + 1 = 7$ nodes and there are $3 \cdot 3 = 9$ Gauss points. Each block in the matrix is given by $\mathbf{G}_0$. This leads to the 9×7 matrix which allows to determine the values of the function at the Gauss points by a matrix multiplication.

$$\begin{pmatrix} u_{1-} \\ u_{10} \\ u_{1+} \\ u_{2-} \\ u_{20} \\ u_{2+} \\ u_{3-} \\ u_{30} \\ u_{3+} \end{pmatrix} = \begin{bmatrix} \frac{3+\sqrt{15}}{10} & \frac{4}{10} & \frac{3-\sqrt{15}}{10} & & & & & & \\ 0 & 1 & 0 & & & & & & \\ \frac{3-\sqrt{15}}{10} & \frac{4}{10} & \frac{3+\sqrt{15}}{10} & & & & & & \\ & & \frac{3+\sqrt{15}}{10} & \frac{4}{10} & \frac{3-\sqrt{15}}{10} & & & & \\ & & 0 & 1 & 0 & & & & \\ & & \frac{3-\sqrt{15}}{10} & \frac{4}{10} & \frac{3+\sqrt{15}}{10} & & & & \\ & & & & \frac{3+\sqrt{15}}{10} & \frac{4}{10} & \frac{3-\sqrt{15}}{10} & & \\ & & & & 0 & 1 & 0 & & \\ & & & & \frac{3-\sqrt{15}}{10} & \frac{4}{10} & \frac{3+\sqrt{15}}{10} & & \end{bmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} \tag{93}$$

Observe that this matrix does not depend on the distances h_i between the nodes.

[33] To verify the formula use $u(0) = u_0$ and for $x = \pm\frac{h}{2}$

$$u(\pm\frac{h}{2}) = u_0 \pm \frac{u_+ - u_-}{h}\,\frac{h}{2} + \frac{u_+ - 2\,u_0 + u_-}{h^2}\,\frac{2\,h^2}{4} = u_0\,(1-1) + u_+\,(\pm\frac{1}{2} + \frac{1}{2}) - u_-\,(\pm\frac{1}{2} - \frac{1}{2})\,.$$

[34] E.g. to obtain the last row in $\mathbf{G}_0$ evaluate $u(+\sqrt{\frac{3}{5}}\,\frac{h}{2})$.

$$\begin{aligned} u(+\sqrt{\frac{3}{5}}\,\frac{h}{2}) &= u_0 + \frac{u_+ - u_-}{h}\,\sqrt{\frac{3}{5}}\,\frac{h}{2} + \frac{u_+ - 2\,u_0 + u_-}{h^2}\,2\,\frac{3}{5}\,\frac{h^2}{2^2} \\ &= \left(-\sqrt{\frac{3}{5}}\,\frac{1}{2} + \frac{3}{10}\right) u_- + \left(1 - \frac{3}{5}\right) u_0 + \left(+\sqrt{\frac{3}{5}}\,\frac{1}{2} + \frac{3}{10}\right) u_+ \\ &= \left(\frac{3}{10} - \frac{\sqrt{\frac{3}{5}}}{2}\right) u_- + \frac{4}{10}\,u_0 + \left(\frac{3}{10} + \frac{\sqrt{\frac{3}{5}}}{2}\right) u_+ \end{aligned}$$

- The above can be repeated to obtain the values of the derivative

$$u'(x) = \frac{u_+ - u_-}{h} + \frac{u_+ - 2\,u_0 + u_-}{h^2}\,4\,x$$

at the Gauss points.

$$\begin{pmatrix} u'(-\sqrt{\frac{3}{5}}\frac{h}{2}) \\ u'(0) \\ u'(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{pmatrix} = \frac{1}{h}\begin{bmatrix} -1-2\sqrt{\frac{3}{5}} & +4\sqrt{\frac{3}{5}} & +1-2\sqrt{\frac{3}{5}} \\ -1 & 0 & +1 \\ -1+2\sqrt{\frac{3}{5}} & -4\sqrt{\frac{3}{5}} & +1+2\sqrt{\frac{3}{5}} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} = \frac{1}{h}\,\mathbf{G}_1 \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} .$$

- Use the same idea as above construct a matrix to evaluate the derivatives at the Gauss points by a matrix multiplication. Each block in this matrix is given by $\frac{1}{h_i}\mathbf{G}_1$.
- Define a weight matrix $\mathbf{W}$ by

$$\mathbf{W} = \mathrm{diag}([\frac{5}{18}, \frac{8}{18}, \frac{5}{18}]) = \begin{bmatrix} \frac{5}{18} & 0 & 0 \\ 0 & \frac{8}{18} & 0 \\ 0 & 0 & \frac{5}{18} \end{bmatrix}$$

and then rewrite the Gauss integration (92) in the form

$$\int_{-h/2}^{+h/2} u(x)\,dx \approx \frac{h}{18}\,\left(5\,u(-\sqrt{\frac{3}{5}}\frac{h}{2}) + 8\,u(0) + 5\,u(+\sqrt{\frac{3}{5}}\frac{h}{2})\right) = h\,\sum_{i=1}^{3}\,(\mathbf{W}\,\mathbf{G_0}\,\vec{u})_i\ ,$$

where the vector $\vec{u}$ contains the values of the function at $\pm\frac{h}{2}$ and 0 .

- To evaluate the function $a(x)$ at the Gauss points use the matrix notation

$$\mathbf{a} = \begin{bmatrix} a(-\sqrt{\frac{3}{5}}\frac{h}{2}) & 0 & 0 \\ 0 & a(0) & 0 \\ 0 & 0 & a(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{bmatrix}$$

and similarly for the functions $b(x)$, $c(x)$ and $d(x)$, leading to the diagonal matrices $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$.

The above notation leads to the required integrals. With $\Delta x_i = x_{i+1} - x_i$ obtain

$$\begin{aligned}
I_f &= \int_{x_i}^{x_{i+1}} -d(x)\,f(x)\,\phi(x)\,dx \;\approx\; \Delta x_i\,\langle \mathbf{W}\,\mathbf{d}\,\mathbf{G}_0\,\vec{f}\,,\,\mathbf{G}_0\,\vec{\phi}\rangle \;=\; -\Delta x_i\,\langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{d}\,\mathbf{G}_0\,\vec{f}\,,\,\vec{\phi}\rangle \\
I_0 &= \int_{x_i}^{x_{i+1}} c(x)\,u(x)\,\phi(x)\,dx \;\approx\; \Delta x_i\,\langle \mathbf{W}\,\mathbf{c}\,\mathbf{G}_0\,\vec{u}\,,\,\mathbf{G}_0\,\vec{\phi}\rangle \;=\; \Delta x_i\,\langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{c}\,\mathbf{G}_0\,\vec{u}\,,\,\vec{\phi}\rangle \\
I_1 &= \int_{x_i}^{x_{i+1}} b(x)\,u'(x)\,\phi(x)\,dx \;\approx\; \frac{\Delta x_i}{\Delta x_i}\,\langle \mathbf{W}\,\mathbf{b}\,\mathbf{G}_1\,\vec{u}\,,\,\mathbf{G}_0\,\vec{\phi}\rangle \;=\; \langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{b}\,\mathbf{G}_1\,\vec{u}\,,\,\vec{\phi}\rangle \\
I_2 &= \int_{x_i}^{x_{i+1}} a(x)\,u'(x)\,\phi'(x)\,dr \;\approx\; \frac{\Delta x_i}{(\Delta x_i)^2}\,\langle \mathbf{W}\,\mathbf{a}\,\mathbf{G}_1\,\vec{u}\,,\,\mathbf{G}_1\,\vec{\phi}\rangle \;=\; \frac{1}{\Delta x_i}\,\langle \mathbf{G}_1^T\mathbf{W}\,\mathbf{a}\,\mathbf{G}_1\,\vec{u}\,,\,\vec{\phi}\rangle
\end{aligned}$$

Using a summation apply the above integrals to the interval $I = [x_0, x_n]$, discretized by $x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n$. For the second order elements the midpoints $x_{i+0.5} = \frac{x_i + x_{i+1}}{2}$ of the intervals will be used too, leading to the nodes at $x_0 < x_{0.5} < x_1 < x_{1.5} < x_2 < \ldots < x_{n-1} < x_{n-0.5} < x_n$, i.e. $\vec{x} \in \mathbb{R}^{2n+1}$. Examine the discrete version of the weak solution, thus integrals (or sums) of the type

$$\begin{aligned}
I &= \int_{x_l}^{x_r} a(x)\,u'(x)\,\phi'(x) + b(x)\,u'(x)\,\phi(x) + c(x)\,u(x)\,\phi(x) - d(x)\,f(x)\,\phi(x)\,dx \\
&= \sum_{i=1}^{n}\int_{x_i}^{x_{i+1}} a(x)\,u'(x)\,\phi'(x) + b(x)\,u'(x)\,\phi(x) + c(x)\,u(x)\,\phi(x) - d(x)\,f)(x)\,\phi(x)\,dx \\
&\approx \sum_{i=1}^{n}\Big(\frac{1}{\Delta x_i}\,\langle \mathbf{G}_1^T\mathbf{W}\,\mathbf{a}_i\,\mathbf{G}_1\,\vec{u}_i\,,\,\vec{\phi}_i\rangle + \langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{b}_i\,\mathbf{G}_1\,\vec{u}_i\,,\,\vec{\phi}_i\rangle + \\
&\qquad + \Delta x_i\,\langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{c}_i\,\mathbf{G}_0\,\vec{u}_i\,,\,\vec{\phi}_i\rangle - \Delta x_i\,\langle \mathbf{G}_0^T\mathbf{W}\,\mathbf{d}_i\,\mathbf{G}_0\,\vec{f}_i\,,\,\vec{\phi}_i\rangle\Big) \\
&= \langle \mathbf{A}\,\vec{u} - \mathbf{M}\,\vec{f}\,,\,\vec{\phi}\rangle \quad \text{for all vectors} \quad \vec{\phi} \in \mathbb{R}^{2n+1}\,.
\end{aligned}$$

The stiffness matrix $\mathbf{A}$ and the weight matrix $\mathbf{M}$ are both of size $(2\,n+1)\times(2\,n+1)$, but the boundary conditions are not taken into account yet. This has to be done with some care, since the differential equation (84) has a unique solution only if boundary conditions are taken into account.

7.3 Taking boundary conditions into account

Ignoring the boundary conditions the linear system to be solved for $\vec{u}\in\mathbb{R}^{2n+1}$ is

$$\mathbf{A}\,\vec{u}=\mathbf{M}\,\vec{f}\,, \tag{94}$$

where $\vec{f}\in\mathbb{R}^{2n+1}$ contains the vaues of $f(x)$ at the nodes $\vec{x}\in\mathbb{R}^{2n+1}$. The contribution in (85) by boundary terms is

$$a(x)\,u'(x)\,\phi(r)\Big|_{x=x_0}^{x_n}=a(x_n)\,u'(x_n)\,\phi(x_n)-a(x_0)\,u'(x_0)\,\phi(x_0)\,.$$

This leads to different algorithms to take Dirichlet or Neumann conditions into account. There are four possible combinations of Dirichlet (D) and Neumann (N) boundary conditions: DD, DN, ND and NN.

DD Dirichlet conditions at both endpoints.
The first component of the vector $\vec{u}$ equals $u(x_0)=g_{D1}$ and the for the last component use $u(x_n)=g_{D2}$. Equation (94) reads as

$$\mathbf{A}\begin{pmatrix} g_{D1}\\ u_{0.5}\\ u_1\\ \vdots\\ u_{2n-1}\\ u_{2n-0.5}\\ g_{D2}\end{pmatrix}=\mathbf{M}\begin{pmatrix} f_0\\ f_{0.5}\\ f_1\\ \vdots\\ f_{n-1}\\ f_{n-0.5}\\ f_n\end{pmatrix}$$

Remove the first and last row in the matrix $\mathbf{A}$. Split off the first column $\vec{a}_f$ and the last column $\vec{a}_l$ form the matrix $[\vec{a}_f,\mathbf{A}_r,\vec{a}_l]$ and in the matrix $\mathbf{M}\in\mathbb{R}^{(2n+1)\times(2n+1)}$ remove the first and last row, leading to $\mathbf{M}_r\in\mathbb{R}^{(2n-1)\times(2n+1)}$. Then examine

$$\mathbf{A}\vec{u}=\mathbf{M}\vec{f}\quad\longrightarrow\quad\begin{bmatrix}\vec{a}_f & \mathbf{A}_r & \vec{a}_l\end{bmatrix}\begin{pmatrix} g_{D1}\\ \vec{u}_r\\ g_{D2}\end{pmatrix}=\mathbf{M}_r\,\vec{f}\quad\longrightarrow\quad\mathbf{A}_r\,\vec{u}_r=\mathbf{M}_r\,\vec{f}-\vec{a}_f\,g_{D1}-\vec{a}_l\,g_{D2}\,,$$

where $\mathbf{A}_r\in\mathbb{R}^{(2n-1)\times(2n-1)}$, $\vec{u}_r\in\mathbb{R}^{2n-1}$, $\mathbf{M}_r\in\mathbb{R}^{(2n-1)\times(2n+1)}$ and $\vec{f}\in\mathbb{R}^{2n+1}$. The solution is then given by $[g_{D1},\vec{u}_r,g_{D2}]\in\mathbb{R}^{2n+1}$.

DN Dirichlet at x_0 and Neumann at x_n
The Dirichlet condition $u(x_0)=g_{D1}$ leads to the modifications of the first rows and columns of $\mathbf{A}$ and $\mathbf{M}$, just as above. For the boundary condition $a(x_n)\,u'(x_n)=g_{N1}+g_{N2}\,u(x_n)$ the correct type of contributions will have to be taken into account. The additional contributions in (88) to the functional $F(u)$ in (87) lead to

$$\mathbf{A}\begin{pmatrix} g_{D1}\\ u_{0.5}\\ u_1\\ \vdots\\ u_{n-1}\\ u_{n-0.5}\\ u_n\end{pmatrix}-\begin{pmatrix} 0\\ 0\\ 0\\ \vdots\\ 0\\ 0\\ g_{N2}\,u_n\end{pmatrix}=\mathbf{M}\begin{pmatrix} f_0\\ f_{0.5}\\ f_1\\ \vdots\\ f_{n-1}\\ f_{n-0.5}\\ f_n\end{pmatrix}+\begin{pmatrix} 0\\ 0\\ 0\\ \vdots\\ 0\\ 0\\ g_{N1}\end{pmatrix}.$$

Thus the matrix $\mathbf{A}$ and the RHS have to be modified accordingly. From the last entry on the diagonal of the matrix $\mathbf{A}$ subtract g_{N2}.

$$\begin{bmatrix}\vec{a}_f & \mathbf{A}_r\end{bmatrix}\begin{pmatrix} g_{D1}\\ \vec{u}_r\end{pmatrix}=\mathbf{M}_r\,\vec{f}+\vec{g}_{N1}\quad\longrightarrow\quad\mathbf{A}_r\,\vec{u}_r=\mathbf{M}_r\,\vec{f}-\vec{a}_f\,g_{D1}+\vec{g}_{N1}\,,$$

where $\mathbf{A}_r \in \mathbb{R}^{2n\times 2n}$, $\vec{u}_r \in \mathbb{R}^{2n}$, $\mathbf{M}_r \in \mathbb{R}^{2n\times(2n+1)}$ and $\vec{f} \in \mathbb{R}^{2n+1}$. The solution is then given by $[g_{D1}, \vec{u}_r] \in \mathbb{R}^{2n+1}$.

ND Neumann at x_0 and Dirichlet at x_n
The Dirichlet condition $u(x_n) = g_{D2}$ leads to the modifications of the last rows and columns, just as above. For the boundary condition $a(x_0)\,u'(x_0) = g_{N1} + g_{N2}\,u(x_0)$ the correct type of contributions will have to be taken into account. The additional contributions in (88) to the functional $F(u)$ leads to

$$\mathbf{A}\begin{pmatrix} u_0 \\ u_{0.5} \\ u_1 \\ \vdots \\ u_{n-0.5} \\ g_{D2} \end{pmatrix} + \begin{pmatrix} g_{N2}\,u_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} = \mathbf{M}\begin{pmatrix} f_0 \\ f_{0.5} \\ f_1 \\ \vdots \\ f_{n-0.5} \\ f_n \end{pmatrix} - \begin{pmatrix} g_{N1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}.$$

Observe the different signs of the Neumann contribution. Thus the matrix $\mathbf{A}$ and the RHS have to be modified accordingly. To the first entry on the diagonal of the matrix $\mathbf{A}$ add g_{N2}.

$$\begin{bmatrix} \mathbf{A}_r & \vec{a}_l \end{bmatrix}\begin{pmatrix} \vec{u}_r \\ g_{D2} \end{pmatrix} = \mathbf{M}_r\,\vec{f} + \vec{g}_{N1} \quad \longrightarrow \quad \mathbf{A}_r\,\vec{u}_r = \mathbf{M}_r\,\vec{f} - \vec{a}_l\,g_{D1} - \vec{g}_{N1}\ ,$$

where $\mathbf{A}_r \in \mathbb{R}^{2n\times 2n}$, $\vec{u}_r \in \mathbb{R}^{2n}$, $\mathbf{M}_r \in \mathbb{R}^{2n\times(2n+1)}$ and $\vec{f} \in \mathbb{R}^{2n+1}$. The solution is then given by $[\vec{u}_r, g_{D2}] \in \mathbb{R}^{2n+1}$.

NN Neumann conditions at both endpoints.
The modifications are given by

$$\mathbf{A}\begin{pmatrix} u_0 \\ u_{0.5} \\ u_1 \\ \vdots \\ u_{n-0.5} \\ u_n \end{pmatrix} + \begin{pmatrix} +g_{N2\_left}\,u_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -g_{N2\_right}u_n \end{pmatrix} = \mathbf{M}\begin{pmatrix} f_0 \\ f_{0.5} \\ f_1 \\ \vdots \\ f_{n-0.5} \\ f_n \end{pmatrix} + \begin{pmatrix} -g_{N1\_left} \\ 0 \\ 0 \\ \vdots \\ 0 \\ +g_{N1\_right} \end{pmatrix}.$$

The size of the matrices $\mathbf{A}$ and $\mathbf{M}$ remains unchanged. To the first entry on the diagonal of the matrix g$\mathbf{A}$ add g_{N2_left} and from the last entry subtract g_{N2_right}.

$$\mathbf{A}_r\,\vec{u} = \mathbf{M}\,\vec{f} + \vec{g}_{N1},$$

where $\mathbf{A}_r \in \mathbb{R}^{(2n+1)\times(2n+1)}$, $\vec{u} \in \mathbb{R}^{2n+1}$, $\mathbf{M} \in \mathbb{R}^{(2n+1)\times(2n+1)}$ and $\vec{f} \in \mathbb{R}^{2n+1}$.

7.4 Solving the BVP with a system of linear equations

Using the above algorithms leads to the linear system

$$\mathbf{A}_r\,\vec{u}_r - \mathbf{M}_r\,\vec{f} = \vec{0} \quad \text{or} \quad \vec{u} = \mathbf{A}_r^{-1}\,\mathbf{M}_r\,\vec{f} = \mathbf{A}_r \backslash \mathbf{M}_r\,\vec{f}.$$

The resulting matrix $\mathbf{A}_r$ is symmetric if $b(x)$ vanishes identically and it has a band structure with semi-bandwidth 3, i.e. in each row there are up to 5 nonzero entries around the diagonal.

7.5 Evaluation of the solution between nodes and evaluation of derivatives

The above algorithms lead to the values of the solution at the nodes. Between nodes the algorithm is based on piece–wise quadratic interpolation. Thus it is advisable to use a quadratic interpolation if the values at more points are required. The *Octave* function `pwquadinterp()` does just that, and it can be used to evaluate first and second derivatives too.

To evaluate the derivatives at the nodes use the quadratic interpolation. For a quadratic function with $u_- = u(-\frac{h}{2})$, $u(0) = u_0$ and $u_+ = u(+\frac{h}{2})$ the derivative

$$u'(x) = \frac{u_+ - u_-}{h} + \frac{u_+ - 2\,u_0 + u_-}{h^2}\,4\,x$$

leads to values at the end- and mid–points.

$$\begin{aligned} u'(-\frac{h}{2}) &= \frac{u_+ - u_-}{h} - \frac{u_+ - 4\,u_0 + u_-}{h^2}\,2\,h = \frac{1}{h}\,(-3\,u_- + 2\,u_0 - 1\,u_+) \\ u'(0) &= \frac{1}{h}\,(-1\,u_- + 0\,u_0 + 1\,u_+) \\ u'(+\frac{h}{2}) &= \frac{u_+ - u_-}{h} + \frac{u_+ - 4\,u_0 + u_-}{h^2}\,2\,h = \frac{1}{h}\,(+1\,u_- - 2\,u_0 + 3\,u_+) \end{aligned}$$

With a matrix notation write this in the form

$$\begin{pmatrix} u'(-\frac{h}{2}) \\ u'(0) \\ u'(+\frac{h}{2}) \end{pmatrix} = \frac{1}{h} \begin{bmatrix} -3 & +4 & -1 \\ -1 & 0 & +1 \\ +1 & -4 & +3 \end{bmatrix} \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} .$$

The second derivative is constant on each subinterval and given by

$$u''(x) = 4\,\frac{u_- - 2\,u_0 + u_+}{h^2} .$$

This is used in the code `FEM1DEvaluateDu()` to determine the derivatives at the nodes, generated by `BVP1D()`. At the endpoints of the subintervals the average of derivatives in the two neighboring intervals is used.

FEM1DEvaluateDu.m

```
function [du,ddu] = FEM1DEvaluateDu(x,u)
%% [du,ddu] = FEM1DEvaluateDu(x,u)
%% evaluate first and second derivatives at the nodes x
%% requires the interval x and u to be generated by BVP1D()
  n = (length(x)-1)/2;  %% number of subintervals
  du = zeros(size(x)); ddu = du;
  M = 0.5*[-3,4,-1;-2,0,2;1,-4,3];  %% matrix to determine first derivatives
  h = diff(x); h = h(1:2:end)*2;
  for jj = 1:n
    range = [2*jj-1:2*jj+1];
    du(range)  += M*u(range)/h(jj);
    ddu(range) += 2*[1;2;1]*[1 -2 1]*u(range)/h(jj)^2;
  endfor
  du([1;end]) *= 2; ddu([1;end]) *= 2;
endfunction
```

7.6 The first order dynamic problem

The approach to solve the dynamic problem (89)

$$w(x)\,\frac{\partial}{\partial t}\,u(x,t) - (a(x)\,u'(x,t))' + b(x)\,u'(x,t) + c(x)\,u(x,t) = d(x)\,f(x,t)$$

is similar to the 2D approach presented in Section 6.10.1. The first step is to solve the static problem

$$-\,(a(x)\,u'_B(x))' + b(x)\,u'_B(x) + c(x)\,u_B(x) = 0 \quad \text{with} \quad u_B(x_i) = g_D \quad \text{or} \quad a(x_i)\,u'_B(x_i) = g_{N1} + g_{N2}\,u_B(x_i)$$

Then the new function $v(x,t) = u(x,t) - u_B(x)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. at the boundary points x_0 and x_n

$$v(x_i,t) = 0 \quad \text{or} \quad a(x_i)\,v'(x_i,t) = 0 + g_{N2}(x_i)\,v(x_i,t)$$

and the initial condition $v(x,0) = u_0(x) - u_B(x)$. This problem is then discretized with respect to x, leading to a system of ordinary differential equations, similar to equation (79)

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \mathbf{M}\,\vec{f}(t) \quad \text{with} \quad \vec{v}(t_0) = \vec{v}_0 \tag{95}$$

There are many algorithms available to perform a time step from $\vec{v}(t)$ to $\vec{v}(t+\Delta t)$. Table 21 shows a few key properties of the solvers used in `IBVP1D()` and `IBVP2D()`.

- order of consistency: order of consistency of the time stepping algorithm.
- A–stability: a solver is called A–stable if for ODEs of the type $\frac{d}{dt}\,\vec{u}(t) = \mathbf{A}\,\vec{u}(t)$ with negative eigenvalues the approximate solutions remain bounded, independent on the stepsize Δt.
- L–stability: a solver is called L–stable if the above approximate solutions converge to zero rapidly for large eigenvalues $\lambda < 0$.[35]
- number (#) of systems to solve: the number of linear systems that have to be solved for one time step.

algorithm	order of consistency	A–stability	L–stability	# of systems to solve
explicit	1	conditional	no	1
implicit	1	unconditional	yes	1
Crank–Nicolson	2	unconditional	no	1
Runge–Kutta, L–stable	2	unconditional	yes	2

Table 21: Properties of the ODE solvers used in `IBVP1D()`

`FEMoctave` uses very simple implementations of the time steppers, i.e. no stepsize control or adaptation is performed. The stepsize Δt is constant on the time interval to be examined. One can specify the number of time slices to be returned as result and how many steps to take between the returned times. For the explicit time stepper a warning is issued if the algorithm is very likely to be unstable, i.e. when $\Delta t > \frac{2}{\lambda_{max}}$. This authors advise is to **not use** the explicit time stepper.

7.6.1 An explicit time step

To solve (95) with an explicit solver with time step Δt use

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t)}{\Delta t} + \mathbf{A}\,\vec{v}(t) &= \mathbf{M}\,\vec{f}(t) \\
\mathbf{W}\,\vec{v}(t+\Delta t) &= \mathbf{W}\,\vec{v}(t) + \Delta t\,\left(-\mathbf{A}\,\vec{v}(t) + \mathbf{M}\,\vec{f}(t)\right) \\
\vec{v}(t+\Delta t) &= \vec{v}(t) + \Delta t\,\mathbf{W}^{-1}\left(-\mathbf{A}\,\vec{v}(t) + \mathbf{M}\,\vec{f}(t)\right)\,.
\end{aligned}$$

[35] For an ODE $\dot{u}(t) = -\lambda\,u(t)$ with exact solution $u(t) = c\,\exp(-\lambda\,t)$ applying one time step is implemented by multiplying with a factor $g(\lambda\,\Delta t) = g(z)$. The function $g(z)$ depends on the algorithm.

- explicit: $\frac{u_{i+1}-u_i}{\Delta t} = -\lambda\,u_i$, thus $u_{i+i} = (1-\lambda\,\Delta t)\,u_i$ and $g(z) = 1-z$. The stability condition $|g(z)| < 1$ leads to $z = \lambda\,\Delta t < 2$ or $\Delta t < \frac{2}{\lambda_{max}}$.
- implicit: $\frac{u_{i+1}-u_i}{\Delta t} = -\lambda\,u_{i+1}$, thus $u_{i+i} = \frac{1}{1+\lambda\,\Delta t}\,u_i$ and $g(z) = \frac{1}{1+z}$.
- Crank–Nicolson: $\frac{u_{i+1}-u_i}{\Delta t} = -\lambda\,\frac{u_{i+1}+u_i}{2}$, thus $u_{i+i} = \frac{2-\lambda\,\Delta t}{2+\lambda\,\Delta t}\,u_i$ and $g(z) = \frac{2-z}{2+z}$.
- L–stable Runge–Kutta: $g(z) \approx -\frac{4.83}{z}$ for $z \gg 1$. For details see [Stah08, §4.5.8].

If $|g(z)| < 1$ for $\mathrm{Re}(z) > 0$, then the algorithm is A–stable. If $\lim_{z\to+\infty} |g(z)| = 0$, then the algorithm is L–stable.

7.6.2 An implicit time step

To solve (95) with an implicit solver with time step Δt use

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}(t+\Delta t)-\vec{v}(t)}{\Delta t}+\mathbf{A}\,\vec{v}(t+\Delta t) &= \mathbf{M}\,\vec{f}(t+\Delta t)\\
(\mathbf{W}+\Delta t\mathbf{A})\,\vec{v}(t+\Delta t) &= \mathbf{W}\,\vec{v}(t)+\Delta t\,\mathbf{M}\,\vec{f}(t+\Delta t)\\
\vec{v}(t+\Delta t) &= (\mathbf{W}+\Delta t\mathbf{A})^{-1}\left(\mathbf{W}\,\vec{v}(t)+\Delta t\,\mathbf{M}\,\vec{f}(t+\Delta t)\right)\,.
\end{aligned}$$

7.6.3 A Crank–Nicolson time step

In Section 6.10.1 (page 203) the algorithm of Crank–Nicolson is presented as time stepper. With the above notation a time step is given by

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}(t+\Delta t)-\vec{v}(t)}{\Delta t}+\mathbf{A}\,\frac{\vec{v}(t+\Delta t)+\vec{v}(t)}{2} &= \mathbf{M}\,\vec{f}(t+\Delta t/2)\\
\left(\mathbf{W}+\frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}(t+\Delta t) &= +\left(\mathbf{W}-\frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}(t)+\Delta t\,\mathbf{M}\,\vec{f}(t+\Delta t/2)\,.
\end{aligned}$$

This is the default time stepper used in `FEMoctave`.

7.6.4 An L–stable Runge–Kutta solver, DIRK

This is a *diagonally implicit Runge–Kutta* method or a DIRK method. Find more details in [Butc03, §361]. The Butcher table of the algorithm is given by

$$\begin{array}{c|cc}
\theta & \theta & 0\\
1 & 1-\theta & \theta\\
\hline
y_{n+1} & 1-\theta & \theta
\end{array}
\qquad \text{with } \theta = 1-\tfrac{1}{\sqrt{2}}\text{, i.e.} \qquad
\begin{array}{c|cc}
1-\frac{1}{\sqrt{2}} & 1-\frac{1}{\sqrt{2}} & 0\\
1 & \frac{1}{\sqrt{2}} & 1-\frac{1}{\sqrt{2}}\\
\hline
y_{n+1} & \frac{1}{\sqrt{2}} & 1-\frac{1}{\sqrt{2}}
\end{array}\,.$$

To apply this algorithm to the ODE $\mathbf{W}\frac{d}{dt}\,\vec{u}(t) = -\mathbf{A}\,\vec{u}(t)+\mathbf{M}\,\vec{f}(t)$ use the above Butcher table.

$$\begin{aligned}
\mathbf{W}\,\vec{k}_1 &= -\mathbf{A}\,(\vec{u}_n+\theta\,\Delta t\,\vec{k}_1)+\mathbf{M}\,\vec{f}(t_n+\theta\,\Delta t)\\
\mathbf{W}\,\vec{k}_2 &= -\mathbf{A}\,(\vec{u}_n+\Delta t\,((1-\theta)\,\vec{k}_1+\theta\,\vec{k}_2))+\mathbf{M}\,\vec{f}(t_n+1\,\Delta t)\\
\vec{u}_{n+1} &= \vec{u}_n+\Delta t\,((1-\theta)\,\vec{k}_1+\theta\,\vec{k}_2)
\end{aligned}$$

With $\frac{1}{2}-\theta = \theta\,(1-\theta) = \frac{\theta}{\sqrt{2}}$ and tedious algebra (spelled out in [Stah08, §4.5.8]) this leads to two linear systems to be solved, with the same matrix $\mathbf{W}+\theta\,\Delta t\,\mathbf{A}$. Thus only one LU factorization is necessary for the time stepper.

$$(\mathbf{W}+\theta\,\Delta t\,\mathbf{A})\,\vec{k}_1 = -\mathbf{A}\,\vec{u}_n+\mathbf{M}\,\vec{f}(t_n+\theta\,\Delta t) \tag{96}$$

$$\begin{aligned}
(\mathbf{W}+\theta\,\Delta t\,\mathbf{A})\,\vec{u}_{n+1} &= (\mathbf{W}-\Delta t\,\frac{1}{\sqrt{2}}\,\mathbf{A})\,\vec{u}_n-(\Delta t)^2\,(\frac{1}{2}-\theta)\,\mathbf{A}\,\vec{k}_1+\\
&\quad +\Delta t\,\mathbf{M}\,\left((1-\theta)\,\vec{f}(t_n+\theta\,\Delta t)+\theta\,\vec{f}(t_n+\Delta t)\right)
\end{aligned} \tag{97}$$

7.6.5 A Crank–Nicolson solver for semilinear dynamic problems

A semilinear initial boundary value problem with one space dimension is given by

$$w(x)\,\frac{\partial}{\partial t}\,u(x,t)-(a(x)\,u'(x,t))'+b(x)\,u'(x,t)+c(x)\,u(x,t) = d(x)\,f(x,t,u(x,t)) \tag{98}$$

For the function f on the right hand side the partial derivative with respect to u has to be known and a linear approximation is used during the time steppers[36]. Use the linear approximation

$$f_u(x,t,u) = \frac{\partial}{\partial u}\,f_u(x,t,u) \quad \text{and} \quad f(x,t,u+\phi)\approx f(x,t,u)+f_u(x,t,u)\,\phi\,.$$

[36] The algorithm could also be implemented for functions $f(x,t,u,u')$ depending on the derivative u'. Since I did not find good examples this is not done (yet).

The first step is to solve

$$w(x)\,\frac{\partial}{\partial t}\,u_B(x,t) - (a(x)\,u_B'(x,t))' + b(x)\,u_B'(x,t) + c(x)\,u_B(x,t) = 0$$

with the possibly nonhomogenous boundary conditions. Then the function $v(x,t) = u(x,t) - u_B(x)$ satisfies zero boundary conditions and is a solution of

$$w(x)\,\frac{\partial}{\partial t}\,v(x,t) - (a(x)\,v'(x,t))' + b(x)\,v'(x,t) + c(x)\,v(x,t) = d(x)\,f(x,t,v(x,t)+u_B(x),u'(x,t)+u_B'(x)) \quad (99)$$

Discretizing with respect to the space variable x and a standard Crank–Nicolson time stepper for the system of ODEs

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \mathbf{M}\,\vec{f}(t,\vec{v}(t)+\vec{u}_B)$$

leads to a nonlinear system of equations for each time step $\vec{v}(t) = \vec{v}_n \;\rightarrow\; \vec{v}(t+\Delta t) = \vec{v}_{n+1}$.

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}_{n+1}-\vec{v}_n}{\Delta t} + \mathbf{A}\,\frac{\vec{v}_{n+1}+\vec{v}_n}{2} &= \frac{1}{2}\,\mathbf{M}\,(\vec{f}(t,\vec{v}_n+\vec{u}_B) + \vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B)) \\
\mathbf{W}\,\vec{v}_{n+1} + \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_{n+1} - \frac{\Delta t}{2}\,\mathbf{M}\,(\vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B)) &= +\mathbf{W}\,\vec{v}_n - \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_n + \frac{\Delta t}{2}\,\mathbf{M}\,\vec{f}(t,\vec{v}_n+\vec{u}_B) =: \mathbf{G}(\vec{\mathbf{v}}_\mathbf{n}) \\
\vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B+\vec{\phi}) &\approx \vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B) + \vec{f}_u(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B)\,\vec{\phi} \\
\mathbf{W}\,\vec{\phi} + \frac{\Delta t}{2}\,\mathbf{A}\,\vec{\phi} - \frac{\Delta t}{2}\,\mathbf{M}\,f_u(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B)\,\vec{\phi} &= \mathbf{G}(\vec{\mathbf{v}}_\mathbf{n}) - \mathbf{W}\,\vec{v}_{n+1} - \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_{n+1} + \\
&\quad + \frac{\Delta t}{2}\,\mathbf{M}\,\vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A} - \frac{\Delta t}{2}\,\mathbf{M}\,f_u(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B)\right)\vec{\phi} &= \mathbf{G}(\vec{\mathbf{v}}_\mathbf{n}) - \mathbf{W}\,\vec{v}_{n+1} - \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_{n+1} + \\
&\quad + \frac{\Delta t}{2}\,\mathbf{M}\,\vec{f}(t+\Delta t,\vec{v}_{n+1}+\vec{u}_B) \\
\vec{v}_{n+1} &\rightarrow \vec{v}_{n+1} + \vec{\phi} \quad \text{and iterate until } \|\vec{\phi}\| \text{ small}
\end{aligned}$$

As initial guess for $\vec{v}_{n+1}$ the previous solution $\vec{v}_n$ is used. For small time steps this is very good approximation and thus Newton's method should converge after very few steps. This algorithm is implemented as default in `IBVP1DNL()`.

7.6.6 Two semi–explicit solvers for semilinear dynamic problems

To solve the initial boundary value problem (98) reduce the PDE to a system of ODEs with the help of finite elements. Examine

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \mathbf{M}\,\vec{f}(t,\vec{v}(t)+\vec{u}_B)$$

- Use a standard Crank–Nicolson approach for the linear contribution and an explicit step for the nonlinear contribution. This leads to a linear system of equations for each time step $\vec{v}(t) = \vec{v}_n \;\rightarrow\; \vec{v}(t+\Delta t) = \vec{v}_{n+1}$. The matrix for the linear system to be solved does not change from step to step, thus only one LU factorization is required.

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}_{n+1}-\vec{v}_n}{\Delta t} + \mathbf{A}\,\frac{\vec{v}_{n+1}+\vec{v}_n}{2} &= \mathbf{M}\,\vec{f}(t+\frac{1}{2}\,\Delta t,\vec{v}_n+\vec{u}_B) \\
\mathbf{W}\,\vec{v}_{n+1} + \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_{n+1} &= +\mathbf{W}\,\vec{v}_n - \frac{\Delta t}{2}\,\mathbf{A}\,\vec{v}_n + \Delta t\,\mathbf{M}\,\vec{f}(t+\frac{1}{2}\,\Delta t,\vec{v}_n+\vec{u}_B) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_{n+1} &= \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_n + \Delta t\,\mathbf{M}\,\vec{f}(t+\frac{1}{2}\,\Delta t,\vec{v}_n+\vec{u}_B)
\end{aligned}$$

  This is the algorithm CNEXP, short for Crank–Nicolson explicit. This algorithm is convergent of order 1 .

- In addition a predictor–corrector step can be used, i.e.

$$\begin{aligned}
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_{temp} &= \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_n + \Delta t\,\mathbf{M}\,\vec{f}(t,\vec{v}_n+\vec{u}_B) \\
\mathbf{W}\,\frac{\vec{v}_{n+1}-\vec{v}_n}{\Delta t} + \mathbf{A}\,\frac{\vec{v}_{n+1}+\vec{v}_n}{2} &= \frac{\Delta t}{2}\,\mathbf{M}\,\left(\vec{f}(t,\vec{v}_n+\vec{u}_B) + \vec{f}(t+\Delta t,\vec{v}_{temp}+\vec{u}_B)\right) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_{n+1} &= \left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\vec{v}_n + \frac{\Delta t}{2}\,\mathbf{M}\,\left(\vec{f}(t,\vec{v}_n+\vec{u}_B) + \vec{f}(t+\Delta t,\vec{v}_{temp}+\vec{u}_B)\right)
\end{aligned}$$

This is the algorithm CNPC, short for Crank–Nicolson, predictor–corrector. This algorithm is convergent of order 2 .

Both algorithms are used for the commands `IBVP1DNL()` and `IBVP2DNL()`. Observe that both algorithms are not unconditionally stable. A 2D heat equation is used in Section 5.10 (page 138) to illustrate the convergence behavior of these two algorithms.

7.7 The second order dynamic problem

The approach to solve the dynamic problem (90)

$$w_2(x)\,\frac{\partial^2}{\partial t^2}\,u(x,t) + 2\,w_1(x)\,\frac{\partial}{\partial t}\,u(x,t) - \left(a(x)\,u'(x,t)\right)' + b(x)\,u'(x,t) + c(x)\,u(x,t) = d(x)\,f(x,t)$$

is very similar to the 2D approach presented in Section 6.10.4. The first step is to solve the static problem

$$-\left(a(x)\,u'_B(x)\right)' + b(x)\,u'_B(x) + c(x)\,u_B(x) = 0 \quad \text{with} \quad u_B(x_i) = g_D \quad \text{or} \quad a(x_i)\,u'_B(x_i) = g_{N1} + g_{N2}\,u_B(x_i)$$

Then the new function[37] $v(x,t) = u(x,t) - u_B(x)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. at the boundary points x_i

$$v(x_i,t) = 0 \quad \text{or} \quad a(x_i)\,v'(x_i,t) = 0 + g_{N2}(x_i)\,v(x_i,t)$$

and the initial condition $v(x,t_0) = u_0(x) - u_B(x)$. This problem is then discretized with respect to x, leading to a system of ordinary differential equations or order 2, similar to equation (81).

$$\mathbf{W}_2\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,\mathbf{W}_1\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \mathbf{M}\,\vec{f}(t) \quad \text{with} \quad \vec{v}(t_0) = \vec{u}_0 - \vec{u}_B \quad \text{and} \quad \frac{d}{dt}\,v(t_0) = \vec{v}_0 \tag{100}$$

For this second order system of ordinary differential equations two algorithms are examined: an implicit and an explicit solver.

7.7.1 An implicit solver

The implementation assumes that the coefficient functions w_2, w_1, a, b, c and d depend x only, while f may depend on time t and position x. Then use an implicit approximation to advance the solution from time $t-\Delta t$ and t to $t+\Delta t$.

$$\begin{aligned}
\mathbf{W}_2\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -2\,\mathbf{W}_1\,\frac{d}{dt}\,\vec{v}(t) - \mathbf{A}\,\vec{v}(t) + \mathbf{M}\,\vec{f}(t) \\
\mathbf{W}_2\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -2\,\mathbf{W}_1\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t-\Delta t)}{2\,\Delta t} - \\
&\quad -\mathbf{A}\,\frac{\vec{v}(t-\Delta t) + 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{4} + \mathbf{M}\,\vec{f}(t) \\
\left(+\mathbf{W}_2 + \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= -\left(\mathbf{W}_2 - \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t) + \\
&\quad + \left(2\,\mathbf{W}_2 - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t) + (\Delta t)^2\,\mathbf{M}\,\vec{f}(t)
\end{aligned}$$

This scheme is unconditionally stable and consistent of order 2. Observe that the matrices do not change as time advances. Thus use again a sparsity preserving LU factorization for the time stepping.

To construct the solution at the initial time $t_0 + \Delta t$ use the initial value u_0 and initial velocity v_0 and a scheme with the same order of consistency, with respect to time. For sake of a shorter notation the derivation uses $t_0 = 0$.

$$\begin{aligned}
\frac{d}{dt}\,\vec{v}(0) &= \vec{v}_0 \approx \frac{\vec{v}(\Delta t) - \vec{v}(-\Delta t)}{2\,\Delta t} \quad \Longrightarrow \quad \vec{v}(-\Delta t) \approx \vec{v}(\Delta t) - 2\,\Delta t\,\vec{v}_0 \\
\left(+\mathbf{W}_2 + \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(+\Delta t) &= -\left(\mathbf{W}_2 - \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(-\Delta t) + \\
&\quad + \left(2\,\mathbf{W}_2 - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(0) + (\Delta t)^2\,\mathbf{M}\,\vec{f}(0)
\end{aligned}$$

[37] Observe that $v(x,t)$ is **not** the velocity, sorry for the inconvenient notation, but it is consistent with other parts of these notes.

$$
\begin{aligned}
&= -\left(\mathbf{W}_2 - \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,(\vec{v}(+\Delta t) - 2\,\Delta t\,\vec{v}_0) + \\
&\quad + \left(2\,\mathbf{W}_2 - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(0) + (\Delta t)^2\,\mathbf{M}\,\vec{f}(0) \\
2\left(+\mathbf{W}_2 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(+\Delta t) &= +2\,\Delta t\left(\mathbf{W}_2 - \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}_0 + \\
&\quad +2\left(\mathbf{W}_2 - \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(0) + (\Delta t)^2\,\mathbf{M}\,\vec{f}(0) \\
\left(+\mathbf{W}_2 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(+\Delta t) &= \Delta t\left(\mathbf{W}_2 - \Delta t\,\mathbf{W}_1 + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}_0 + \\
&\quad + \left(\mathbf{W}_2 - \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(0) + \frac{(\Delta t)^2}{2}\,\mathbf{M}\,\vec{f}(0)\;.
\end{aligned}
$$

If there is no damping ($\mathbf{W}_1 = \mathbf{0}$) a linear system of the same type as for the time stepper has to be solved.

7.7.2 An explicit solver

To construct an explicit solver for the system of ODEs (100)

$$\mathbf{W}_2\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,\mathbf{W}_1\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \mathbf{M}\,\vec{f}(t)$$

use the finite difference approximations

$$
\begin{aligned}
\mathbf{W}_2\,\frac{d^2}{dt^2}\,\vec{v}(t) &\approx \mathbf{W}_2\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} \\
\mathbf{W}_1\,\frac{d}{dt}\,\vec{v}(t) &\approx \mathbf{W}_1\,\frac{-\vec{v}(t-\Delta t) + \vec{v}(t+\Delta t)}{2\,\Delta t}\;.
\end{aligned}
$$

These approximations lead to

$$\left(\frac{\mathbf{W}_2}{(\Delta t)^2} - \frac{\mathbf{W}_1}{2\,\Delta t}\right)\,\vec{v}(t-\Delta t) + \left(-\frac{2\,\mathbf{W}_2}{(\Delta t)^2} + \mathbf{A}\right)\,\vec{v}(t) + \left(\frac{\mathbf{W}_2}{(\Delta t)^2} + \frac{\mathbf{W}_1}{2\,\Delta t}\right)\,\vec{v}(t+\Delta t) = \mathbf{M}\,\vec{f}(t)\;.$$

For given $\vec{v}(t-\Delta t$ and $\vec{v}(t)$ solve for $\vec{v}(t+\Delta t)$.

$$
\begin{aligned}
\left(\frac{\mathbf{W}_2}{(\Delta t)^2} + \frac{\mathbf{W}_1}{2\,\Delta t}\right)\,\vec{v}(t+\Delta t) &= \left(\frac{2\,\mathbf{W}_2}{(\Delta t)^2} - \mathbf{A}\right)\,\vec{v}(t) - \left(\frac{\mathbf{W}_2}{(\Delta t)^2} - \frac{\mathbf{W}_1}{2\,\Delta t}\right)\,\vec{v}(t-\Delta t) + \mathbf{M}\,\vec{f}(t) \\
\left(\mathbf{W}_2 + \frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t+\Delta t) &= \left(2\,\mathbf{W}_2 - (\Delta t)^2\mathbf{A}\right)\,\vec{v}(t) - \left(\mathbf{W}_2 - \frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t-\Delta t) + (\Delta t)^2\mathbf{M}\,\vec{f}(t)
\end{aligned}
$$

The initial conditions

$$\vec{v}(t_0) = \vec{u}_0 - \vec{u}_B \quad \text{and} \quad \frac{d}{dt}\,v(t_0) = \vec{v}_0$$

are implemented by

$$\vec{v}_0 \approx \frac{\vec{v}(t_0+\Delta t) - \vec{v}(t_0-\Delta t)}{2\,\Delta t} \quad \text{or} \quad \vec{v}(t_0-\Delta t) \approx \vec{v}(t_0+\Delta t) - 2\,\Delta t\,\vec{v}_0$$

and thus the first time step is given by

$$
\begin{aligned}
\left(\mathbf{W}_2 + \frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t_0+\Delta t) &= \left(2\,\mathbf{W}_2 - (\Delta t)^2\mathbf{A}\right)\,\vec{v}(t_0) - \left(\mathbf{W}_2 - \frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t_0-\Delta t) + (\Delta t)^2\mathbf{M}\,\vec{f}(t_0) \\
2\,\mathbf{W}_2\,\vec{v}(t_0+\Delta t) &= \left(2\,\mathbf{W}_2 - (\Delta t)^2\mathbf{A}\right)\,\vec{v}(t_0) + 2\,\left(\Delta t\,\mathbf{W}_2 - (\Delta t)^2\mathbf{W}_1\right)\,\vec{v}_0 + (\Delta t)^2\mathbf{M}\,\vec{f}(t_0)\;.
\end{aligned}
$$

This scheme is conditionally stable with the stability condition

$$\lambda \le \frac{4}{(\Delta t)^2} \quad \text{or} \quad \Delta t \le \frac{2}{\sqrt{\lambda}} \quad \text{for all generalized eigenvalues of} \quad \mathbf{A}\,\vec{u} = \lambda\,\mathbf{W}_2\,\vec{u}\;.$$

7.7.3 Solvers for problems with a nonlinear expressions $f(x,t,u)$

In expressions for the implicit and explicit solvers replace $\vec{f}(t)$ by $\vec{f}(t,\vec{u}(t))$, where $\vec{u}(t)=\vec{u}_B+\vec{v}(t)$.

- The implicit solver in Section 7.7.1 leads to

$$\left(+\mathbf{W}_2+\Delta t\,\mathbf{W}_1+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) \;=\; -\left(\mathbf{W}_2-\Delta t\,\mathbf{W}_1+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t)+ \\ +\left(2\,\mathbf{W}_2-\frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t)+(\Delta t)^2\,\mathbf{M}\,\vec{f}(t,\vec{u}_B+\vec{v}(t))$$

  The first step from $t=0$ to $t=\Delta t$ is determined by

$$\left(+\mathbf{W}_2+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(+\Delta t) \;=\; \Delta t\,\left(\mathbf{W}_2-\Delta t\,\mathbf{W}_1+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}_0+ \\ +\left(\mathbf{W}_2-\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(0)+\frac{(\Delta t)^2}{2}\,\mathbf{M}\,\vec{f}(0,\vec{u}_0)\;.$$

- The explicit solver in Section 7.7.2 leads to

$$\left(\mathbf{W}_2+\frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t+\Delta t)=\left(2\,\mathbf{W}_2-(\Delta t)^2\mathbf{A}\right)\,\vec{v}(t)-\left(\mathbf{W}_2-\frac{\Delta t}{2}\,\mathbf{W}_1\right)\,\vec{v}(t-\Delta t)+(\Delta t)^2\mathbf{M}\,\vec{f}(t,\vec{u}_B+\vec{v}(t))$$

  The first step from $t=t_0$ to $t=t_0+\Delta t$ is determined by

$$2\,\mathbf{W}_2\,\vec{v}(t_0+\Delta t) \;=\; \left(2\,\mathbf{W}_2-(\Delta t)^2\mathbf{A}\right)\,\vec{v}(t_0)+2\,\left(\Delta t\,\mathbf{W}_2-(\Delta t)^2\mathbf{W}_1\right)\,\vec{v}_0+(\Delta t)^2\mathbf{M}\,\vec{f}(t_0,\vec{u}_0)\;.$$

Based on the above find the code in `I2BVP1DNL.m` as minor modification of `I2BVP1D.m` .

7.8 Nonlinear boundary value problems, Newton's method and partial substitution

For smooth functions $a(x,u,u')$ and $f(x,u,u')$ examine a nonlinear boundary value problem of the form (91)

$$-\left(a(x,u(x),u'(x))\,u'(x)\right)' + b(x)\,u'(x) + c(x)\,u(x) = d(x)\,f(x,u(x),u'(x))\;, \tag{101}$$

with linear, constant boundary conditions, Dirichlet or Neumann. The essential tool is Newton's method, combined with a partial substitution. For this use a linear Taylor approximation of the nonlinear function $f(x,u,u')$

$$f(x,u+\phi,u'+\phi') \approx f(x,u,u') + f_u(x,u,u')\,\phi + f_{u'}(x,u,u')\,\phi'$$

with the notations $f_u = \frac{\partial f}{\partial u}$ and $f_{u'} = \frac{\partial f}{\partial u'}$. For an approximate solution $u_n(x)$ search a solution of the form $u_n(x)+\phi(x)$. Examine the linear boundary value problem for the perturbation ϕ.

$$\begin{aligned} -\left(a\,(u_n'+\phi')\right)' + b\,(u_n'+\phi') + c\,(u_n+\phi) &= d\cdot(f(\cdot,u_n,u_n')+ \\ &\quad +f_u(\cdot,u_n,u_n')\,\phi + f_{u'}(\cdot,u_n,u_n')\,\phi') \\ -\left(a\,\phi'\right)' + (b - d\cdot f_{u'}(\cdot,u_n,u_n'))\,\phi' + (c - d\cdot f_u(\cdot,u_n,u_n'))\,\phi &= +\left(a\,u_n'\right)' - b\,u_n' - c\,u_n + d\cdot f(\cdot,u_n,u_n') \end{aligned}$$

and then update the solution to $u_{n+1} = u_n + \phi$. For the perturbation ϕ use zero boundary conditions, assuming that u_0 satisfied the boundary conditions. For an initial guess $u_0(x)$ close enough to an isolated solution the Newton based algorithm will converge.

To implement the above algorithm in `FEMoctave` start out with an initial function $u(x) = u_0(x)$, hopefully close to the true solution.

- Start with the function $u_0(x)$ and evaluate (if necessary) $f_0 = f(x,u_0(x))$, or $f_0 = f(x,u_0(x),u_0'(x))$ at the nodes. Evaluate at the Gauss points (if necessary)

  $$a_0 = a(x,u_0(x),u_0'(x)) \quad , \quad b_0 = b(x) \quad \text{and} \quad c_0 = c(x)\;.$$

- Solve the boundary value problem for u_n

  $$-\left(a_0\,u_n'(x)\right)' + b_0\,u_n'(x) + c_0\,u_n(x) = d\cdot f_0$$

  with the correct boundary conditions.

- Repeat
  - Store the current solution $u_{old} = u_n$.
  - If a depends on u or u' evaluate

    $$a_n = x(x,u_n(x),u_n'(x)) \quad \text{at the Gauss points.}$$

    If a does not depend on u and u', reuse $a_n = a_0$.
  - If f depends on u or u' evaluate

    $$\begin{aligned} f_n &= f(x,u_n(x),u_n'(x)) \quad \text{at the nodes} \\ f_u &= \frac{\partial}{\partial u} f(x,u_n(x),u_n'(x)) \quad \text{at the Gauss points} \\ f_{u'} &= \frac{\partial}{\partial u'} f(x,u_n(x),u_n'(x)) \quad \text{at the Gauss points} \end{aligned}$$

  - Evaluate

    $$\text{RHS}_n = -(a_n\,u_n')' + b_0\,u_n' + c_0\,u_n - d\cdot f_n$$

    This can be done with a matrix multiplication.
  - Solve the boundary value problem for the perturbation ϕ

    $$-\left(a_n\,\phi'(x)\right)' + (b_0 - d\cdot f_{u'})\,\phi'(x) + (c_0 - d\cdot f_u)\,\phi(x) = -\text{RHS}_n$$

    with homogeneous boundary conditions.
  - Update $u_n \;\longrightarrow\; u_n + \phi$.

- If f depends on u or u' evaluate at the nodes.

$$f_n = f(x, u_n(x), u'_n(x))$$

- If a depends on u or u' evaluate

$$a_n = x(x, u_n(x), u'_n(x)) \quad \text{at the Gauss points.}$$

  If a does not depend on u, reuse $a_n = a_0$.
- Solve the boundary value problem for u_n

$$-\left(a_0\, u'_n(x)\right)' + b_0\, u'(x) + c_0\, u_n(x) = d \cdot f_n$$

  with the correct boundary conditions.

- until $\|u_n - u_{old}\|$ small enough or too many iterations.

For the convergence test absolute and relative values are used, if one of them is small enough the algorithm stops.

This is **not a pure Newton's approach**, but a combination with a (partial) substitution method.

8 The Algorithms for Plane Elasticity and Axially Symmetric Elasticity

Find the description of the plane elasticity problems in Section 2.19, starting on page 17.

8.1 The plane stress problem

For a plane stress problem it is assumed that there are no stresses in z–direction, i.e. $\sigma_z = \tau_{xz} = \tau_{yz} = 0$. The elastic energy density is given by equation (28), i.e.

$$W_{stress} = \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2\right)\ .$$

With FEMoctave examine plane stress deformations with (only) three types of boundary conditions.

$$\begin{array}{rcll} \vec{u} & = & \vec{g}_D & \text{on Dirichlet boundary } \Gamma_1\text{, i.e. prescribed displacement} \\ \text{force density} & = & \vec{g}_N & \text{on Neumann boundary } \Gamma_2\text{, i.e. prescribed force density} \\ \text{force density} & = & \vec{0} & \text{on free boundary } \Gamma_3 \end{array} \tag{102}$$

With this the total energy of a plane stress problem can be written in the form[38]

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}\ ,\ \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle\ dA - \\ &\quad - \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\ dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\ ds\ . \end{aligned} \tag{103}$$

Using the Bernoulli principle this energy has to be minimized, leading to the Euler–Lagrange equations (30). A discretization of the displacements $u_1(x,y)$ and $u_2(x,y)$ leads to a vector $\vec{u} = (\vec{u}_1, \vec{u}_2)$ and the above total energy has to be written in the form

$$\frac{1}{2}\,\langle \vec{u}, \mathbf{A}\vec{u}\rangle + \langle \vec{u}, \mathbf{W}\vec{f}\rangle\ .$$

Then the approximate minimizer is given as solution of the linear system $\mathbf{A}\vec{u} = -\mathbf{W}\vec{f}$. This setup is very similar to the connections shown in Figure 90 on page 162.

Another approach is to use perturbed displacements $u_1 + \phi_1$ and $u_2 + \phi_2$ and dropping higher order contributions in ϕ_i. Use the approximations

$$\begin{aligned} \varepsilon_{xx} &= \frac{\partial\,(u_1+\phi_1)}{\partial x} = \frac{\partial\,u_1}{\partial x} + \frac{\partial\,\phi_1}{\partial x} \quad , \quad \varepsilon_{yy} = \frac{\partial\,(u_2+\phi_2)}{\partial y} = \frac{\partial\,u_2}{\partial y} + \frac{\partial\,\phi_2}{\partial y} \\ \varepsilon_{xx}^2 &= (\frac{\partial\,(u_1+\phi_1)}{\partial x})^2 \approx (\frac{\partial\,u_1}{\partial x})^2 + 2\,(\frac{\partial\,u_1}{\partial x})\,(\frac{\partial\,\phi_1}{\partial x}) \\ \varepsilon_{yy}^2 &\approx (\frac{\partial\,u_2}{\partial y})^2 + 2\,(\frac{\partial\,u_2}{\partial y})\,(\frac{\partial\,\phi_2}{\partial y}) \\ \varepsilon_{xx}\varepsilon_{yy} &\approx \frac{\partial\,u_1}{\partial x}\,\frac{\partial\,u_2}{\partial y} + \frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_2}{\partial y} + \frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_1}{\partial x} \\ 2\,\varepsilon_{xy} &= \frac{\partial\,(u_1+\phi_1)}{\partial y} + \frac{\partial\,(u_2+\phi_2)}{\partial x} = \frac{\partial\,u_1}{\partial y} + \frac{\partial\,\phi_1}{dy} + \frac{\partial\,u_2}{\partial x} + \frac{\partial\,\phi_2}{\partial x} \\ 4\,\varepsilon_{xy}^2 &\approx (\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})^2 + 2\,\frac{\partial\,u_1}{\partial y}\,(\frac{\partial\,\phi_1}{\partial y} + \frac{\partial\,\phi_2}{\partial x}) + 2\,\frac{\partial\,u_2}{\partial x}\,(\frac{\partial\,\phi_1}{\partial y} + \frac{\partial\,\phi_2}{\partial x}) \\ &= (\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})^2 + 2\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x}) + 2\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x}) \end{aligned}$$

[38] We quietly dropped the constant thickness H from all expressions.

Based on $\frac{2\,(1-\nu^2)}{E}\,W(u) = \varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2$ conclude

$$\begin{aligned}\frac{2\,(1-\nu^2)}{E}\,\Big(W(\vec{u}+\vec{\phi}) - W(\vec{u})\Big) \;\approx\; & 2\,\frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_1}{\partial x} + 2\,\frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_2}{\partial y} + 2\,\nu\,(\frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_2}{\partial y} + \frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_1}{\partial x}) + \\ & + \frac{4}{4}\,(1-\nu)\,\left(\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x}) + \frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right)\end{aligned}$$

and use

$$\begin{aligned}\iint\limits_{\Omega} \vec{f}\cdot(\vec{u}+\vec{\phi})\;dA \;&=\; \iint\limits_{\Omega} \vec{f}\cdot\vec{u}\;dA + \iint\limits_{\Omega} f_1\,\phi_1 + f_2\,\phi_2\;dA \\ \int_{\Gamma_2} \vec{g}_N\cdot(\vec{u}+\vec{\phi})\;ds \;&=\; \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\;ds + \int_{\Gamma_2} g_1\,\phi_1 + g_2\,\phi_2\;ds\;.\end{aligned}$$

This leads to

$$\begin{aligned}U(\vec{u}+\vec{\phi}) - U(\vec{u}) \;\approx\; & + \iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x} + \nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) - \phi_1\,f_1\;dA + \\ & + \iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y} + \nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) - \phi_2\,f_2\;dA - \\ & - \int_{\Gamma_2} \phi_1\,g_1 + \phi_2\,g_2\;ds \;=\; 0\;. \qquad (104)\end{aligned}$$

Using Bernoulli's principle this expression should vanish for all perturbations $\vec{\phi}$. Use discrete approximations of the functions u_i and ϕ_i to write the vanishing condition for expression (104) in the form

$$\langle\vec{\phi}, \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\rangle = 0 \quad \text{for all } \vec{\phi}\;.$$

If thermal expansion is taken into account there is an additional contribution to the energy $U(\vec{u})$ of

$$-\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy}) = -\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\frac{\partial\,u_1}{\partial x} + \frac{\partial\,u_2}{\partial y})\;.$$

Examine

$$\begin{aligned}I_{therm}(\vec{u}) \;&=\; \iint\limits_{\Omega} -\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\frac{\partial\,u_1}{\partial x} + \frac{\partial\,u_2}{\partial y})\;dA \\ I_{therm}(\vec{u}+\vec{\phi}) - I_{therm}(\vec{u}) \;&\approx\; \iint\limits_{\Omega} -\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\frac{\partial\,\phi_1}{\partial x} + \frac{\partial\,\phi_2}{\partial y})\;dA\;. \qquad (105)\end{aligned}$$

8.2 The plane stress eigenvalue and dynamic problem

For the eigenvalue problem (32)

$$\begin{aligned}-\,\mathrm{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{\partial\,u_1}{\partial x} + \nu\,\frac{\partial\,u_2}{\partial y} \\ \frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\end{pmatrix}\right) \;&=\; \lambda\,\rho\,u_1 \\ -\,\mathrm{div}\left(\frac{E}{1-\nu^2}\,\begin{pmatrix}\frac{1-\nu}{2}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x}) \\ \frac{\partial\,u_2}{\partial y} + \nu\,\frac{\partial\,u_1}{\partial x}\end{pmatrix}\right) \;&=\; \lambda\,\rho\,u_2\end{aligned}$$

use the equation

$$\begin{aligned}0 \;=\; & + \iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x} + \nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) - \phi_1\,\lambda\,\rho\,u_1\;dA + \\ & + \iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y} + \nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) - \phi_2\,\lambda\,\rho\,u_2\;dA\end{aligned}$$

for all smooth test functions ϕ_1 and ϕ_2. Write this in the form $\langle \vec{\phi}\,,\ \mathbf{A}\vec{u} - \lambda\,\mathbf{W}\vec{u}\rangle = 0$ to lead to a generalized eigenvalue problem $\mathbf{A}\vec{u} = \lambda\,\mathbf{W}\vec{u}$. `FEMoctave` allows to determine a few of the smallest eigenvalues λ and the corresponding eigen modes.

For the dynamic problem (31)

$$\begin{aligned} -\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{\partial u_1}{\partial x} + \nu\,\frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2}\,(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \end{pmatrix}\right) + f_1 &= \rho\,\tfrac{\partial^2}{\partial t^2}\,u_1 \\ -\operatorname{div}\left(\frac{E}{1-\nu^2}\begin{pmatrix} \frac{1-\nu}{2}\,(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \\ \frac{\partial u_2}{\partial y} + \nu\,\frac{\partial u_1}{\partial x} \end{pmatrix}\right) + f_2 &= \rho\,\tfrac{\partial^2}{\partial t^2}\,u_2 \end{aligned}$$

use the equation

$$\begin{aligned} 0 \;=\; &+\iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x} + \nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) + \phi_1\,f_1 - \phi_1\,\rho\,\frac{\partial^2}{\partial t^2}\,u_1\;dA + \\ &+\iint\limits_{\Omega} \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y} + \nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x})\right) + \phi_2\,f_2 - \phi_2\,\rho\,\frac{\partial^2}{\partial t^2}\,u_2\;dA \end{aligned}$$

for all smooth test functions ϕ_1 and ϕ_2. This will lead to a dynamic problem of the form

$$\langle \vec{\phi}\,,\ \mathbf{A}\,\vec{u} - \mathbf{W_f}\,\vec{f} + \mathbf{W}_\rho\,\frac{d^2}{dt^2}\,\vec{u}\rangle = 0 \quad \text{for all} \quad \vec{\phi}\,.$$

The matrices $\mathbf{A}$, $\mathbf{W}_\rho$ and $\mathbf{W}_f$ have to take the active degrees of freedom into account, leading to a vector $\vec{v}$, the active contribution of $\vec{u}$. These matrices are generated by the codes `PStressEquation*WM.m` for elements of order 1, 2 and 3. As consequence the dynamic problem to be solved is a system of ODEs

$$\mathbf{W}_\rho\,\frac{d^2}{dt^2}\,\vec{u}(t) + \mathbf{A}\,\vec{u}(t) = \mathbf{W_f}\,\vec{f}(t) \tag{106}$$

with appropriate initial conditions $\vec{u}(t_0) = \vec{u}_0 - \vec{u}_B$ and $\frac{d}{dt}\,\vec{u}(t_0) = \vec{v}_0$. Algorithms to solve this problem are spelled out in Section 7.7, used for the wave equation (100) with one space variable. An implicit and an explicit solver are used for dynamic plane stress and strain problems.

For the implicit approximation adapt the notation from Section 7.7, leading to the result

$$\begin{aligned} \mathbf{W}_\rho\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -\mathbf{A}\,\vec{v}(t) + \mathbf{W}_f\,\vec{f}(t) \\ \mathbf{W}_\rho\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -\mathbf{A}\,\frac{\vec{v}(t-\Delta t) + 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{4} + \mathbf{W}_f\,\vec{f}(t) \\ \left(+\mathbf{W}_\rho + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= -\left(\mathbf{W}_\rho + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t) + \\ &\quad + \left(2\,\mathbf{W}_\rho - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t) + (\Delta t)^2\,\mathbf{W}_f\,\vec{f}(t)\,. \end{aligned}$$

For the first step use $\vec{v}(t_0 - \Delta t) \approx \vec{v}(t_0 + \Delta t) - 2\,\Delta t\,\vec{v}_0$, leading to

$$\left(+\mathbf{W}_\rho + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t_0+\Delta t) = \Delta t\,\left(\mathbf{W}_\rho + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}_0 + \left(\mathbf{W}_\rho - \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,(\vec{u}_0 - \vec{u}_B) + \frac{(\Delta t)^2}{2}\,\mathbf{W}_f\,\vec{f}(0)$$

where $\vec{u}_B$ is the steady state contribution with inhomogeneous boundary contributions.

For the explicit solver of the ODE (106) use

$$\begin{aligned} \mathbf{W}_\rho\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -\mathbf{A}\,\vec{v}(t) + \mathbf{W}_f\,\vec{f}(t) \\ \mathbf{W}_\rho\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -\mathbf{A}\,\vec{v}(t) + \mathbf{W}_f\,\vec{f}(t) \\ \mathbf{W}_\rho\vec{v}(t+\Delta t) &= -\mathbf{W}_\rho\,\vec{v}(t-\Delta t) + \left(2\,\mathbf{W}_\rho - (\Delta t)^2\,\mathbf{A}\right)\,\vec{v}(t) + (\Delta t)^2\,\mathbf{W}_f\,\vec{f}(t) \end{aligned}$$

and for the first step use again the approximation $\vec{y}(-\Delta t) \approx \vec{v}(+\Delta t) - 2\,\Delta t\,\vec{v}_0$, leading to

$$\begin{aligned}
\mathbf{W}_\rho\,\vec{v}(+\Delta t) &= -\mathbf{W}_\rho\,(\vec{v}(+\Delta t) - 2\,\Delta t\,\vec{v}_0) + \left(2\,\mathbf{W}_\rho - (\Delta t)^2\,\mathbf{A}\right)\,\vec{v}(0) + (\Delta t)^2\,\mathbf{W}_f\,\vec{f}(0) \\
2\,\mathbf{W}_\rho\,\vec{v}(+\Delta t) &= +2\,\Delta t\,\mathbf{W}_\rho\,\vec{v}_0 + \left(2\,\mathbf{W}_\rho - (\Delta t)^2\,\mathbf{A}\right)\,\vec{v}(0) + (\Delta t)^2\,\mathbf{W}_f\,\vec{f}(0) \\
\mathbf{W}_\rho\,\vec{v}(+\Delta t) &= +\Delta t\,\mathbf{W}_\rho\,\vec{v}_0 + \left(\mathbf{W}_\rho - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,(\vec{u}_0 - \vec{u}_B) + \frac{(\Delta t)^2}{2}\,\mathbf{W}_f\,\vec{f}(0)\,.
\end{aligned}$$

This explicit scheme is conditionally stable with the stability condition

$$\lambda \le \frac{4}{(\Delta t)^2} \quad \text{or} \quad \Delta t \le \frac{2}{\sqrt{\lambda}} \quad \text{for all generalized eigenvalues of} \quad \mathbf{A}\,\vec{u} = \lambda\,\mathbf{W}_\rho\,\vec{u}\,.$$

8.3 Construction of first order elements

The algorithm in this section is based on the results in Section 6.4 (p. 167), with the expressions in equation (104) to be integrated over a triangle T. The approximation consists of piecewise linear, triangular segments. Thus the first order partial derivatives are constant on each triangle. Consequently the strains are constant on each triangle. This is the reason for the name Constant Strain Triangle, short CST.

8.3.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$

- If the values of the functions f_1 and f_2 at the Gauss points are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2$, then use the approximation

$$\begin{aligned}
\iint_T f_1\,\phi_1 + f_2\,\phi_2\,dA &\approx \frac{\text{area}(T)}{3}\,\left(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\vec{f}_2\rangle\right) \\
&= \frac{\text{area}(T)}{3}\,\left(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\vec{f}_2\rangle\right)\,.
\end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{3\times 3}$ is the symmetric matrix for interpolation from the nodes to the Gauss points, given by

$$\mathbf{M} = \frac{1}{6}\,\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix} = \mathbf{M}^T\,.$$

- If the values of the functions f_1 and f_2 at the nodes are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2$, then use the approximation

$$\begin{aligned}
\iint_T f_1\,\phi_1 + f_2\,\phi_2\,dA &\approx \frac{\text{area}(T)}{3}\,\left(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\mathbf{M}\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\mathbf{M}\,\vec{f}_2\rangle\right) \\
&= \frac{\text{area}(T)}{3}\,\left(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\mathbf{M}\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\mathbf{M}\,\vec{f}_2\rangle\right)\,.
\end{aligned}$$

Thus find one contribution to (104). With a block matrix notation the above can be written in the form

$$\frac{\text{area}(T)}{3}\,\langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle \quad \text{or} \quad \frac{\text{area}(T)}{3}\,\langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\mathbf{M} \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle\,.$$

8.3.2 Integration of the terms involving derivatives of u_1, u_2, ϕ_1 and ϕ_2

For linear elements the gradient of the functions u_i and ϕ_i are constant on a triangular element and using equation (52) given by

$$\nabla u = \frac{-1}{2\,\text{area}(T)}\,\begin{bmatrix} (y_3 - y_2) & (y_1 - y_3) & (y_2 - y_1) \\ (x_2 - x_3) & (x_3 - x_1) & (x_1 - x_2) \end{bmatrix} \cdot \vec{u} = \begin{bmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{bmatrix}\,\vec{u}\,.$$

Evaluate the coefficients E and ν at the Gauss points $\vec{g}_i$ and define the averaged values

$$a_1 = \frac{1}{3}\,\sum_{i=1}^{3}\,\frac{E(\vec{g}_i)}{1-\nu^2(\vec{g}_i)} \quad , \quad a_2 = \frac{1}{3}\,\sum_{i=1}^{3}\,\frac{\nu(\vec{g}_i)\,E(\vec{g}_i)}{1-\nu^2(\vec{g}_i)} \quad \text{and} \quad a_3 = \frac{1}{3}\,\sum_{i=1}^{3}\,\frac{E(\vec{g}_i)}{2\,(1+\nu(\vec{g}_i))}\,.$$

This leads to the approximations

$$\begin{aligned}
\text{area}(T)\, I_{\phi_1} &= \iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\
&\approx \langle \mathbf{G}_x\vec{\phi}_1, \mathbf{G}_x\vec{u}_1\rangle + a_2\,\langle \mathbf{G}_x\vec{\phi}_1, \mathbf{G}_y\vec{u}_2\rangle + a_3\,\langle \mathbf{G}_y\vec{\phi}_1, \mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\
&= a_1\,\langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{G}_x\vec{u}_1\rangle + a_2\,\langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{G}_y\vec{u}_2\rangle + a_3\,\langle \vec{\phi}_1, \mathbf{G}_y^T\mathbf{G}_y\vec{u}_1+\mathbf{G}_y^T\mathbf{G}_x\vec{u}_2\rangle \\
\text{area}(T)\, I_{\phi_2} &= \iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\
&\approx a_1\,\langle \mathbf{G}_y\vec{\phi}_2, \mathbf{G}_y\vec{u}_2\rangle + a_2\,\langle \mathbf{G}_y\vec{\phi}_2, \mathbf{G}_x\vec{u}_1\rangle + a_3\,\langle \mathbf{G}_x\vec{\phi}_2, \mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\
&= a_1\,\langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{G}_y\vec{u}_2\rangle + a_2\,\langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{G}_x\vec{u}_1\rangle + a_3\,\langle \vec{\phi}_2, \mathbf{G}_x^T\mathbf{G}_y\vec{u}_1+\mathbf{G}_x^T\mathbf{G}_x\vec{u}_2\rangle
\end{aligned}$$

With a block matrix notation write the above in the form

$$I_{\vec{\phi}} \approx \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} a_1\mathbf{G}_x^T\mathbf{G}_x + a_3\mathbf{G}_y^T\mathbf{G}_y & a_2\mathbf{G}_x^T\mathbf{G}_y + a_3\mathbf{G}_y^T\mathbf{G}_x \\ a_2\mathbf{G}_y^T\mathbf{G}_x + a_3\mathbf{G}_x^T\mathbf{G}_y & a_1\mathbf{G}_y^T\mathbf{G}_y + a_3\mathbf{G}_x^T\mathbf{G}_x \end{bmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \end{pmatrix} \rangle =: \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \mathbf{G} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \end{pmatrix} \rangle\,.$$

The symmetric 6×6 matrix $\mathbf{G}\in\mathbb{R}^{6\times 6}$ is the element stiffness matrix for the triangle T. The expression $\text{area}(T)\, I_{\vec{\phi}}$ is one of the contributions to the integral in (104) by the triangular element T.

8.3.3 The boundary integral

The boundary integral is similar to (53) on page 172. With $\alpha = \frac{1-1/\sqrt{3}}{2}$ use the symmetric interpolation matrix from nodes to Gauss points

$$\mathbf{M}_b = \begin{bmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{bmatrix}$$

and the length L of the edge segment for the approximate integral

$$\int_{\text{edge}} \vec{g}_N\cdot\vec{\phi}\,ds = \int_{\text{edge}} g_1\,\phi_1 + g_2\,\phi_2\,ds \approx \frac{L}{2}\,\langle\vec{\phi}_1, \mathbf{M}_b\,\vec{g}_1\rangle + \frac{L}{2}\,\langle\vec{\phi}_2, \mathbf{M}_b\,\vec{g}_2\rangle\,,$$

where the functions g_1 and g_2 are evaluated at the Gauss points on the edge.

8.3.4 Construct a weight matrix W

For eigenvalue and dynamic problems it is necessary to evaluate

$$\iint\limits_\Omega \rho(x,y)\,(f_1(x,y)\,\phi_1(x,y)+f_2(x,y)\,\phi_2(x,y)\,dA \approx \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{W}_1 & 0 \\ 0 & \mathbf{W}_2 \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle\,,$$

where $\vec{\phi}_i$ and $\vec{f}_i$ are evaluated at the nodes. For a weight function $\rho(x,y)$ construct the matrix $\mathbf{W}_T$ for a given triangle with the values of $\rho(x,y)$ at the Gauss points by

$$\mathbf{W}_T = \begin{bmatrix} \rho(\vec{g}_1) & 0 & 0 \\ 0 & \rho(\vec{g}_2) & 0 \\ 0 & 0 & \rho(\vec{g}_3) \end{bmatrix}$$

and then use the interpolation matrix $\mathbf{M}$ from nodes to Gauss points.

$$\begin{aligned}
\iint\limits_T \rho\,(f_1\,\phi_1+f_2\,\phi_2)\,dA &\approx \frac{\text{area}(T)}{3}\left(\langle \mathbf{W}_T\mathbf{M}\,\vec{\phi}_1\,,\,\mathbf{M}\vec{f}_1\rangle + \langle \mathbf{W}_T\mathbf{M}\,\vec{\phi}_2\,,\,\mathbf{M}\vec{f}_2\rangle\right) \\
&= \frac{\text{area}(T)}{3}\left(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\mathbf{W}_T\mathbf{M}\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\mathbf{W}_T\mathbf{M}\vec{f}_2\rangle\right) \\
&= \frac{\text{area}(T)}{3}\,\langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\mathbf{W}_T\mathbf{M} & 0 \\ 0 & \mathbf{M}^T\mathbf{W}_T\mathbf{M} \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle
\end{aligned}$$

For the constant function $\rho(x,y) = 1$ obtain

$$\mathbf{M}^T \mathbf{W}_T \mathbf{M} = \mathbf{M}^2 = \frac{1}{4} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} .$$

8.3.5 The contribution by a thermal expansion

The additional contribution by a thermal expansion is shown in (105).

$$I_{therm}(\vec{u} + \vec{\phi}) - I_{therm}(\vec{u}) \approx \Delta I_{therm} = \iint_T -\frac{E\,\alpha\,\Delta T}{1-\nu} \left(\frac{\partial\,\phi_1}{\partial x} + \frac{\partial\,\phi_2}{\partial y}\right) dA$$

- The function $\text{th}(\vec{x}) = \frac{\alpha(\vec{x})\,\Delta T(\vec{x})}{1-\nu(x)}$ is given at the Gauss points, leading to a weight vector

$$\vec{w}_{therm} = (\text{th}(\vec{g}_1),\, \text{th}(\vec{g}_2),\, \text{th}(\vec{g}_3))^T .$$

- For first order elements the partial derivatives $\frac{\partial\,\phi_1}{\partial x}$ and $\frac{\partial\,\phi_2}{\partial y}$ are constant on each triangle T and given by

$$\begin{aligned} \frac{\partial\,\phi_1}{\partial x} &= \mathbf{G_x}\,\vec{\phi}_1 = \mathbf{G_x} \begin{pmatrix} \phi_{1,1} \\ \phi_{1,2} \\ \phi_{1,3} \end{pmatrix} = \begin{bmatrix} y_3 - y_2 & y_1 - y_3 & y_2 - y_1 \end{bmatrix} \begin{pmatrix} \phi_{1,1} \\ \phi_{1,2} \\ \phi_{1,3} \end{pmatrix} \\ \frac{\partial\,\phi_2}{\partial y} &= \mathbf{G_y}\,\vec{\phi}_2 = \mathbf{G_y} \begin{pmatrix} \phi_{2,1} \\ \phi_{2,2} \\ \phi_{2,3} \end{pmatrix} = \begin{bmatrix} x_2 - x_3 & x_3 - x_1 & x_1 - x_2 \end{bmatrix} \begin{pmatrix} \phi_{2,1} \\ \phi_{2,2} \\ \phi_{2,3} \end{pmatrix} \end{aligned}$$

This leads to

$$\begin{aligned} \Delta I_{therm} &\approx -\frac{\text{area}(T)}{3} \left(\text{th}(\vec{g}_1) + \text{th}(\vec{g}_2) + \text{th}(\vec{g}_3)\right) \left(\mathbf{G_x}\,\vec{\phi}_1 + \mathbf{G_y}\,\vec{\phi}_2\right) \\ &= -\frac{\text{area}(T)}{3} \left(\text{th}(\vec{g}_1) + \text{th}(\vec{g}_2) + \text{th}(\vec{g}_3)\right) \left(\langle \mathbf{G_x}^T, \vec{\phi}_1 \rangle + \langle \mathbf{G_y}^T, \vec{\phi}_2 \rangle\right) \end{aligned}$$

8.4 Construction of second order elements

The algorithm in this section is based on the results in Section 6.5 (p. 173), with the expressions in equation (104) to be integrated over a triangle T. The approximation consists of piecewise quadratic, triangular segments. Thus the first order partial derivatives are linear on each triangle.

8.4.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$

Use the Gauss weights $\vec{w} \in \mathbb{R}^7$ from equation (49) on page 167 for the approximate integration over one triangle T.

- If the values of the functions f_1 and f_2 at the seven Gauss points are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2 \in \mathbb{R}^7$, then use the approximation

$$\begin{aligned} \iint_T f_1\,\phi_1 + f_2\,\phi_2\, dA &\approx \text{area}(T) \left(\langle \mathbf{M}\,\vec{\phi}_1\,,\, \text{diag}(\vec{w})\,\vec{f}_1 \rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\, \text{diag}(\vec{w})\,\vec{f}_2 \rangle\right) \\ &= \text{area}(T) \left(\langle \vec{\phi}_1\,,\, \mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_1 \rangle + \langle \vec{\phi}_2\,,\, \mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_2 \rangle\right) \\ &= \text{area}(T)\, \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\,\text{diag}(\vec{w}) & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\,\text{diag}(\vec{w}) \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle . \end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 6}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (56) on page 174.

- If the values of the functions f_1 and f_2 at the nodes are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2 \in \mathbb{R}^6$, then use the approximation

$$\begin{aligned}\iint\limits_T f_1\,\phi_1 + f_2\,\phi_2\; dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big)\\ &= \text{area}(T)\,\Big(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big)\\ &= \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}, \begin{bmatrix}\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\end{bmatrix}\begin{pmatrix}\vec{f}_1\\ \vec{f}_2\end{pmatrix}\rangle\,.\end{aligned}$$

Thus find one contribution to (104). Observe that $\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}$ is a 6×6 matrix, independent on the triangle T.

8.4.2 Integration of the terms involving derivatives of $\vec{u}$, ϕ_1 and ϕ_2

Using the results from Section 6.5 the partial derivatives at the nodes of functions ϕ given at the notes find for the first component $\varphi_x = \frac{\partial\,\varphi}{\partial x}$ of the gradient at the Gauss points

$$\begin{pmatrix}\varphi_x(\vec{x}_1)\\ \varphi_x(\vec{x}_2)\\ \vdots\\ \varphi_x(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\,\Big[\;(+y_3-y_1)\,\mathbf{M}_\xi^T + (-y_2+y_1)\,\mathbf{M}_\nu^T\;\Big]\cdot\vec{\phi} =: \mathbf{G}_x\,\vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix}\varphi_y(\vec{x}_1)\\ \varphi_y(\vec{x}_2)\\ \vdots\\ \varphi_y(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\,\Big[\;(-x_3+x_1)\,\mathbf{M}_\xi^T + (+x_2-x_1)\,\mathbf{M}_\nu^T\;\Big]\cdot\vec{\phi} =: \mathbf{G}_y\,\vec{\phi}\,.$$

Evaluate the coefficients E and ν at the Gauss points g_i and multiply by the Gauss integration weights w_i to obtain the three diagonal matrices

$$\mathbf{A}_1 = \text{diag}\begin{pmatrix}w_1\,\frac{E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)}\\ w_2\,\frac{E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)}\\ \vdots\\ w_7\,\frac{E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)}\end{pmatrix},\quad \mathbf{A}_2 = \text{diag}\begin{pmatrix}w_1\,\frac{\nu(\vec{g}_1)\,E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)}\\ w_2\,\frac{\nu(\vec{g}_2)\,E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)}\\ \vdots\\ w_7\,\frac{\nu(\vec{g}_7)\,E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)}\end{pmatrix}\quad\text{and}\quad \mathbf{A}_3 = \text{diag}\begin{pmatrix}w_1\,\frac{E(\vec{g}_1)}{2\,(1+\nu(\vec{g}_1))}\\ w_2\,\frac{E(\vec{g}_2)}{2\,(1+\nu(\vec{g}_2))}\\ \vdots\\ w_7\,\frac{E(\vec{g}_7)}{2\,(1+\nu(\vec{g}_7))}\end{pmatrix}.$$

This leads to the approximations

$$\begin{aligned}\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA\\ &\approx \langle \mathbf{A}_1\,\mathbf{G}_x\vec{\phi}_1, \mathbf{G}_x\vec{u}_1\rangle + \langle \text{diag}\,\mathbf{A}_2\,\mathbf{G}_x\vec{\phi}_1, \mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_3\,\mathbf{G}_y\vec{\phi}_1, \mathbf{G}_y\vec{u}_1 + \mathbf{G}_x\vec{u}_2\rangle\\ &= \langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{A}_1\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{A}_2\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_1, \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1 + \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle\\ \frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA\\ &\approx \langle \mathbf{A}_1\mathbf{G}_y\vec{\phi}_2, \mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_2\mathbf{G}_y\vec{\phi}_2, \mathbf{G}_x\vec{u}_1\rangle + \langle \mathbf{A}_3\mathbf{G}_x\vec{\phi}_2, \mathbf{G}_y\vec{u}_1 + \mathbf{G}_x\vec{u}_2\rangle\\ &= \langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{A}_1\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{A}_2\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_2, \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1 + \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle\,.\end{aligned}$$

With a block matrix notation write the above in the form

$$\begin{aligned}I_{\vec{\phi}} = I_{\phi_1} + I_{\phi_2} &\approx \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}, \begin{bmatrix}\mathbf{G}_x^T\mathbf{A}_1\mathbf{G}_x + \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_y & \mathbf{G}_x^T\mathbf{A}_2\mathbf{G}_y + \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_x\\ \mathbf{G}_y^T\mathbf{A}_2\mathbf{G}_x + \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_y & \mathbf{G}_y^T\mathbf{A}_1\mathbf{G}_y + \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_x\end{bmatrix}\begin{pmatrix}\vec{u}_1\\ \vec{u}_2\end{pmatrix}\rangle\\ &=: \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}, \mathbf{G}\begin{pmatrix}\vec{u}_1\\ \vec{u}_2\end{pmatrix}\rangle\,.\end{aligned}$$

The symmetric 12×12 matrix $\mathbf{G} \in \mathbb{R}^{12\times 12}$ is the element stiffness matrix for the triangle T, containing contributions to the integrals in (104).

8.4.3 The boundary integral

The boundary integral is similar to (62) on page 181, i.e. based on

$$\int_{-h/2}^{h/2} f(x)\, dx \approx \frac{h}{18}\, \left(5\, f(-\frac{\sqrt{3}}{2\sqrt{5}}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{3}}{2\sqrt{5}}\, h) \right) .$$

If the values of a function f at the two endpoints and the midpoint are denoted by (f_1, f_2, f_3) use a quadratic interpolation to find the values at the three Gauss integration points, given by

$$\begin{pmatrix} f(\vec{p}_1) \\ f(\vec{p}_2) \\ f(\vec{p}_3) \end{pmatrix} = \mathbf{M}_B \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \approx \begin{bmatrix} +0.68730 & 0.4 & -0.08730 \\ 0 & 1 & 0 \\ -0.08730 & 0.4 & +0.68730 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

and with the length L of the segment on the edge obtain the approximate integral

$$\begin{aligned} \int_{\text{edge}} g_1\, \phi_1 + g_2\, \phi_2\, ds &\approx \frac{L}{18}\, \langle \mathbf{M}_B \vec{\phi}_1\, , \begin{pmatrix} 5\, g_1(\vec{p}_1) \\ 8\, g_1(\vec{p}_2) \\ 5\, g_1(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18}\, \langle \mathbf{M}_B \vec{\phi}_2\, , \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle \\ &= \frac{L}{18}\, \langle \vec{\phi}_1\, , \mathbf{M}_B^T \begin{pmatrix} 5\, g_1(\vec{p}_1) \\ 8\, g_1(\vec{p}_2) \\ 5\, g_1(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18}\, \langle \vec{\phi}_2\, , \mathbf{M}_B^T \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle \end{aligned}$$

The integration weights can be combined with the interpolation matrix $\mathbf{M}_B$ by

$$\mathbf{M}_{BC} = \frac{1}{18}\, \mathbf{M}_B^T \begin{bmatrix} 5 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 5 \end{bmatrix} \approx \begin{bmatrix} 0.1909 & 0 & -0.0242 \\ 0.1111 & 0.4444 & 0.1111 \\ -0.0242 & 0 & 0.1909 \end{bmatrix} .$$

This matrix $\mathbf{M}_{BC}$ does not depend on the current edge segment and now use

$$\int_{\text{edge}} g_1\, \phi_1 + g_2\, \phi_2\, ds \approx L\, \langle \vec{\phi}_1\, , \mathbf{M}_{BC} \begin{pmatrix} g_1(\vec{p}_1) \\ g_1(\vec{p}_2) \\ g_1(\vec{p}_3) \end{pmatrix} \rangle + L\, \langle \vec{\phi}_2\, , \mathbf{M}_{BC} \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \\ g_2(\vec{p}_3) \end{pmatrix} \rangle .$$

The effect of the boundary integral on the global stiffness matrix and the vector is very similar to the approach shown at the end of Section 6.5.9.

8.4.4 Construct a weight matrix W

Use the same approach as for first order elements in Section 8.3.4 to approximate

$$\iint_\Omega \rho(x,y)\, (f_1(x,y)\, \phi_1(x,y) + f_2(x,y)\, \phi_2(x,y)\, dA \approx \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}\, , \begin{bmatrix} \mathbf{W}_1 & 0 \\ 0 & \mathbf{W}_2 \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle\, ,$$

where $\vec{\phi}_i$ and $\vec{f}_i$ are evaluated at the nodes. With the Gauss weights w_i and the values $\rho(\vec{g}_i)$ at the Gauss points $\vec{g}_i$ construct

$$\mathbf{W}_T = \text{diag} \begin{pmatrix} w_1\, \rho(\vec{g}_1) \\ w_2\, \rho(\vec{g}_2) \\ \vdots \\ w_7\, \rho(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} w_1\, \rho(\vec{g}_1) & & & \\ & w_2\, \rho(\vec{g}_2) & & \\ & & \ddots & \\ & & & w_7\, \rho(\vec{g}_7) \end{bmatrix} \in \mathbb{R}^{7\times 7} .$$

Use the interpolation matrix $\mathbf{M} \in \mathbb{R}^{7\times 6}$ from equation (56) to interpolate from the six nodes to the seven Gauss points. This leads to

$$\begin{aligned}\iint_T \rho\,(f_1\,\phi_1 + f_2\,\phi_2)\,dA &\approx \langle \mathbf{W}_T\mathbf{M}\,\vec{\phi}_1\,,\,\mathbf{M}\vec{f}_1\rangle + \langle \mathbf{W}_T\mathbf{M}\,\vec{\phi}_2\,,\,\mathbf{M}\vec{f}_2\rangle \\ &= \langle \vec{\phi}_1\,,\,\mathbf{M}\mathbf{W}_T\mathbf{M}\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}\mathbf{W}_T\mathbf{M}\vec{f}_2\rangle \\ &= \langle \begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}\,,\,\begin{bmatrix}\mathbf{M}^T\mathbf{W}_T\mathbf{M} & 0\\ 0 & \mathbf{M}^T\mathbf{W}_T\mathbf{M}\end{bmatrix}\begin{pmatrix}\vec{f}_1\\ \vec{f}_2\end{pmatrix}\rangle\end{aligned}$$

Thus the matrix to use is $\mathbf{M}^T\mathbf{W}_T\mathbf{M} \in \mathbb{R}^{6\times 6}$.

8.4.5 The contribution by a thermal expansion

The additional contribution by a thermal expansion is shown in (105).

$$I_{therm}(\vec{u}+\vec{\phi}) - I_{therm}(\vec{u}) \approx \Delta I_{therm} = \iint_T -\frac{E\,\alpha\,\Delta T}{1-\nu}\,(\frac{\partial\,\phi_1}{\partial x} + \frac{\partial\,\phi_2}{\partial y})\,dA$$

- The function $\text{th}(\vec{x}) = \frac{E(\vec{x})\,\alpha(\vec{x})\,\Delta T(\vec{x})}{1-\nu(x)}$ is given at the Gauss points, leading to a weight vector

$$\vec{w}_{therm} = \text{area}(T)\,(w_1\,\text{th}(\vec{g}_1),\ w_2\,\text{th}(\vec{g}_2),\ \ldots,\ w_7\,\text{th}(\vec{g}_7))^T \in \mathbb{R}^7$$

- For second order elements the partial derivatives $\frac{\partial\,\phi_1}{\partial x}$ and $\frac{\partial\,\phi_2}{\partial y}$ are required at the Gauss points in each triangle T and given by

$$\frac{\partial\,\phi_1}{\partial x} = \mathbf{G_x}\,\vec{\phi}_1 \quad \text{and} \quad \frac{\partial\,\phi_2}{\partial y} = \mathbf{G_y}\,\vec{\phi}_2\ .$$

This leads to

$$\Delta I_{therm} \approx -\langle \vec{w}_{therm}, \mathbf{G_x}\,\vec{\phi}_1\rangle - \langle \vec{w}_{therm}, \mathbf{G_y}\,\vec{\phi}_2\rangle = -\langle {\mathbf{G_x}}^T\vec{w}_{therm}, \vec{\phi}_1\rangle - \langle {\mathbf{G_y}}^T\vec{w}_{therm},\ \vec{\phi}_2\rangle\ .$$

The identical construction can be used for third order elements.

8.5 Construction of third order elements

The methods in this section are a combination of the tools used to construct third order elements for elliptic problems (Section 6.6) and the methods in the previous Section 8.4 to construct second order elements.

8.5.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$

Use the Gauss weights $\vec{w} \in \mathbb{R}^7$ from equation (49) on page 167 for the approximate integration over one triangle T.

- If the values of the functions f_1 and f_2 at the seven Gauss points are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2 \in \mathbb{R}^7$, then use the approximation

$$\begin{aligned}\iint_T f_1\,\phi_1 + f_2\,\phi_2\,dA &\approx \text{area}(T)\,\left(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\right) \\ &= \text{area}(T)\,\left(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\right) \\ &= \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}\,,\,\begin{bmatrix}\mathbf{M}^T\,\text{diag}(\vec{w}) & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\,\text{diag}(\vec{w})\end{bmatrix}\begin{pmatrix}\vec{f}_1\\ \vec{f}_2\end{pmatrix}\rangle\ .\end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 10}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (67) on page 184.

- If the values of the functions f_1 and f_2 at the nodes are denoted by the vectors $\vec{f}_1$ and $\vec{f}_2 \in \mathbb{R}^{10}$, then use the approximation

$$\begin{aligned}\iint\limits_T f_1\,\phi_1 + f_2\,\phi_2\,dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big)\\ &= \text{area}(T)\,\Big(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big)\ .\end{aligned}$$

Thus find one contribution to (104). Observe that $\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}$ is a 10×10 matrix, independent on the triangle T.

8.5.2 Integration of the terms involving derivatives of $\vec{u}$, ϕ_1 and ϕ_2

Using the results from Section 6.6 the partial derivatives at the nodes of functions ϕ given at the notes find for the first component $\varphi_x = \frac{\partial\,\varphi}{\partial x}$ of the gradient at the Gauss points

$$\begin{pmatrix}\varphi_x(\vec{x}_1)\\ \varphi_x(\vec{x}_2)\\ \vdots\\ \varphi_x(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\ \Big[\ (+y_3-y_1)\,\mathbf{M}_\xi^T + (-y_2+y_1)\,\mathbf{M}_\nu^T\ \Big]\cdot\vec{\phi} =: \mathbf{G}_x\,\vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix}\varphi_y(\vec{x}_1)\\ \varphi_y(\vec{x}_2)\\ \vdots\\ \varphi_y(\vec{x}_7)\end{pmatrix} = \frac{1}{\det(\mathbf{T})}\ \Big[\ (-x_3+x_1)\,\mathbf{M}_\xi^T + (+x_2-x_1)\,\mathbf{M}_\nu^T\ \Big]\cdot\vec{\phi} =: \mathbf{G}_y\,\vec{\phi}\,,$$

using the interpolation matrices $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ in equation (70) for the partial derivatives and the transformation rule (58) for the gradient. The matrices $\mathbf{G}_x$ and $\mathbf{G}_y$ are of size 7×10 and depend on the actual element, i.e. the triangle T.

Evaluate the coefficients E and ν at the Gauss points $\vec{g}_i$ and multiply by the Gauss integration weights w_i to obtain the three diagonal matrices

$$\mathbf{A}_1 = \text{diag}\begin{pmatrix} w_1\,\frac{E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)}\\ w_2\,\frac{E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)}\\ \vdots\\ w_7\,\frac{E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)}\end{pmatrix}\quad,\quad \mathbf{A}_2 = \text{diag}\begin{pmatrix} w_1\,\frac{\nu(\vec{g}_1)\,E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)}\\ w_2\,\frac{\nu(\vec{g}_2)\,E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)}\\ \vdots\\ w_7\,\frac{\nu(\vec{g}_7)\,E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)}\end{pmatrix}\quad\text{and}\quad \mathbf{A}_3 = \text{diag}\begin{pmatrix} w_1\,\frac{E(\vec{g}_1)}{2\,(1+\nu(\vec{g}_1))}\\ w_2\,\frac{E(\vec{g}_2)}{2\,(1+\nu(\vec{g}_2))}\\ \vdots\\ w_7\,\frac{E(\vec{g}_7)}{2\,(1+\nu(\vec{g}_7))}\end{pmatrix}\ .$$

This leads to the approximations

$$\begin{aligned}\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA\\ &\approx \langle \mathbf{A}_1\,\mathbf{G}_x\vec{\phi}_1, \mathbf{G}_x\vec{u}_1\rangle + \langle \text{diag}\,\mathbf{A}_2\,\mathbf{G}_x\vec{\phi}_1, \mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_3\,\mathbf{G}_y\vec{\phi}_1, \mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle\\ &= \langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{A}_1\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_1, \mathbf{G}_x^T\mathbf{A}_2\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_1, \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1+\mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle\\ \frac{I_{\phi_2}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA\\ &\approx \langle \mathbf{A}_1\mathbf{G}_y\vec{\phi}_2, \mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_2\mathbf{G}_y\vec{\phi}_2, \mathbf{G}_x\vec{u}_1\rangle + \langle \mathbf{A}_3\mathbf{G}_x\vec{\phi}_2, \mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle\\ &= \langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{A}_1\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_2, \mathbf{G}_y^T\mathbf{A}_2\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_2, \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1+\mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle\ .\end{aligned}$$

With a block matrix notation write the above in the form

$$\begin{aligned}I_{\vec{\phi}} = I_{\phi_1} + I_{\phi_2} &\approx \text{area}(T)\,\langle\begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}, \begin{bmatrix}\mathbf{G}_x^T\mathbf{A}_1\mathbf{G}_x + \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_y & \mathbf{G}_x^T\mathbf{A}_2\mathbf{G}_y + \mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_x\\ \mathbf{G}_y^T\mathbf{A}_2\mathbf{G}_x + \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_y & \mathbf{G}_y^T\mathbf{A}_1\mathbf{G}_y + \mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_x\end{bmatrix}\begin{pmatrix}\vec{u}_1\\ \vec{u}_2\end{pmatrix}\rangle\\ &=: \text{area}(T)\,\langle\begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix}, \mathbf{G}\begin{pmatrix}\vec{u}_1\\ \vec{u}_2\end{pmatrix}\rangle\ .\end{aligned}$$

The symmetric 20×20 matrix $\mathbf{G} \in \mathbb{R}^{20\times 20}$ is the element stiffness matrix for the triangle T, containing contributions to the integrals in (104).

8.5.3 The boundary integral

The boundary integral is similar to Section (6.6.7) on page 190, i.e. based on

$$\int_{-h/2}^{h/2} f(x)\, dx \approx \frac{h}{18}\left(5\, f(-\frac{\sqrt{3}}{2\sqrt{5}}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{3}}{2\sqrt{5}}\, h)\right) .$$

If the values of a function f at the two endpoints and the two points on the edge are denoted by $(f_{-2}, f_{-1}, f_{+1}, f_{+2})$ use a cubic interpolation to find the values at the three Gauss integration points, given by

$$\begin{pmatrix} u(\vec{p}_1) \\ u(\vec{p}_2) \\ u(\vec{p}_3) \end{pmatrix} = \mathbf{M}_B \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

and with the length L of the segment on the edge obtain the approximate integral

$$\begin{aligned} \int_{\text{edge}} g_1\, \phi_1 + g_2\, \phi_2\, ds &\approx \frac{L}{18}\, \langle \mathbf{M}_B \vec{\phi}_1\, , \begin{pmatrix} 5\, g_1(\vec{p}_1) \\ 8\, g_1(\vec{p}_2) \\ 5\, g_1(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18}\, \langle \mathbf{M}_B \vec{\phi}_2\, , \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle \\ &= \frac{L}{18}\, \langle \vec{\phi}_1\, , \mathbf{M}_B^T \begin{pmatrix} 5\, g_1(\vec{p}_1) \\ 8\, g_1(\vec{p}_2) \\ 5\, g_1(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18}\, \langle \vec{\phi}_2\, , \mathbf{M}_B^T \begin{pmatrix} 5\, g_2(\vec{p}_1) \\ 8\, g_2(\vec{p}_2) \\ 5\, g_2(\vec{p}_3) \end{pmatrix} \rangle . \end{aligned}$$

The integration weights can be combined with the interpolation matrix $\mathbf{M}_B$ by

$$\mathbf{M}_{BC} = \frac{1}{18}\, \mathbf{M}_B^T \begin{bmatrix} 5 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 5 \end{bmatrix} \approx \begin{bmatrix} 0.1356 & -0.0278 & 0.0172 \\ 0.2077 & 0.2500 & -0.0827 \\ -0.0827 & 0.2500 & 0.2077 \\ 0.0172 & -0.0278 & 0.1356 \end{bmatrix} .$$

This matrix $\mathbf{M}_{BC}$ does not depend on the current edge segment and leads to

$$\int_{\text{edge}} g_1\, \phi_1 + g_2\, \phi_2\, ds \approx L\, \langle \vec{\phi}_1\, , \mathbf{M}_{BC} \begin{pmatrix} g_1(\vec{p}_1) \\ g_1(\vec{p}_2) \\ g_1(\vec{p}_3) \end{pmatrix} \rangle + L\, \langle \vec{\phi}_2\, , \mathbf{M}_{BC} \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \\ g_2(\vec{p}_3) \end{pmatrix} \rangle .$$

The effect of the boundary integral on the global stiffness matrix and the vector is very similar to the approach shown at the end of Section 6.6.7.

8.5.4 Construct a weight matrix W

Use the same approach as for first or second order elements in Sections 8.3.4 and 8.4.4. Use the interpolation matrix $\mathbf{M} \in \mathbb{R}^{7\times 10}$ from equation (67) to interpolate from the ten nodes to the seven Gauss points. This leads to

$$\begin{aligned} \iint_T \rho\, (f_1\, \phi_1 + f_2\, \phi_2)\, dA &\approx \langle \mathbf{W}_T \mathbf{M}\, \vec{\phi}_1\, , \mathbf{M} \vec{f}_1 \rangle + \langle \mathbf{W}_T \mathbf{M}\, \vec{\phi}_2\, , \mathbf{M} \vec{f}_2 \rangle \\ &= \langle \vec{\phi}_1\, , \mathbf{M}^T \mathbf{W}_T \mathbf{M} \vec{f}_1 \rangle + \langle \vec{\phi}_2\, , \mathbf{M}^T \mathbf{W}_T \mathbf{M} \vec{f}_2 \rangle \\ &= \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}\, , \begin{bmatrix} \mathbf{M}^T \mathbf{W}_T \mathbf{M} & 0 \\ 0 & \mathbf{M}^T \mathbf{W}_T \mathbf{M} \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle \end{aligned}$$

Thus the matrix to use is $\mathbf{M}^T \mathbf{W}_T \mathbf{M} \in \mathbb{R}^{10\times 10}$.

8.6 The plane strain problem

For a plane strain problem it is assumed that there are no strains in z–direction, i.e.

$$\varepsilon_{xz} = \varepsilon_{yz} = \varepsilon_z = 0 \; .$$

With the modified material parameters in equation (36) $\nu^\star = \frac{\nu}{1-\nu}$ and $E^\star = \frac{E}{1-\nu^2}$ this leads to a simplification of Hooke's law.

$$\begin{aligned}
\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} &= \frac{E}{(1+\nu)(1-2\,\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\,\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \\
&= \frac{E^\star}{(1-\nu^\star)(1+\nu^\star)} \begin{bmatrix} 1 & \nu^\star & 0 \\ \nu^\star & 1 & 0 \\ 0 & 0 & 1-\nu^\star \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \\
\sigma_z &= \frac{E\;\nu\;(\varepsilon_{xx}+\varepsilon_{yy})}{(1+\nu)(1-2\,\nu)}
\end{aligned}$$

This is very similar to Hooke's law (27) for the plane stress situation, but with $E^\star$ and $\nu^\star$ instead of E and ν. The energy density is in this case given by

$$\begin{aligned}
W_{strain} &= \frac{1}{2}\,\frac{E}{(1+\nu)(1-2\,\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2\,(1-2\,\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle \\
&= \frac{E\,(1-\nu)}{2\,(1+\nu)\,(1-2\,\nu)} \left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\frac{\nu}{1-\nu}\,\varepsilon_{xx}\varepsilon_{yy} + 2\,\frac{1-2\,\nu}{1-\nu}\,\varepsilon_{xy}^2 \right) \\
&= \frac{E^\star}{2\,(1-(\nu^\star)^2)} \left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu^\star\,\varepsilon_{xx}\varepsilon_{yy} + 2\,(1-\nu^\star)\,\varepsilon_{xy}^2 \right)
\end{aligned} \tag{107}$$

This is very similar to the elastic energy density (28) for plane stress problems.

As a consequence of the similarity of the plane strain and plane stress problem there is no need for extensive new codes for plane strain problems. It is sufficient to write a wrapper to modify the material parameters.

$$E \;\to\; E^\star = \frac{E}{1-\nu^2} \geq E \quad \text{and} \quad \nu \;\to\; \nu^\star = \frac{\nu}{1-\nu} \geq \nu$$

If thermal expansion is taken into account there is an additional contribution of

$$-\frac{E\,\alpha\,\Delta T}{1-2\,\nu}\,(\varepsilon_{xx}+\varepsilon_{yy}) = -\frac{E\,\alpha\,\Delta T}{1-2\,\nu}\,(\frac{\partial\,u_1}{\partial x} + \frac{\partial\,u_2}{\partial y}) \; .$$

This leads to

$$I_{therm}(\vec{u}+\vec{\phi}) - I_{therm}(\vec{u}) \approx \iint\limits_{\Omega} -\frac{E\,\alpha\,\Delta T}{1-2\,\nu}\,(\frac{\partial\,\phi_1}{\partial x} + \frac{\partial\,\phi_2}{\partial y})\;dA \tag{108}$$

Evaluate the values ofthe function $\mathrm{th}(\vec{x}) = \frac{E(\vec{x})\,\alpha(\vec{x})\,\Delta T(\vec{x})}{1-2\,\nu(x)}$ at the Gauss points, leading to a weight vector

$$\vec{w}_{therm} = \mathrm{area}(T)\,(w_1\,\mathrm{th}(\vec{g}_1),\; w_2\,\mathrm{th}(\vec{g}_2),\; \ldots,\; w_7\,\mathrm{th}(\vec{g}_7))^T \in \mathbb{R}^7$$

used for the integration, i.e.

$$\Delta I_{therm} \approx -\langle \vec{w}_{therm}, \mathbf{G_x}\,\vec{\phi}_1\rangle - \langle \vec{w}_{therm}, \mathbf{G_y}\,\vec{\phi}_2\rangle = -\langle \mathbf{G_x}^T\vec{w}_{therm}, \vec{\phi}_1\rangle - \langle \mathbf{G_y}^T\vec{w}_{therm}, \vec{\phi}_2\rangle \; .$$

8.7 Elasticity for axially symmetric setups

For functions $u_r(r,z)$ and $u_z(r,z)$ examine displacements of the form

$$\begin{pmatrix} u_1(x,y,z) \\ u_2(x,y,z) \\ u_3(x,y,z) \end{pmatrix} = \begin{pmatrix} u_r(r,z)\,\cos\varphi \\ u_r(r,z)\,\sin\varphi \\ u_z(r,z) \end{pmatrix} .$$

and the total energy to be minimized is given by expression (40) on page 25.

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint\limits_{\Omega} \frac{2\,\pi\,r\,E}{2\,(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,(\varepsilon_{rr}^2 + \varepsilon_{zz}^2 + \frac{1}{r^2}\,u_r^2) + 2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz} + \frac{1}{r}\,u_r\,(\varepsilon_{rr}+\varepsilon_{zz})) \right) dA + \\ &\quad + \iint\limits_{\Omega} \frac{2\,\pi\,r\,E}{1+\nu}\,\varepsilon_{rz}^2\,dA - \iint\limits_{\Omega} 2\,\pi\,r\,\vec{f}\cdot\vec{u}\,dA - \int_{\Gamma_2} 2\,\pi\,r\,\vec{g}_N\cdot\vec{u}\,ds\ . \end{aligned}$$

Use the strains

$$\begin{pmatrix} \varepsilon_{rr} \\ \varepsilon_{\phi\phi} \\ \varepsilon_{zz} \\ \varepsilon_{rz} \end{pmatrix} = \begin{pmatrix} \frac{\partial\,u_r}{\partial r} \\ \frac{1}{r}\,u_r \\ \frac{\partial\,u_z}{\partial z} \\ \frac{1}{2}\,(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}) \end{pmatrix}$$

to rewrite the above with the displacement functions u_r and u_z in the form

$$\begin{aligned} \frac{U(\vec{u})}{2\,\pi} &= \iint\limits_{\Omega} \frac{r\,E}{2\,(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,((\frac{\partial\,u_r}{\partial r})^2 + (\frac{\partial\,u_z}{\partial z})^2 + \frac{1}{r^2}\,u_r^2) + \right. \\ &\qquad \left. + 2\,\nu\,((\frac{\partial\,u_r}{\partial r})(\frac{\partial\,u_z}{\partial z}) + \frac{1}{r}\,u_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z})) \right) dA + \\ &\quad + \iint\limits_{\Omega} \frac{r\,E}{1+\nu}\,\frac{1}{4}\,(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r})^2\,dA - \\ &\quad - \iint\limits_{\Omega} r\,(f_r\,u_r + f_z\,u_z)\,dA - \int_{\Gamma_2} r\,(g_{Nr}\,u_r + g_{Nz}\,u_z)\,ds\ . \end{aligned}$$

Expanding and ignoring quadratic terms of the perturbation $\vec{\phi}$ leads to

$$\begin{aligned} \frac{U(\vec{u}+\vec{\phi})}{2\,\pi} &\approx \frac{U(\vec{u})}{2\,\pi} + \iint\limits_{\Omega} \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \left((1-\nu) \left(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r} + \frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z} + \frac{1}{r^2}\,u_r\,\phi_r \right) + \right. \\ &\qquad \left. + \nu \left((\frac{\partial\,u_r}{\partial r})(\frac{\partial\,\phi_z}{\partial z}) + (\frac{\partial\,\phi_r}{\partial r})(\frac{\partial\,u_z}{\partial z}) + \frac{1}{r}\,u_r\,(\frac{\partial\,\phi_r}{\partial r} + \frac{\partial\,\phi_z}{\partial z}) + \frac{1}{r}\,\phi_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z}) \right) \right) dA + \\ &\quad + \iint\limits_{\Omega} \frac{r\,E}{1+\nu}\,\frac{1}{2} \left(\frac{\partial\,u_r}{\partial z}\,\frac{\partial\,\phi_r}{\partial z} + \frac{\partial\,u_z}{\partial r}\,\frac{\partial\,\phi_z}{\partial r} + \frac{\partial\,u_r}{\partial z}\,\frac{\partial\,\phi_z}{\partial r} + \frac{\partial\,\phi_r}{\partial z}\,\frac{\partial\,u_z}{\partial r} \right) dA - \\ &\quad - \iint\limits_{\Omega} r\,(f_r\,\phi_r + f_z\,\phi_z)\,dA - \int_{\Gamma_2} r\,(g_{Nr}\,\phi_r + g_{Nz}\,\phi_z)\,ds\ . \end{aligned}$$

These integrals can be separated into contributions with ϕ_r, ϕ_z, f and g_N.

$$\frac{U(\vec{u}+\vec{\phi})}{2\,\pi} \approx \frac{U(\vec{u})}{2\,\pi} + I_{\phi_r} + I_{\phi_z} + I_f + I_g \tag{109}$$

$$\begin{aligned} I_{\phi_r} &= \iint\limits_{\Omega} \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r^2}\,u_r\,\phi_r) + \right. \\ &\qquad \left. + \nu \left(\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,u_r\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,\phi_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z}) \right) \right) + \end{aligned} \tag{110}$$

$$
\begin{aligned}
& + \frac{r\,E}{2\,(1+\nu)} \left(\frac{\partial\, u_r}{\partial z} + \frac{\partial\, u_z}{\partial r} \right) \frac{\partial\, \phi_r}{\partial z}\; dA \\
I_{\phi_z} \;=\; & \iint\limits_{\Omega} \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\, \frac{\partial\, u_z}{\partial z}\, \frac{\partial\, \phi_z}{\partial z} + \nu\, (\frac{\partial\, u_r}{\partial r}\, \frac{\partial\, \phi_z}{\partial z} + \frac{1}{r}\, u_r\, \frac{\partial\, \phi_z}{\partial z}) \right) + \qquad (111) \\
& + \frac{r\,E}{2\,(1+\nu)} \left(\frac{\partial\, u_r}{\partial z} + \frac{\partial\, u_z}{\partial r} \right) \frac{\partial\, \phi_z}{\partial r}\; dA \\
I_f \;=\; & -\iint\limits_{\Omega} r\,(f_r\,\phi_r + f_z\,\phi_z)\; dA \qquad (112) \\
I_g \;=\; & -\int\limits_{\Gamma_2} r\,(g_{Nr}\,\phi_r + g_{Nz}\,\phi_z)\; ds \qquad (113)
\end{aligned}
$$

Using these integrals derive the FEM algorithm for elements of order 1, 2 and 3.

8.8 Construction of first order elements

This is similar to the computations in Section 8.3, starting in page 227. In this section the element stiffness matrix is constructed. Then use the procedure in Section 6.4 (page 167) to determine the global stiffness matrix.

8.8.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$

Evaluate the radius r at the three Gauss points of the triangle T, leading to the diagonal matrix $\mathbf{R} = \mathrm{diag}([r_1, r_2, r_3])$ and use the interpolation matrix from the corners to the Gauss points

$$
\mathbf{M} = \frac{1}{6} \begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix} .
$$

- If the values of the functions f_r and f_z at the Gauss points are denoted by the vectors $\vec{f_r}$ and $\vec{f_z}$, then use the approximation

$$
\begin{aligned}
\iint\limits_{T} r\,(f_r\,\phi_r + f_z\,\phi_z)\; dA \;&\approx\; \frac{\mathrm{area}(T)}{3} \left(\langle \mathbf{M}\,\vec{\phi}_r\,,\, \mathbf{R}\vec{f_r} \rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\, \mathbf{R}\vec{f_z} \rangle \right) \\
&=\; \frac{\mathrm{area}(T)}{3} \left(\langle \vec{\phi}_r\,,\, \mathbf{M}^T\mathbf{R}\vec{f_r} \rangle + \langle \vec{\phi}_z\,,\, \mathbf{M}^T\mathbf{R}\vec{f_z} \rangle \right) .
\end{aligned}
$$

- If the values of the functions f_r and f_z at the nodes are denoted by the vectors $\vec{f_r}$ and $\vec{f_z}$, then use the approximation

$$
\begin{aligned}
\iint\limits_{T} r\,(f_r\,\phi_r + f_z\,\phi_z)\; dA \;&\approx\; \frac{\mathrm{area}(T)}{3} \left(\langle \mathbf{M}\,\vec{\phi}_r\,,\, \mathbf{R}\mathbf{M}\,\vec{f_r} \rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\, \mathbf{R}\mathbf{M}\,\vec{f_z} \rangle \right) \\
&=\; \frac{\mathrm{area}(T)}{3} \left(\langle \vec{\phi}_r\,,\, \mathbf{M}^T\mathbf{R}\mathbf{M}\,\vec{f_r} \rangle + \langle \vec{\phi}_z\,,\, \mathbf{M}^T\mathbf{R}\mathbf{M}\,\vec{f_z} \rangle \right)
\end{aligned}
$$

With the above the contributions in (112) for each element stiffness matrix can be determined.

8.8.2 Integration of the terms involving derivatives of ϕ_z and ϕ_z

For linear elements the gradient of the functions u_i and ϕ_i are constant and using equation (52) given by

$$
\nabla u = \frac{-1}{2\,\mathrm{area}(T)} \begin{bmatrix} (z_3 - z_2) & (z_1 - z_3) & (z_2 - z_1) \\ (r_2 - r_3) & (r_3 - r_1) & (r_1 - r_2) \end{bmatrix} \cdot \vec{u} = \begin{bmatrix} \mathbf{G}_r \\ \mathbf{G}_z \end{bmatrix} \vec{u}\,.
$$

Evaluate the coefficients E and ν at the Gauss points $\vec{g}_i$ and define the average values a_j, vector $\vec{a}_4$ and the diagonal matrix $\mathbf{A}_2$. Since the derivatives of order one of the displacements are piecewise constant, some of the expressions

require less computational effort to determine.

$$
\begin{aligned}
a_1 &= \frac{\text{area}(T)}{3} \sum_{i=1}^{3} \frac{r(\vec{g}_i)\, E(\vec{g}_i)\, (1-\nu(\vec{g}_i))}{(1+\nu(\vec{g}_i))\, (1-2\,\nu(\vec{g}_i))} \\
\mathbf{A}_2 &= \frac{\text{area}(T)}{3}\ \text{diag}\Big(\frac{E(\vec{g}_i)\, (1-\nu(\vec{g}_i))}{r(\vec{g}_i)\, (1+\nu(\vec{g}_i))\, (1-2\,\nu(\vec{g}_i))}\Big) \\
a_3 &= \frac{\text{area}(T)}{3} \sum_{i=1}^{3} \frac{r(\vec{g}_i)\, E(\vec{g}_i)\, \nu(\vec{g}_i)}{(1+\nu(\vec{g}_i))\, (1-2\,\nu(\vec{g}_i))} \\
(\vec{a}_4)_i &= \frac{\text{area}(T)}{3}\ \frac{E(\vec{g}_i)\, \nu(\vec{g}_i)}{(1+\nu(\vec{g}_i))\, (1-2\,\nu(\vec{g}_i))} \\
a_5 &= \frac{\text{area}(T)}{3} \sum_{i=1}^{3} \frac{r(\vec{g}_i)\, E(\vec{g}_i)}{2\,(1+\nu(\vec{g}_i))}
\end{aligned}
$$

This leads to the approximate integrals

$$
\begin{aligned}
I_{\phi_r} &= \iint_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \Big((1-\nu)\,(\frac{\partial\, u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r^2}\, u_r\,\phi_r) + \\
&\qquad + \nu\,\Big(\frac{\partial\, u_z}{\partial z}\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\, u_r\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,\phi_r\,(\frac{\partial\, u_r}{\partial r} + \frac{\partial\, u_z}{\partial z})\Big)\Big) + \\
&\qquad + \frac{r\,E}{2\,(1+\nu)}\,\Big(\frac{\partial\, u_r}{\partial z} + \frac{\partial\, u_z}{\partial r}\Big)\,\frac{\partial\,\phi_r}{\partial z}\; dA \\
&\approx a_1\,\langle \mathbf{G}_r\vec{\phi}_r, \mathbf{G}_r\vec{u}_r\rangle + \langle \mathbf{M}\vec{\phi}_r, \mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + a_3\,\langle \mathbf{G}_r\vec{\phi}_r, \mathbf{G}_z\vec{u}_z\rangle + \\
&\quad + \langle \vec{a}_4\mathbf{G}_r\vec{\phi}_r, \mathbf{M}\vec{u}_r\rangle + \langle \mathbf{M}\vec{\phi}_r, \vec{a}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + a_5\,\langle \mathbf{G}_z\vec{\phi}_r, (\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= a_1\,\langle \vec{\phi}_r, \mathbf{G}_r^T\mathbf{G}_r\vec{u}_r\rangle + \langle \vec{\phi}_r, \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + a_3\,\langle \vec{\phi}_r, \mathbf{G}_r^T\mathbf{G}_z\vec{u}_z\rangle + \\
&\quad + \langle \vec{\phi}_r, (\vec{a}_4\mathbf{G}_r)^T\mathbf{M}\vec{u}_r\rangle + \langle \vec{\phi}_r, \mathbf{M}^T\vec{a}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + a_5\,\langle \vec{\phi}_r, \mathbf{G}_z^T(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= \langle a_1\,\mathbf{G}_r^T\mathbf{G}_r\vec{u}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r + a_3\,\mathbf{G}_r^T\mathbf{G}_z\vec{u}_z + (\vec{a}_4\mathbf{G}_r)^T\mathbf{M}\vec{u}_r, \vec{\phi}_r\rangle + \\
&\quad + \langle \mathbf{M}^T\vec{a}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z) + a_5\,\mathbf{G}_z^T(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z), \vec{\phi}_r\rangle \\
&= \langle \big(a_1\,\mathbf{G}_r^T\mathbf{G}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M} + (\vec{a}_4\mathbf{G}_r)^T\mathbf{M} + \mathbf{M}^T\vec{a}_4\,\mathbf{G}_r + a_5\,\mathbf{G}_z^T\mathbf{G}_z\big)\ \vec{u}_r, \vec{\phi}_r\rangle \\
&\quad + \langle \big(a_3\,\mathbf{G}_r^T\mathbf{G}_z + \mathbf{M}^T\vec{a}_4\,\mathbf{G}_z + a_5\,\mathbf{G}_z^T\mathbf{G}_r\big)\ \vec{u}_z, \vec{\phi}_r\rangle
\end{aligned}
$$

and

$$
\begin{aligned}
I_{\phi_z} &= \iint_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \Big((1-\nu)\,\frac{\partial\, u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z} + \nu\,(\frac{\partial\, u_r}{\partial r}\,\frac{\partial\,\phi_z}{\partial z} + \frac{1}{r}\, u_r\,\frac{\partial\,\phi_z}{\partial z})\Big) + \\
&\qquad + \frac{r\,E}{2\,(1+\nu)}\,\Big(\frac{\partial\, u_r}{\partial z} + \frac{\partial\, u_z}{\partial r}\Big)\,\frac{\partial\,\phi_z}{\partial r}\; dA \\
&\approx a_1\,\langle \mathbf{G}_z\vec{\phi}_z, \mathbf{G}_z\vec{u}_z\rangle + a_3\,\langle \mathbf{G}_z\vec{\phi}_z, \mathbf{G}_r\vec{u}_r\rangle + \langle \vec{a}_4\mathbf{G}_z\vec{\phi}_z, \mathbf{M}\vec{u}_r\rangle + a_5\,\langle \mathbf{G}_r\vec{\phi}_z, (\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= a_1\,\langle \vec{\phi}_z, \mathbf{G}_z^T\mathbf{G}_z\vec{u}_z\rangle + a_3\,\langle \vec{\phi}_z, \mathbf{G}_z^T\mathbf{G}_r\vec{u}_r\rangle + \langle \vec{\phi}_z, (\vec{a}_4\mathbf{G}_z)^T\mathbf{M}\vec{u}_r\rangle + a_5\,\langle \vec{\phi}_z, \mathbf{G}_r^T(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= \langle a_1\,\mathbf{G}_z^T\mathbf{G}_z\vec{u}_z + a_3\,\mathbf{G}_z^T\mathbf{G}_r\vec{u}_r + (\vec{a}_4\mathbf{G}_z)^T\mathbf{M}\vec{u}_r + a_5\,\mathbf{G}_r^T(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z), \vec{\phi}_z\rangle \\
&= \langle \big(a_3\,\mathbf{G}_z^T\mathbf{G}_r + (\vec{a}_4\mathbf{G}_z)^T\mathbf{M} + a_5\,\mathbf{G}_r^T\mathbf{G}_z\big)\,\vec{u}_r, \vec{\phi}_z\rangle + \langle \big(a_1\,\mathbf{G}_z^T\mathbf{G}_z + a_5\,\mathbf{G}_r^T\mathbf{G}_r\big)\,\vec{u}_z, \vec{\phi}_z\rangle
\end{aligned}
$$

All of the above contributions are of the form $\langle \mathbf{A}_r\vec{u}_{r,z}, \vec{\phi}_r\rangle$ or $\langle \mathbf{A}_z\vec{u}_{r,z}, \vec{\phi}_z\rangle$ and thus contributions to the element stiffness matrix $\mathbf{A} \in \mathbb{M}^{6\times 6}$.

$$
\langle \mathbf{A} \begin{pmatrix} \vec{u}_r \\ \vec{u}_z \end{pmatrix},\ \begin{pmatrix} \vec{\phi}_r \\ \vec{\phi}_z \end{pmatrix}\rangle = \langle \begin{bmatrix} \mathbf{A}_{r,r} & \mathbf{A}_{r,z} \\ \mathbf{A}_{z,r} & \mathbf{A}_{z,z} \end{bmatrix} \begin{pmatrix} \vec{u}_r \\ \vec{u}_z \end{pmatrix},\ \begin{pmatrix} \vec{\phi}_r \\ \vec{\phi}_z \end{pmatrix}\rangle
$$

The 3×3 submatrices (e.g. $\mathbf{A}_{r,z}$) satisfy $\mathbf{A}_{r,z} = \mathbf{A}_{r,z}^T$ and $\mathbf{A}_{r,r}$, $\mathbf{A}_{z,z}$ are symmetric. For sake of completeness the symmetric 6×6 element stiffness matrix.

$$
\mathbf{A} = \begin{bmatrix} a_1\mathbf{G}_r^T\mathbf{G}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M} + (\vec{a}_4\mathbf{G}_r)^T\mathbf{M} + \mathbf{M}^T\vec{a}_4\mathbf{G}_r + a_5\mathbf{G}_z^T\mathbf{G}_z & a_3\mathbf{G}_r^T\mathbf{G}_z + \mathbf{M}^T\vec{a}_4\mathbf{G}_z + a_5\mathbf{G}_z^T\mathbf{G}_r \\ a_3\mathbf{G}_z^T\mathbf{G}_r + (\vec{a}_4\mathbf{G}_z)^T\mathbf{M} + a_5\mathbf{G}_r^T\mathbf{G}_z & a_1\mathbf{G}_z^T\mathbf{G}_z + a_5\mathbf{G}_r^T\mathbf{G}_r \end{bmatrix}
$$

8.8.3 The boundary integral

The boundary integral is similar to (53) on page 172. With $\alpha = \frac{1-1/\sqrt{3}}{2} \approx 0.2113$ use the symmetric interpolation matrix from nodes to Gauss points

$$\mathbf{M}_b = \begin{bmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{bmatrix} = \mathbf{M}_b^T$$

to find the two Gauss points $\vec{p}_1$ and $\vec{p}_2$ and to evaluate the radius r at the Gauss points, leading to the diagonal matrix $\mathbf{R} = \mathrm{diag}([r(\vec{p}_1), r(\vec{p}_2)])$. Then use the length L of the edge segment for the approximate integral

$$\begin{aligned} \int_{\text{edge}} r\,(g_{Nr}\,\phi_r + g_{Nz}\,\phi_z)\,ds &\approx \frac{L}{2}\,\langle \mathbf{M}_b\vec{\phi}_r, \mathbf{R}\,\vec{g}_{Nr}\rangle + \frac{L}{2}\,\langle \mathbf{M}_b\vec{\phi}_z, \mathbf{R}\,\vec{g}_{Nz}\rangle \\ &= \frac{L}{2}\,\langle \vec{\phi}_r, \mathbf{M}_b, \mathbf{R}\,\vec{g}_{Nr}\rangle + \frac{L}{2}\,\langle \vec{\phi}_z, \mathbf{M}_b\,\mathbf{R}\,\vec{g}_{Nz}\rangle\ , \end{aligned}$$

where the functions g_{Nr} and g_{Nz} are evaluated at the Gauss points.

8.9 Construction of second order elements

To construct elements of order 2 combine procedures from Section 8.4 for second order elements for plane stress problems and the previous section 8.8 where first order elements are generated for axisymmetric problems.

8.9.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$

Use the Gauss weights $\vec{w} \in \mathbb{R}^7$ from equation (49) on page 167 for the approximate integration over one triangle T and the vector $\vec{r} = (r_1, r_2, \ldots, r_7)^T$ of the radii at the Gauss points. With these construct the diagonal matrix

$$\mathbf{RW} = \mathrm{diag}([r_1w_1, r_2w_2, \ldots, r_7w_7]) \in \mathbb{M}^{7\times 7}\ .$$

- If the values of the functions f_r and f_z at the seven Gauss points are denoted by the vectors $\vec{f}_r$ and $\vec{f}_z \in \mathbb{R}^7$, then use the approximation

$$\begin{aligned} \iint_T r\,(f_r\,\phi_r + f_z\,\phi_z)\,dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_r\,,\,\mathrm{diag}(\vec{w})\,\vec{f}_r\rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\,\mathbf{RW}\,\vec{f}_z\rangle\Big) \\ &= \text{area}(T)\,\Big(\langle \vec{\phi}_r\,,\,\mathbf{M}^T\mathbf{RW}\,\vec{f}_r\rangle + \langle \vec{\phi}_z\,,\,\mathbf{M}^T\mathbf{RW}\,\vec{f}_z\rangle\Big) \\ &= \text{area}(T)\,\langle \begin{pmatrix} \vec{\phi}_r \\ \vec{\phi}_z \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\mathbf{RW} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\mathbf{RW} \end{bmatrix} \begin{pmatrix} \vec{f}_r \\ \vec{f}_z \end{pmatrix}\rangle\ . \end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 6}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (56) on page 174.

- If the values of the functions f_r and f_z at the nodes are denoted by the vectors $\vec{f}_r$ and $\vec{f}_r \in \mathbb{R}^6$, then use the approximation

$$\begin{aligned} \iint_T r\,(f_r\,\phi_r + f_z\,\phi_z)\,dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_r\,,\,\mathbf{RW}\,\mathbf{M}\,\vec{f}_r\rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\,\mathbf{RW}\,\mathbf{M}\,\vec{f}_z\rangle\Big) \\ &= \text{area}(T)\,\Big(\langle \vec{\phi}_r\,,\,\mathbf{M}^T\mathbf{RW}\,\mathbf{M}\,\vec{f}_r\rangle + \langle \vec{\phi}_z\,,\,\mathbf{M}^T\mathbf{RW}\,\mathbf{M}\,\vec{f}_z\rangle\Big) \\ &= \text{area}(T)\,\langle \begin{pmatrix} \vec{\phi}_r \\ \vec{\phi}_z \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\mathbf{RW}\,\mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\mathbf{RW}\,\mathbf{M} \end{bmatrix} \begin{pmatrix} \vec{f}_r \\ \vec{f}_z \end{pmatrix}\rangle\ . \end{aligned}$$

With the above the contributions in (112) for each element stiffness matrix can be determined. Observe that $\mathbf{M}^T\mathbf{RW}\,\mathbf{M}$ is a 6×6 matrix, independent on the triangle T.

8.9.2 Integration of the terms involving derivatives of ϕ_r and ϕ_z

Using the results from Section 6.5 the partial derivatives at the nodes of functions ϕ given at the notes find for the first component of the gradient at the Gauss points

$$\frac{\partial}{\partial r}\,\vec{\phi} = \frac{1}{\det(\mathbf{T})}\,\left[\;(+z_3 - z_1)\,\mathbf{M}_\xi^T + (-z_2 + z_1)\,\mathbf{M}_\nu^T\;\right]\cdot\vec{\phi} =: \mathbf{G}_r\,\vec{\phi}$$

and for the second component of the gradient

$$\frac{\partial}{\partial z}\,\vec{\phi} = \frac{1}{\det(\mathbf{T})}\,\left[\;(-r_3 + r_1)\,\mathbf{M}_\xi^T + (+r_2 - r_1)\,\mathbf{M}_\nu^T\;\right]\cdot\vec{\phi} =: \mathbf{G}_z\,\vec{\phi}\,.$$

Observe that the matrices $\mathbf{G}_r$ and $\mathbf{G}_z$ depend on the triangle T. Evaluate the coefficients E and ν at the Gauss points $\vec{g}_i$ and define diagonal matrices $\mathbf{A}_j$.

$$\begin{aligned}
\mathbf{A}_1 &= \text{area}(T)\;\text{diag}\Big(\frac{w_i\,r(\vec{g}_i)\,E(\vec{g}_i)\,(1-\nu(\vec{g}_i))}{(1+\nu(\vec{g}_i))\,(1-2\,\nu(\vec{g}_i))}\Big)\\
\mathbf{A}_2 &= \text{area}(T)\;\text{diag}\Big(\frac{w_i\,E(\vec{g}_i)\,(1-\nu(\vec{g}_i))}{r(\vec{g}_i)\,(1+\nu(\vec{g}_i))\,(1-2\,\nu(\vec{g}_i))}\Big)\\
\mathbf{A}_3 &= \text{area}(T)\;\text{diag}\Big(\frac{w_i\,r(\vec{g}_i)\,E(\vec{g}_i)\,\nu(\vec{g}_i)}{(1+\nu(\vec{g}_i))\,(1-2\,\nu(\vec{g}_i))}\Big)\\
\mathbf{A}_4 &= \text{area}(T)\;\text{diag}\Big(\frac{w_i\,E(\vec{g}_i)\,\nu(\vec{g}_i)}{(1+\nu(\vec{g}_i))\,(1-2\,\nu(\vec{g}_i))}\Big)\\
\mathbf{A}_5 &= \text{area}(T)\;\text{diag}\Big(\frac{w_i\,r(\vec{g}_i)\,E(\vec{g}_i)}{2\,(1+\nu(\vec{g}_i))}\Big)
\end{aligned}$$

This leads to the approximate integrals

$$\begin{aligned}
I_{\phi_r} &= \iint\limits_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\,\Big((1-\nu)\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r^2}\,u_r\,\phi_r) +\\
&\qquad +\nu\,\Big(\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,u_r\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,\phi_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z})\Big)\Big) +\\
&\qquad +\frac{r\,E}{2\,(1+\nu)}\,\Big(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}\Big)\,\frac{\partial\,\phi_r}{\partial z}\;dA\\
&\approx \langle\mathbf{G}_r\vec{\phi}_r, \mathbf{A}_1\mathbf{G}_r\vec{u}_r\rangle + \langle\mathbf{M}\vec{\phi}_r, \mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + \langle\mathbf{G}_r\vec{\phi}_r, \mathbf{A}_3\mathbf{G}_z\vec{u}_z\rangle +\\
&\quad +\langle\mathbf{G}_r\vec{\phi}_r, \mathbf{A}_4\mathbf{M}\vec{u}_r\rangle + \langle\mathbf{M}\vec{\phi}_r, \mathbf{A}_4(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + \langle\mathbf{G}_z\vec{\phi}_r, \mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\
&= \langle\vec{\phi}_r, \mathbf{G}_r^T\mathbf{A}_1\mathbf{G}_r\vec{u}_r\rangle + \langle\vec{\phi}_r, \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + \langle\vec{\phi}_r, \mathbf{G}_r^T\mathbf{A}_3\mathbf{G}_z\vec{u}_z\rangle +\\
&\quad +\langle\vec{\phi}_r, \mathbf{G}_r^T\mathbf{A}_4\mathbf{M}\vec{u}_r\rangle + \langle\vec{\phi}_r, \mathbf{M}^T\mathbf{A}_4(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + \langle\vec{\phi}_r, \mathbf{G}_z^T\mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\
&= \langle\mathbf{G}_r^T\mathbf{A}_1\mathbf{G}_r\vec{u}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r + \mathbf{G}_r^T\mathbf{A}_3\mathbf{G}_z\vec{u}_z + \mathbf{G}_r^T\mathbf{A}_4\mathbf{M}\vec{u}_r, \vec{\phi}_r\rangle +\\
&\quad +\langle\mathbf{M}^T\mathbf{A}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z) + \mathbf{G}_z^T\mathbf{A}_1(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z), \vec{\phi}_r\rangle\\
&= \langle\big(\mathbf{G}_r^T\mathbf{A}_1\mathbf{G}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M} + \mathbf{G}_r^T\mathbf{A}_4\mathbf{M} + \mathbf{M}^T\mathbf{A}_4\,\mathbf{G}_r + \mathbf{G}_z^T\mathbf{A}_5\mathbf{G}_z\big)\;\vec{u}_r, \vec{\phi}_r\rangle\\
&\quad +\langle\big(\mathbf{G}_r^T\mathbf{A}_3\mathbf{G}_z + \mathbf{M}^T\mathbf{A}_4\,\mathbf{G}_z + \mathbf{G}_z^T\mathbf{A}_5\mathbf{G}_r\big)\;\vec{u}_z, \vec{\phi}_r\rangle
\end{aligned}$$

and

$$\begin{aligned}
I_{\phi_z} &= \iint\limits_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\,\Big((1-\nu)\,\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z} + \nu\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_z}{\partial z} + \frac{1}{r}\,u_r\,\frac{\partial\,\phi_z}{\partial z})\Big) +\\
&\qquad +\frac{r\,E}{2\,(1+\nu)}\,\Big(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}\Big)\,\frac{\partial\,\phi_z}{\partial r}\;dA\\
&\approx \langle\mathbf{G}_z\vec{\phi}_z, \mathbf{A}_1\mathbf{G}_z\vec{u}_z\rangle + \langle\mathbf{G}_z\vec{\phi}_z, \mathbf{A}_3\mathbf{G}_r\vec{u}_r\rangle + \langle\mathbf{A}_4\mathbf{G}_z\vec{\phi}_z, \mathbf{M}\vec{u}_r\rangle + \langle\mathbf{G}_r\vec{\phi}_z, \mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\
&= \langle\vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z\vec{u}_z\rangle + \langle\vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r\vec{u}_r\rangle + \langle\vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_4\mathbf{M}\vec{u}_r\rangle + \langle\vec{\phi}_z, \mathbf{G}_r^T\mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\
&= \langle\mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z\vec{u}_z + \mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r\vec{u}_r + \mathbf{G}_z^T\mathbf{A}_4\mathbf{M}\vec{u}_r + \mathbf{G}_r^T\mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z), \vec{\phi}_z\rangle\\
&= \langle\big(\mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r + \mathbf{G}_z^T\mathbf{A}_4\mathbf{M} + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_z\big)\;\vec{u}_r, \vec{\phi}_z\rangle + \langle\big(\mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_r\big)\;\vec{u}_z, \vec{\phi}_z\rangle\;.
\end{aligned}$$

All of the above contributions are of the form $\langle \mathbf{A}_r \vec{u}_{r,z}, \vec{\phi}_r \rangle$ or $\langle \mathbf{A}_z \vec{u}_{r,z}, \vec{\phi}_z \rangle$ and thus contributions to the symmetric element stiffness matrix $\mathbf{A} \in \mathbb{M}^{12\times 12}$, where e.g. $\mathbf{A}_{r,z} \in \mathbb{M}^{6\times 6}$. For sake of completeness the symmeytric 12×12 element stiffness matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \begin{bmatrix} \mathbf{G}_r^T \mathbf{A}_1 \mathbf{G}_r + \mathbf{M}^T \mathbf{A}_2 \mathbf{M} + \mathbf{G}_r^T \mathbf{A}_4 \mathbf{M} + \mathbf{M}^T \mathbf{A}_4 \mathbf{G}_r + \mathbf{G}_z^T \mathbf{A}_5 \mathbf{G}_z & \mathbf{G}_r^T \mathbf{A}_3 \mathbf{G}_z + \mathbf{M}^T \mathbf{A}_4 \mathbf{G}_z + \mathbf{G}_z^T \mathbf{A}_5 \mathbf{G}_r \\ \mathbf{G}_z^T \mathbf{A}_3 \mathbf{G}_r + \mathbf{G}_z^T \mathbf{A}_4 \mathbf{M} + \mathbf{G}_r^T \mathbf{A}_5 \mathbf{G}_z & \mathbf{G}_z^T \mathbf{A}_1 \mathbf{G}_z + \mathbf{G}_r^T \mathbf{A}_5 \mathbf{G}_r \end{bmatrix} .$$

8.9.3 The boundary integral

The boundary integral is constructed similar to the procedures in Section 8.4.3, i.e. building on

$$\int_{-h/2}^{h/2} f(x)\, dx \approx \frac{h}{18} \left(5\, f(-\frac{\sqrt{3}}{2\sqrt{5}}\, h) + 8\, f(0) + 5\, f(\frac{\sqrt{3}}{2\sqrt{5}}\, h) \right) .$$

If the values of a function f at the two endpoints and the midpoint are denoted by $\vec{f} = (f_1, f_2, f_3)^T$ use a quadratic interpolation to find the values at the three Gauss integration points, given by $\mathbf{M}_B \vec{f}$ and evaluate the radii r_i at the Gauss points. With the length L of the segment on the edge obtain the approximate integral

$$\begin{aligned} \int_{\text{edge}} r\,(g_r\, \phi_r + g_z\, \phi_z)\, ds &\approx \frac{L}{18} \langle \mathbf{M}_B \vec{\phi}_r\,, \begin{pmatrix} 5\, r_1\, g_r(\vec{p}_1) \\ 8\, r_2\, g_r(\vec{p}_2) \\ 5\, r_3\, g_r(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18} \langle \mathbf{M}_B \vec{\phi}_z\,, \begin{pmatrix} 5\, r_1\, g_z(\vec{p}_1) \\ 8\, r_2\, g_z(\vec{p}_2) \\ 5\, r_3\, g_z(\vec{p}_3) \end{pmatrix} \rangle \\ &= \frac{L}{18} \langle \vec{\phi}_r\,, \mathbf{M}_B^T \begin{pmatrix} 5\, r_1\, g_r(\vec{p}_1) \\ 8\, r_2\, g_r(\vec{p}_2) \\ 5\, r_3\, g_r(\vec{p}_3) \end{pmatrix} \rangle + \frac{L}{18} \langle \vec{\phi}_z\,, \mathbf{M}_B^T \begin{pmatrix} 5\, r_1\, g_z(\vec{p}_1) \\ 8\, r_2\, g_z(\vec{p}_2) \\ 5\, r_3\, g_z(\vec{p}_3) \end{pmatrix} \rangle . \end{aligned}$$

The integration weights can be combined with the interpolation matrix $\mathbf{M}_B$ by

$$\mathbf{M}_{BC} = \frac{1}{18} \mathbf{M}_B^T \begin{bmatrix} 5 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 5 \end{bmatrix} \approx \begin{bmatrix} 0.1909 & 0 & -0.0242 \\ 0.1111 & 0.4444 & 0.1111 \\ -0.0242 & 0 & 0.1909 \end{bmatrix} .$$

This matrix $\mathbf{M}_{BC}$ does not depend on the current edge segment and leads to

$$\int_{\text{edge}} r\,(g_r\, \phi_r + g_z\, \phi_z)\, ds \approx L\, \langle \vec{\phi}_r\,, \mathbf{M}_{BC} \begin{pmatrix} r_1\, g_r(\vec{p}_1) \\ r_2\, g_r(\vec{p}_2) \\ r_3\, g_r(\vec{p}_3) \end{pmatrix} \rangle + L\, \langle \vec{\phi}_z\,, \mathbf{M}_{BC} \begin{pmatrix} r_1\, g_z(\vec{p}_1) \\ r_2\, g_z(\vec{p}_2) \\ r_3\, g_z(\vec{p}_3) \end{pmatrix} \rangle .$$

The effect of the boundary integral on the global stiffness matrix and the vector is very similar to the approach shown at the end of Section 6.5.9.

8.10 Construction of third order elements

To construct elements of order 3 combine procedures from Section 8.5 for third order elements for plane stress problems and the previous section 8.9 where second order elements are generated for axisymmetric problems.

8.10.1 Integration of $r\,(f_r\, \phi_r + f_z\, \phi_z)$

The computations are identical to Section 8.9.1 for second order elements. The only difference is the interpolation matrix $\mathbf{M} \in \mathbb{M}^{7\times 10}$, which has to interpolate from the 10 nodes to the 7 Gauss points. See equation (67) on page 184. The contributions in (112) for each element stiffness matrix can be determined. The matrix $\mathbf{M}^T \mathbf{R} \mathbf{W} \mathbf{M}$ is a 10×10 matrix, independent on the triangle T.

8.10.2 Integration of the terms involving derivatives of ϕ_z and ϕ_z

The algorithm is extremely similar to Section 8.9.2 for second order elements, but the matrices $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ are of size 7×10. This leads to the matrices $\mathbf{G}_x$ and $\mathbf{G}_y \in \mathbb{M}^{7\times 10}$ to evaluate the partial derivatives at the nodes, using the values of the function at the nodes. The resulting matrices $\mathbf{A}_{r,z}$ and similar are of size 10×10, leading to the element stiffness matrix $\mathbf{A} \in \mathbb{M}^{20\times 20}$.

8.10.3 The boundary integral

The algorithm is extremely similar to Section 8.9.3 for second order elements. The effect of the boundary integral on the global stiffness matrix and the vector is very similar to the approach shown at the end of Section 6.6.7.

9 Examples, Examples, Examples

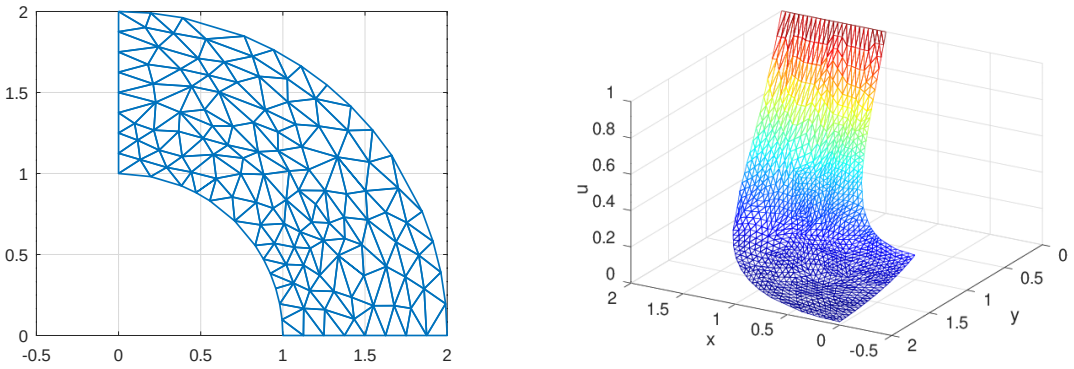
9.1 An elliptic problem with variable coefficients

The elliptic BVP in Section 5.5 is

$$\begin{aligned} -\nabla\left((1+x^2)\nabla u(x,y)\right) &= -4\,(1+x^2)\,\exp(-2\,y) && \text{for} \quad (x,y)\in\Omega \\ \tfrac{\partial\, u(0,y)}{\partial x} &= 0 && \text{for} \quad 1\le y\le 2 \\ u(x,y) &= \exp(-2\,y) && \text{on other sections of the boundary} \end{aligned} \quad .$$

on the domain shown in Figure 102(a). The exact solution is given by $u_e(x,y)=\exp(-2\,y)$. To solve this BVP with FEMoctave use the following steps:

1. Use `CreateMeshTriangle()` to generate a mesh on the rectangle $1\le r\le 2$ and $0\le\varphi\le\pi/2$.
2. With the polar coordinates use
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\,\cos\varphi \\ r\,\sin\varphi \end{pmatrix}$$
to generate the mesh on the section of a ring, visible in Figure 102(a) with the help of an appropriate function `Deform()` and the function `MeshDeform()`.
3. Then use `MeshUpgrade()` to generate a mesh with third order elements.
4. Define the coefficient functions $a(x,y)=1+x^2$ and the right hand side $f(x,y)=-4\,(1+x^2)\,\exp(-2\,y)$ with *Octave* functions.
5. Call the function `BVP2Dsym()` with appropriate arguments to calculate the approximate solution $u(x,y)$.
6. Use `FEMtrimesh()` to display the solution visible in Figure 102(b) and then use `FEMIntegrate()` to determine the L_2–error
$$\left(\iint_\Omega |u(x,y)-u_{exact}(x,y)|^2\,dA\right)^{1/2} \approx 3.3\cdot 10^{-6}\,.$$



(a) the mesh (b) the solution

Figure 102: The mesh and the solution of an elliptic problem with variable coefficients

DeformVariableCoeff.m

```
clear *
h = 0.1
function xy_new = Deform(xy)
  xy_new = [xy(:,1).*cos(xy(:,2)), xy(:,1).*sin(xy(:,2))];
endfunction

function u = f_u_exact(xy)
  u = exp(-2*xy(:,2));
endfunction

function u = f_DDu_exact(xy)
  u = -4*(1+xy(:,1).^2).*exp(-2*xy(:,2));
endfunction

function a = f_a(xy)
  a = 1 + xy(:,1).^2;
endfunction

FEMmesh = CreateMeshTriangle('Test',[1,0,-1;2,0,-1;2,pi/2,-2;1,pi/2,-1],h^2);
FEMmesh = MeshDeform(FEMmesh,'Deform');
figure(1);   FEMtrimesh(FEMmesh)
FEMmesh = MeshUpgrade(FEMmesh,'cubic');
u = BVP2Dsym(FEMmesh,'f_a',0,'f_DDu_exact','f_u_exact',0,0);
figure(2); FEMtrimesh(FEMmesh,u)
           xlabel('x'); ylabel('y'); zlabel('u');  view([-150,30])
u_exact = f_u_exact(FEMmesh.nodes);
L2Error =  sqrt(FEMIntegrate(FEMmesh,(u-u_exact).^2))
-->
L2Error = 3.3205e-06
```

9.2 An animated wave

With a narrow Gauss bell surface around $(x, y) \approx (1, 0)$ as initial value and zero initial velocity observe the waves traveling away from the initial location and the different types of reflections at the boundaries. Figure 103 shows the final status.

WaveAnimation.m

```
if 0 %% linear elements
  FEMmesh = CreateMeshRect(linspace(0,pi,101),linspace(-pi,pi,101),-1,-2,-2,-2);
else %% quadratic elements
  FEMmesh = CreateMeshRect(linspace(0,pi,51),linspace(-pi,pi,51),-1,-2,-2,-2);
  FEMmesh = MeshUpgrade(FEMmesh);
endif
x = FEMmesh.nodes(:,1); y = FEMmesh.nodes(:,2);

m=1; alpha=0.0; a=1; b0=0; bx=0; by=0; f=0; gD=0; gN1=0; gN2=0;
t0=0; tend=3 ; steps = [150,10];

u0 = exp(-25*((x-1).^2+(y-0).^2));
v0 = zeros(length(FEMmesh.nodes),1);
[u_dyn,t] = I2BVP2D(FEMmesh,m,alpha,a,b0,bx,by,f,gD,gN1,gN2,u0,v0,t0,tend,steps);

figure(1) % show animation
for t_ii = 1:length(t)
  FEMtrimesh(FEMmesh,u_dyn(:,t_ii))
  axis([0 pi -pi pi -0.2 0.4]);  xlabel('x'); ylabel('y')
  drawnow();
endfor
```

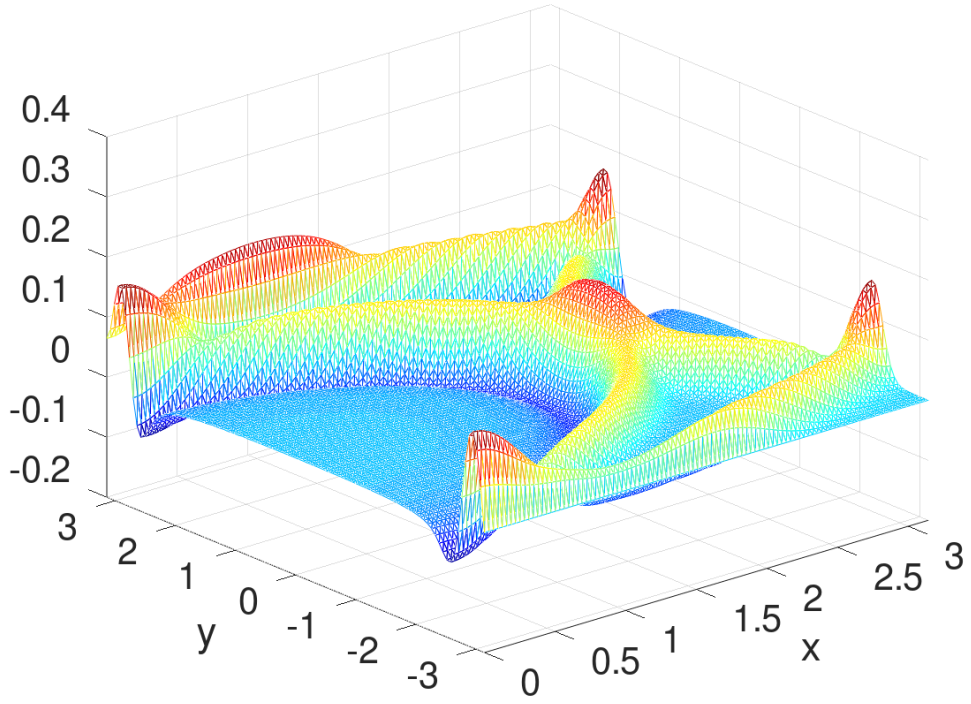


Figure 103: Traveling waves on a rectangle

9.3 An elliptic problem with radial symmetry, superconvergence

The Bessel function

$$u(x,y) = f(x,y) = J_0(\sqrt{x^2+y^2})$$

is an exact solution of the BVP

$$\begin{aligned} -\Delta u + u &= 2\,f && \text{for} \quad 0 < x, y < 1 \\ u &= f && \text{for } (1,y) \text{ and } (x,1) \\ \frac{\partial u}{\partial n} &= 0 && \text{for } (0,y) \text{ and } (x,0) \end{aligned} \quad .$$

A solution is shown in Figure 104. This BVP is solved by two slightly different approaches, and then the difference to the known exact solution is displayed in Figure 105. In both cases first a mesh with linear elements is generated, then upgraded to a mesh with quadratic elements, using `MeshUpgrade()`. Then a mesh with identical nodes and DOF with linear elements is generated by `MeshQuad2Linear()`.

1. Use a uniform mesh generated by `CreateMeshRect`, leading to 400 degrees of freedom. The result in Figure 105(a) shows the effect of super-convergence. Caused by the extremely regular structure of the grid points the differences are smaller than can reasonably be expected.

2. Use a non-uniform mesh generated by `CreateMeshTriangle`, leading to 432 degrees of freedom. Thus one expects to obtain similar accuracy. The result in Figure 105(b) confirms this.

```
N = 10;  Triangle = 1
if Triangle
  FEMmesh = CreateMeshTriangle('test1',[0 0 -2;1 0 -1; 1 1 -1; 0 1 -2],0.75/N^2);
  FEMmesh = MeshUpgrade(FEMmesh);
  FEMmesh1 = MeshQuad2Linear(FEMmesh);
  nDOFTri = [FEMmesh.nDOF, FEMmesh1.nDOF]
else
```

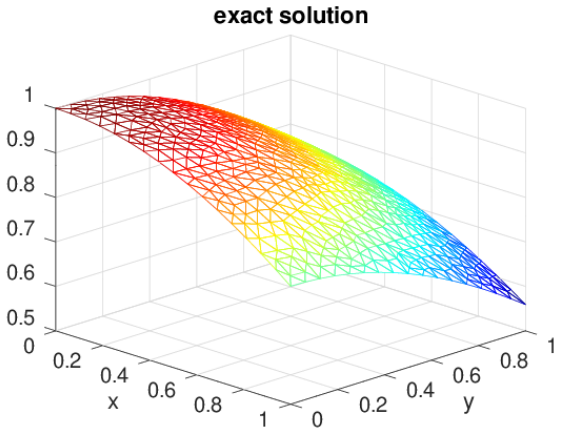
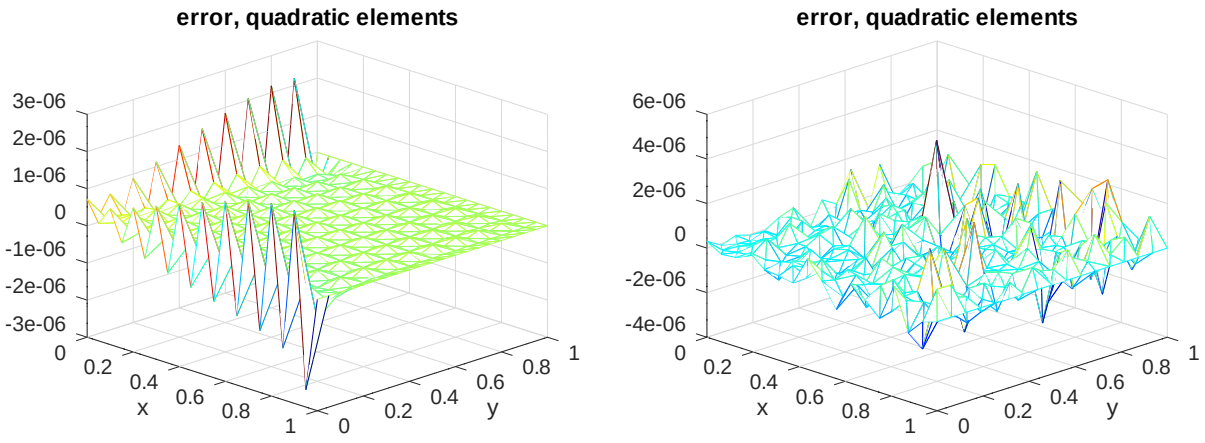


Figure 104: The radial Bessel function as solution of a BVP

```
  FEMmesh = CreateMeshRect(linspace(0,1,N+1),linspace(0,1,N+1),-2,-1,-2,-1);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
  FEMmesh1 = MeshQuad2Linear(FEMmesh);
  nDOFRect = [FEMmesh.nDOF, FEMmesh1.nDOF]
endif
```



(a) uniform grid

(b) nonumiform grid

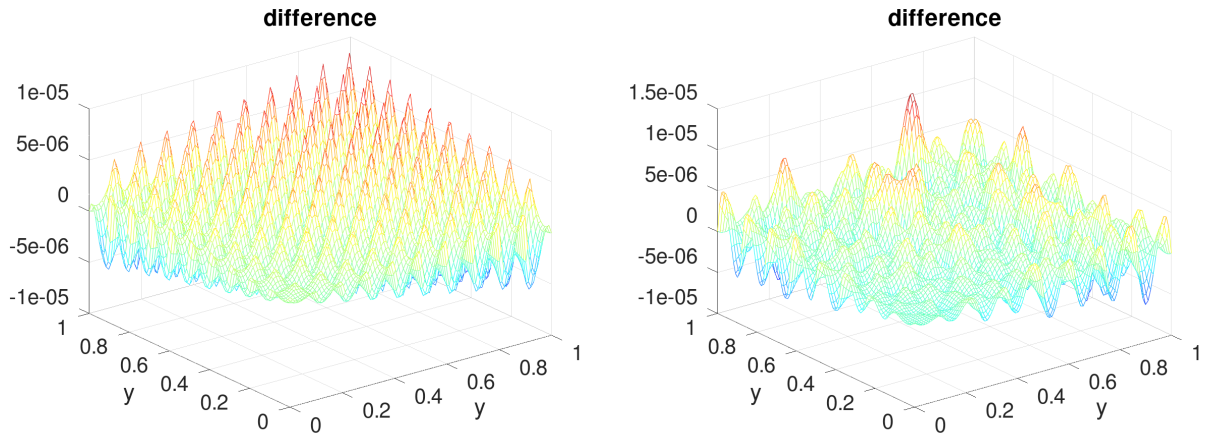
Figure 105: Difference to the exact solution of a BVP

To generate Figure 106 the command `FEMgriddata()` is used to evaluate the functions on a much finer grid (not recomputing, just evaluation) and then display the difference between the approximate and exact solution. Observe that the error is considerably larger, compared to evaluation at the nodes only. This illustrates that the effect of superconvergence does not provide additional accuracy one can reliably count on.

The gradient of this solution u can be determined using $\frac{\partial}{\partial r} J_0(r) = -J_1(r)$ and

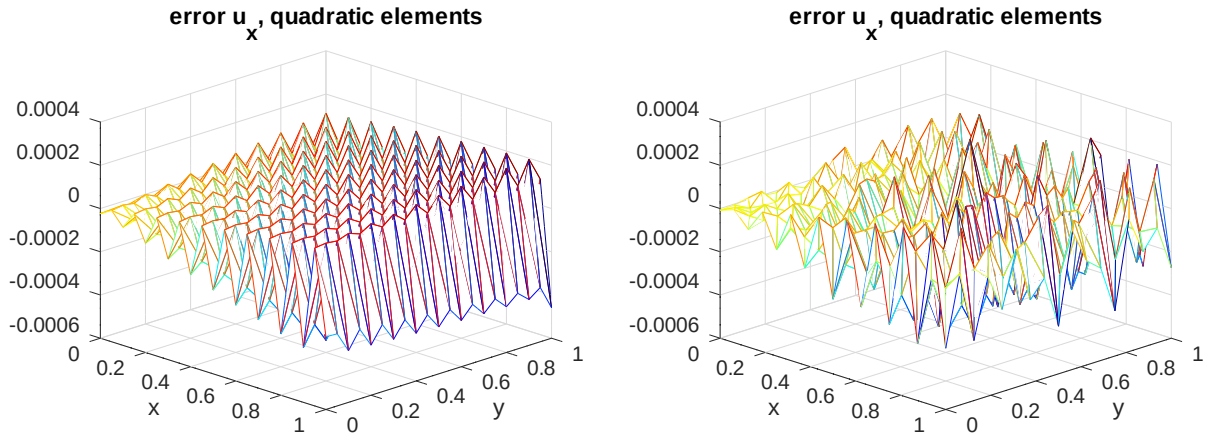
$$\begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{\partial u}{\partial r} + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{\partial u}{\partial \phi} = -\begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} J_1(r)\,.$$

Using the above FEM results compare the true partial derivative $\frac{\partial u}{\partial x}$ with the one obtained by FEM with second order elements. Find the result in Figure 107. Observe the structure of the difference for the uniform mesh.



(a) uniform grid (b) nonumiform grid

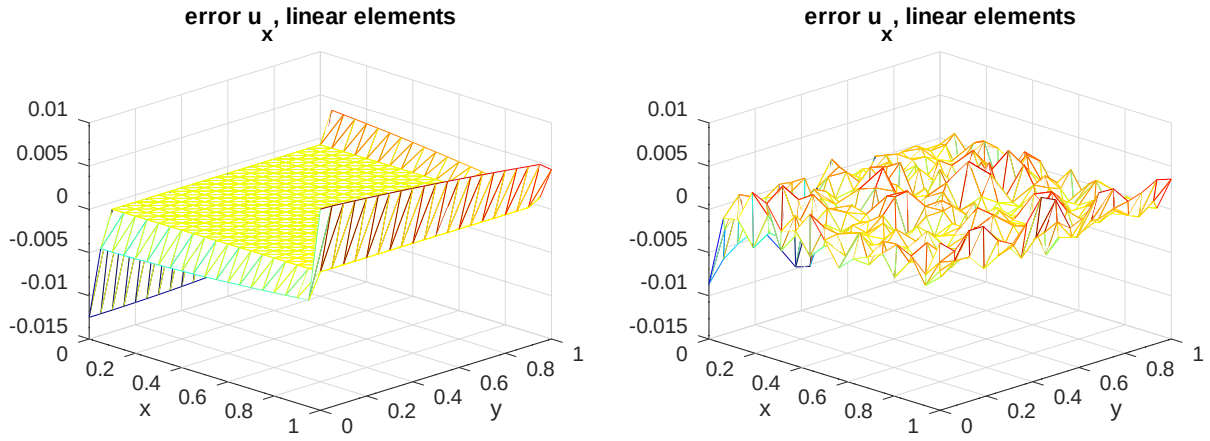
Figure 106: Difference to the exact solution of a BVP, using quadratic elements and interpolation to a finer grid.



(a) uniform grid (b) nonumiform grid

Figure 107: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using second order elements

The above can be repeated using first order elements, leading to Figure 108. The size of the elements was set such that the same number of degrees of freedom are used. Observe that superconvergence strikes again. In this case I have a solid argument for the structural difference along the border.



(a) uniform grid (b) nonumiform grid

Figure 108: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using first order elements

Find more information on superconvergence in [Zien13, §15.2] or a short demo in [Stah08, §6.8.2].

9.4 An example with limited regularity

Let $\Omega \in \mathbb{R}^2$ be the unit square $-1 < x, y < 1$, with the fourth quadrant ($x > 0$, $y < 0$) cut out. For some of the calculations identify $(x, y) \in \mathbb{R}^2$ with $z = x + i\,y \in \mathbb{C}$. Examine the functions

$$\begin{aligned} w(z) &= z^{2/3} = \left(r\,e^{i\phi}\right)^{2/3} = r^{2/3}\,e^{i\phi\,2/3} = r^{2/3}\left(\cos(\phi\,2/3) + i\,\sin(\phi\,2/3)\right) \\ u(z) &= r^{2/3}\,\sin(\phi\,2/3) \\ u(x,y) &= (x^2+y^2)^{1/3}\,\sin(\frac{2}{3}\,\text{atan2}(y,x))\;. \end{aligned}$$

This function satisfies $-\Delta u = 0$ and $u(t,0) = u(0,-t) = 0$ for $t > 0$. Since $\frac{\partial}{\partial r}\,u = \frac{2}{3}\,r^{-1/3}\,\sin(\frac{2}{3}\,\phi)$ and $\frac{\partial}{\partial \phi}\,u = \frac{2}{3}\,r^{2/3}\,\cos(\frac{2}{3}\,\phi)$ the partial derivatives of this function have a singularity at the origin. Compute

$$\begin{aligned} \|\nabla u\|^2 &= |\frac{\partial\,u}{\partial r}|^2 + |\frac{1}{r}\,\frac{\partial\,u}{\partial \phi}|^2 = \frac{4}{9}\,r^{-2/3} + \frac{4}{9}\,\frac{1}{r^2}\,\cos^2(\frac{2}{3}\,\phi) \\ \iint\limits_{\Omega} \|\nabla u\|^2\,dA &= \frac{4}{9}\int_0^1 \left(\int_0^{3\pi/2} r^{-2/3} + r^{-2}\,\cos^2(\frac{2}{3}\,\phi)\,d\phi\right) r\,dr \\ &= \frac{4}{9}\int_0^1 \left(\frac{3\,\pi}{2}\,r^{-2/3} + r^{-2}\,\frac{3\,\pi}{4}\right) r\,dr = \frac{2\,\pi}{3}\int_0^1 r^{1/3}\,dr + \frac{\pi}{3}\int_0^1 \frac{1}{r}\,dr = \infty \end{aligned}$$

to observe that the gradient is not bounded in the L_2 sense. Thus the standard error estimates based on Céa's Lemma do not apply. Expect approximation and convergence problems close to the origin. This is confirmed by the code below and the resulting Figure 109. This example illustrates that non-convex domains with sharp corners might cause convergence problems.

SingularDisc.m

```
x_p = [0;1;1;-1;-1;0]; y_p = [0;0;1;1;-1;-1];

FEMmesh = CreateMeshTriangle("circle34",[x_p,y_p,-ones(size(x_p))], 0.01);
FEMmesh = MeshUpgrade(FEMmesh);

function res = gD(xy)
```

```
  phi = mod(atan2(xy(:,2),xy(:,1)),2*pi);
  res = (xy(:,1).^2+ xy(:,2).^2).^(1/3).*sin(2/3*phi);
endfunction

u = BVP2Dsym(FEMmesh,1,0,0,'gD',0,0);
figure(1); FEMtrimesh(FEMmesh,u);
           xlabel("x"); ylabel("y"); title('FEM solution'); view([30,30])

u_exact = gD(FEMmesh.nodes);
figure(2); FEMtrimesh(FEMmesh,-u+u_exact);
           xlabel("x"); ylabel("y"); title('Error of FEM solution'); view([30,30])
```

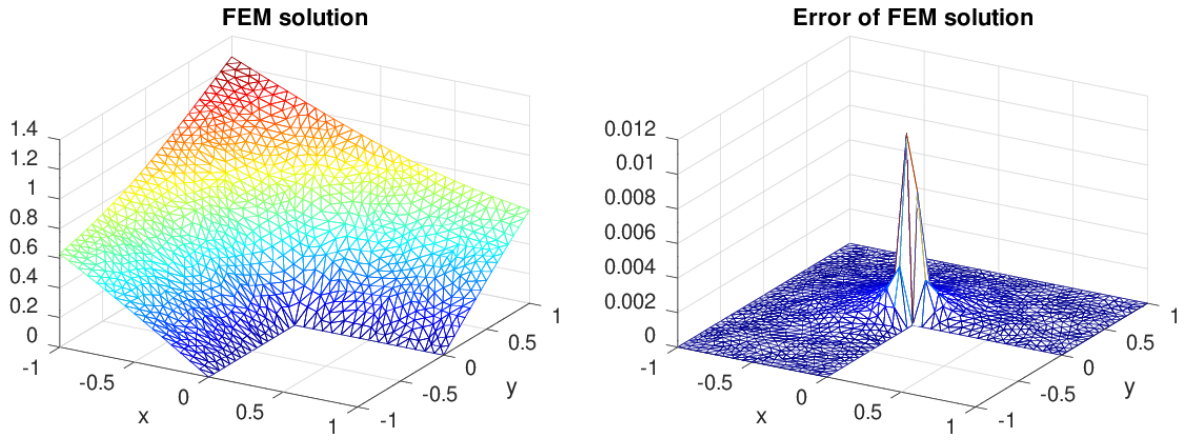


Figure 109: A solution with singular partial derivatives at the origin

The gradient in Cartesian coordinates can be determined by

$$\begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{\partial u}{\partial r} + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{\partial u}{\partial \phi}$$
$$= \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{2}{3} r^{-1/3} \sin(\phi\, 2/3) + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{2}{3} r^{+2/3} \cos(\phi\, 2/3)$$

and then visualized, leading to Figure 110. It is clearly visible that the FEM solution is not accurate where the gradient has a singularity.

```
[ux,uy] = FEMEvaluateGradient(FEMmesh,u);
figure(3); FEMtrimesh(FEMmesh,ux);
           xlabel("x"); ylabel("y"); title('FEM solution, u_x'); view([30,30])

figure(4); FEMtrimesh(FEMmesh,uy);
           xlabel("x"); ylabel("y"); title('FEM solution, u_y'); view([30,30])

figure(5); FEMtrimesh(FEMmesh,sqrt(ux.^2+uy.^2));
           xlabel("x"); ylabel("y"); title('FEM solution, norm of gradient');
           view([30,30])
```

Singularities can show up in mechanical problems, e.g. for the washer fastener example in Section 9.48.

9.5 Accuracy of Hermite elements

In this section the accuracy of solutions based on Hermite elements is illustrated. Based on the doubtful implementation of Dirichlet boundary conditions a rather poor accuracy should not be surprising, see Section 6.7.7 on page 199.

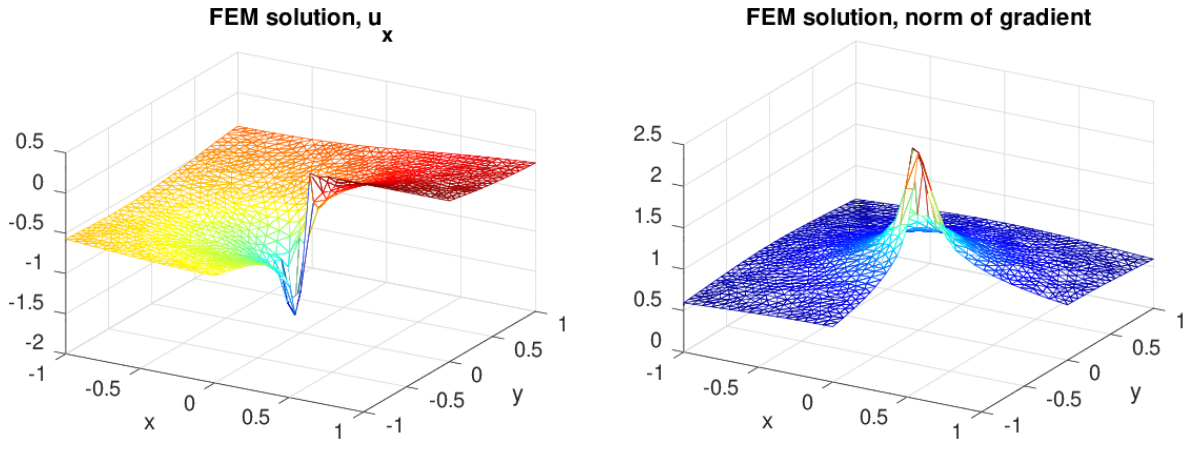


Figure 110: A solution with singular partial derivatives, graphs of $\frac{\partial u}{\partial x}$ and $\|\nabla u\|$

9.5.1 A first Hermite example

Examine the solution of the BVP

$$\begin{aligned} \Delta u &= x^2 && \text{for } 0 < x, y < 1 \\ u(x,y) &= 1 && \text{on boundary } (x,0) \text{ and } (0,y) \\ \tfrac{\partial}{\partial n} u(x,y) &= 0 && \text{on boundary } (x,1) \text{ and } (1,y) \end{aligned} \tag{114}$$

- To solve (114):
  - define the coefficients and the function $f(x,y) = x^2$.
  - generate a (rather coarse) mesh for the Hermite elements.
  - generate a fine mesh for the reference solution u_{ref}. Use the command `MeshUpgrade()` twice.
  - with `BVP2D()` determine the Hermite solution u_{Herm} and the refrence solution u_{ref}.
  - With `FEMgriddata()` evaluate the solutions in a small corner $0 \le x \le 0.2$ and $0 \le y \le 0.1$.
  - Display the solutions with `FEMtrimesh()` and `mesh()`, leading to Figure 111. The RMS (Root Mean Squared) value of $\approx 1.1 \cdot 10^{-4}$ for the difference $u_{Herm} - u_{ref}$ shows that the solution is not too bad.

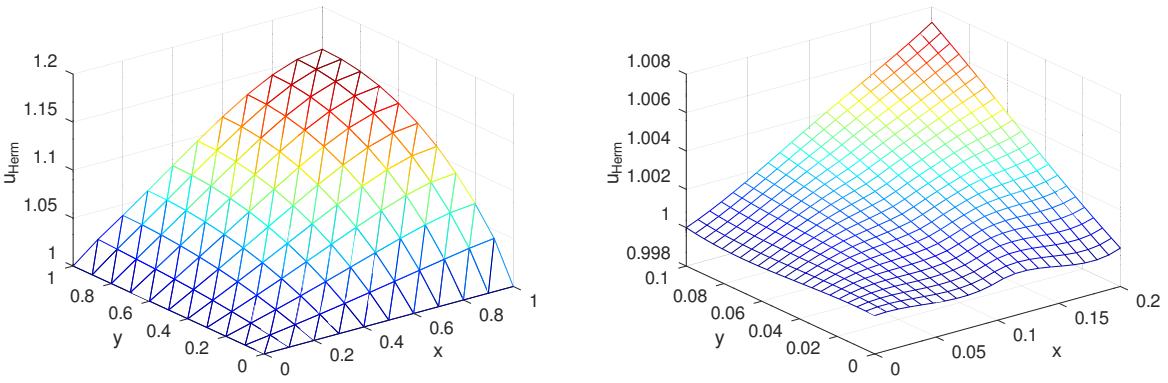
HermiteAccuracy_A.m

```
N = 11;
Mesh = CreateMeshRect(linspace(0,1,N),linspace(0,1,N),-1,-2,-1,-2);
MeshHermite = MeshUpgrade(Mesh,'Hermite');
MeshCubic = MeshUpgrade(Mesh,'cubic');
MeshCubic = MeshCubic2Linear(MeshCubic);
MeshCubic = MeshUpgrade(MeshCubic,'cubic');% a fine mesh with cubic elements

a = 1; b0 = 0; bx = 0; by = 0; gD = 1; gN1 = 0; gN2 = 0;
function res = f(xy,dummy)
   res = 1*xy(:,1).^2;
endfunction

u_ref  = BVP2D(MeshCubic,a,b0,bx,by,"f",gD,gN1,gN2);
u_Herm = BVP2D(MeshHermite,a,b0,bx,by,"f",gD,gN1,gN2);
figure(1); FEMtrimesh(MeshHermite,u_Herm);
           xlabel('x'); ylabel('y'); zlabel('u_{Herm}')
%% use griddata on small part of the domain
[xx,yy] = meshgrid(linspace(0,0.2,2*N),linspace(0,0.1,2*N));
u_ref_g  = FEMgriddata(MeshCubic,u_ref,xx,yy);
u_Herm_g = FEMgriddata(MeshHermite,u_Herm,xx,yy);
```

```
figure(2); mesh(xx,yy,u_Herm_g);
           xlabel('x'); ylabel('y'); zlabel('u_{Herm}')
Difference_RMS = sqrt(mean((u_Herm_g(:)-u_ref_g(:)).^2))
-->
Difference_RMS = 1.1446e-04
```



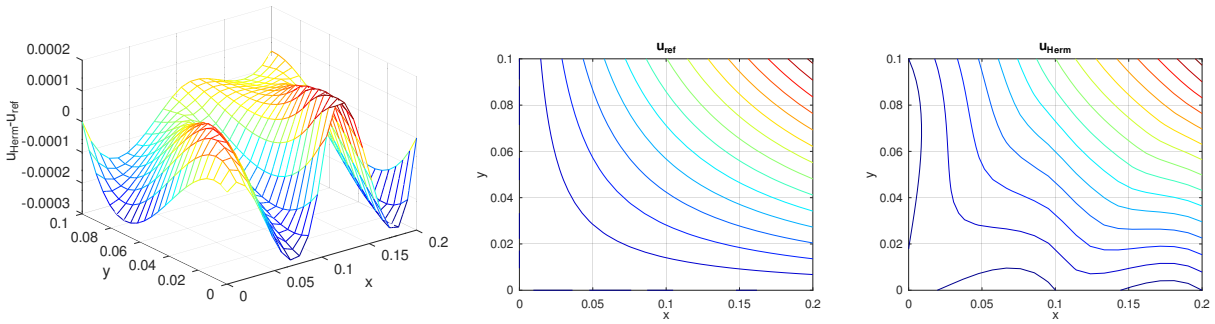
(a) the full domain (b) a small corner

Figure 111: The solution of the BVP to examine the accuracy of Hermite elements

- In Figure 112 the small corner is examined more carfully.
  - The difference $u_{Herm} - u_{ref}$ diplays a cler structural problem close to the corner $(x, y) \approx (0, 0)$.
  - The contour of the reference solution u_{ref} shows the expected behavior, while the contour of the Hermite solution u_{Herm} does certainly not coincide with the exact solution.

HermiteAccuracy_A.m

```
figure(21); mesh(xx,yy,u_Herm_g-u_ref_g);
            xlabel('x'); ylabel('y'); zlabel('u_{Herm}-u_{ref}')
levels = 1 + linspace(-0.001,0.01,23);
figure(22); contour(xx,yy,u_ref_g,levels);
            xlabel('x'); ylabel('y'); title('u_{ref}')
figure(23); contour(xx,yy,u_Herm_g,levels);
            xlabel('x'); ylabel('y'); title('u_{Herm}')
```



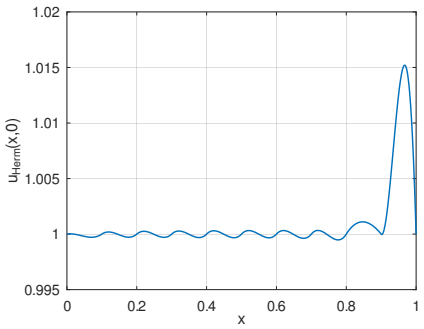
(a) difference $u_{Herm} - u_{ref}$ (b) contour of u_{ref} (c) contour of u_{Herm}

Figure 112: The Hermite solution of the BVP in a corner

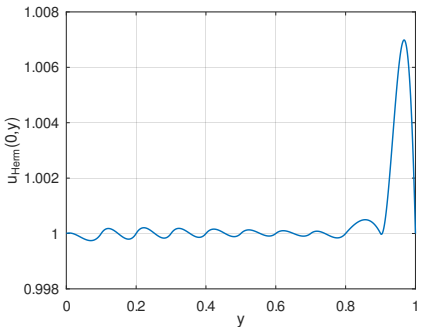
- Along the two edges $(x, 0)$ and $(0, y)$ use `FEMgriddata()` to evaluate the function and the partial derivatives on a very fine grid.
  - The Dirichlet condition $u(x, 0) = u(0, y) = 1$ should lead to constant values in Figure 113.
  - The Hermite element assures continuity of the first order partial derivatives at the nodes. Along the coordinate axis this leads to continuous derivatives. In Figure 113 even zooming in closely will not show any jumps. The derivatives display corners at the nodes at x, y multiples of 0.1, i.e. the second order derivatives are not constant.

HermiteAccuracy_A.m

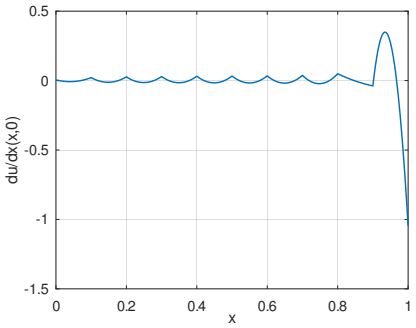
```
%%  along the two edges
N1 = 1001;
x = linspace(0,1,N1); y = zeros(size(x));
[u_Herm_gx,ux,uy] = FEMgriddata(MeshHermite,u_Herm,x,y);
figure(31); plot(x,u_Herm_gx); xlabel('x'); ylabel('u_{Herm}(x,0)')
figure(32); plot(x,ux); xlabel('x'); ylabel('du/dx(x,0)')
[u_Herm_gy,ux,uy] = FEMgriddata(MeshHermite,u_Herm,y,x);
figure(33); plot(x,u_Herm_gy); xlabel('y'); ylabel('u_{Herm}(0,y)')
figure(34); plot(x,uy); xlabel('y'); ylabel('du/dy(0,y)')
```



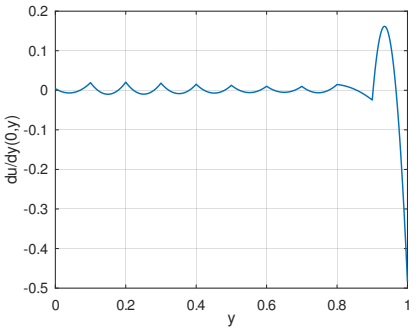
(a) u along x–axis



(b) u along y–axis



(c) $\frac{\partial u}{\partial x}$ along x–axis



(d) $\frac{\partial u}{\partial y}$ along y–axis

Figure 113: The Hermite solution of the BVP along the coordinate axis

9.5.2 A second Hermite example

The exact solution of the BVP

$$\begin{aligned} \Delta u &= 4\,(x^2+y^2-1)\,e^{-x^2-y^2} && \text{on the domain } x^2+y^2 < R^2 \\ u(x,y) &= e^{-x^2-y^2} && \text{on the boundary } x^2+y^2 = R^2 \end{aligned} \tag{115}$$

is given by

$$u_{exact}(x,y) = e^{-x^2-y^2}\ .$$

In the code `HermiteAccuracy_B.m` shown below

- The coefficients and functions are defined and a first mesh with linear elements is generated.
- The linear mesh is upgraded to a mesh with cubic elements and the resulting BVP is solved. The time required to perform these two steps is measured.
- The linear mesh is upgraded to a mesh with Hermite elements and the resulting BVP is solved. The time required to perform these two steps is measured.
- The solution of with the Hermite elements on the coarse mesh is shown on the left in Figure 114.

Find some of the results in Table 22. Obviously cubic elements require considerably less computation time[39] and lead to better accuracy.

element type	computation time	average difference
cubic	0.16 sec	$2.8 \cdot 10^{-6}$
Hermite	1.08 sec	$7.0 \cdot 10^{-4}$

Table 22: Computation time and error for generating the Bell surface

HermiteAccuracy_B.m

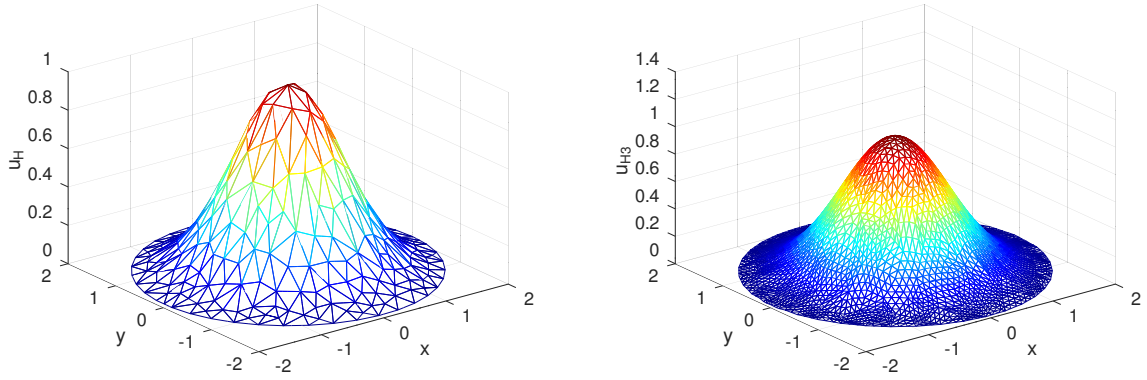
```
%% solution is a bell surface on a full circle
N = 20; R = 2;
angles = linspace(0,2*pi,2*N)'; angles = angles(1:end-1);
xy = [R*cos(angles),R*sin(angles),-ones(size(angles))];
Mesh = CreateMeshTriangle('circ',xy,4*R^2/N^2);
x = Mesh.nodes(:,1); y = Mesh.nodes(:,2);
a = 1; b0 = 0; bx = 0; by = 0; gN1 = 0; gN2 = 0;

function res = gD(xy)
  res = exp(-sum(xy.^2,2));
endfunction
function res = f(xy,d)
  x = xy(:,1); y = xy(:,2);
  res = -(4*x.^2+4*y.^2-4).*exp(-sum(xy.^2,2));
endfunction

tic();
  Mesh3 = MeshUpgrade(Mesh,'cubic');
  u3 = BVP2D(Mesh3,a,b0,bx,by,'f','gD', gN1,gN2);
Time_cubic = toc()
figure(1); FEMtrimesh(Mesh3,u3); xlabel('x'); ylabel('y'); zlabel('u_3')
x3 = Mesh3.nodes(:,1); y3 = Mesh3.nodes(:,2);
Difference_u3 = mean(abs(u3-gD([x3,y3])))
%% Hermite elements
tic();
  MeshH = MeshUpgrade(Mesh,'Hermite');
  uH = BVP2D(MeshH,a,b0,bx,by,'f','gD', gN1,gN2);
Time_Hermite = toc()
figure(2); FEMtrimesh(MeshH,uH);
           xlabel('x'); ylabel('y'); zlabel('u_{H}')
tic();
uHI = FEMgriddata(MeshH,uH,x,y);
Time_HermInterpol = toc()
figure(3); FEMtrimesh(Mesh,uHI);
           xlabel('x'); ylabel('y'); zlabel('u_{HI}')
Difference_uHI = mean(abs(uHI-gD([x,y])))
-->
```

[39] This is not really a fair comparison, the cubic elements use compiled code, while the Hermite elements use an Octave script. A run with a scipt file for cubic elements lead to a computation time of ≈ 0.8 sec.

```
Time_cubic        = 0.1579
Difference_u3     = 2.7733e-06
Time_Hermite      = 1.0757
Time_HermInterpol = 0.063192
Difference_uHI    = 7.0373e-04
```



(a) The Hermite solution on the coarse mesh

(b) The Hermite solution on the fine mesh

Figure 114: A Gauss Bell surface as solution of a BVP

The above solution generated by Hermite polynomials and then displayed, leading to the right part of Figure 114. Then the difference to the true solution is evaluated and displayed, leading to Figure 115. The errors are mostly located along the boundary of the circular domain. This should not be a surprise. See Section 6.7.7 on page 199.

HermiteAccuracy_B.m

```
%% interpolation for Hermite
tic();
uH3 = FEMgriddata(MeshH,uH,x3,y3);
Time_HermInterpol3 = toc()
figure(4); FEMtrimesh(Mesh3,uH3);
           xlabel('x'); ylabel('y'); zlabel('u_{H3}')
Difference_uH3 = mean(abs(uH3-gD([x3,y3])))
figure(5); FEMtrimesh(Mesh3,uH3-u3);
           xlabel('x'); ylabel('y'); zlabel('difference')
-->
Time_HermInterpol3 = 0.1688
Difference_uH3     = 8.6542e-04
```

9.6 Solving a Liouville equation

On a semidisk of radius $R = 1$ solve the nonlinear BVP

$$\begin{aligned} -\Delta u &= e^{c\,u} && \text{for } x^2 + y^2 < R^2 \text{ and } y > 0 \\ u(x,y) &= gD && \text{for } (x,y) = (R\cos\phi\,,\, R\sin\phi) \text{ with } 0 \le \phi \le \pi \\ \tfrac{\partial}{\partial y}\,u(x,0) &= 0 && \text{for } -R \le x \le +R \end{aligned}$$

The solution will depend on the radius $r = \sqrt{x^2 + y^2}$ only. With polar coordinates the above simplifies to a 1D BVP

$$\begin{aligned} \frac{1}{r}\,\frac{\partial}{\partial r}\,(r\,\frac{\partial\,u(r)}{\partial r}) &= e^{c\,u(r)} && \text{for } 0 < r < R \\ \frac{\partial}{\partial r}\,(r\,\frac{\partial\,u(r)}{\partial r}) &= r\,e^{c\,u(r)} && \text{for } 0 < r < R \end{aligned}$$

with the boundary conditions $u'(0) = u(R) = 0$. This is an example of a Liouville equation. The code in `Liouville.m` generates numerical solutions of these two boundary value problems. Find the results in Figures 116

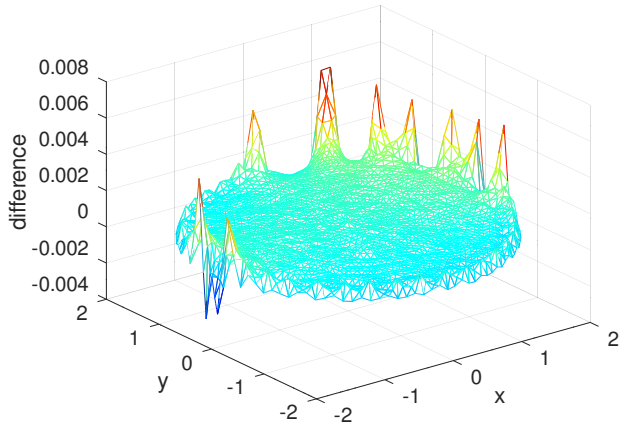


Figure 115: A Gauss Bell surface, the difference to the exact solution

and 117. Observe that this is a nonlinear problem and the existence of a solution is not guaranteed. Modifying the values of R, c and gD in the code might no lead to a solution!

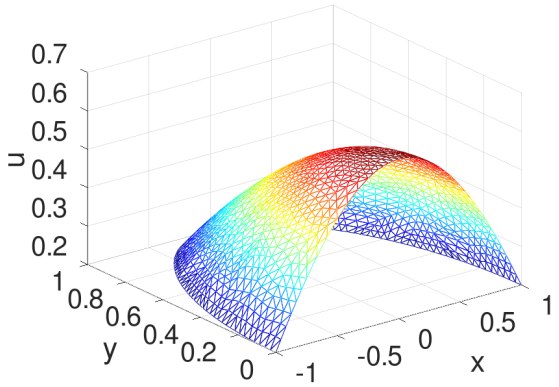


Figure 116: The solution of a Liouville equation in $\mathbb{R}^2$

A substantial contribution to the difference of the 1D and 2D solutions is generated at $r \approx R$, visible on the right in Figure 117. It is caused by the discretization of the domain, i.e. the circle is replaced by a polygon in $\mathbb{R}^2$. This is explained in Section 5.8 on page 134.

Liouville.m

```
R = 1; N = 36; alpha = linspace(0,pi,N)';
xy = [R*cos(alpha),R*sin(alpha),[-ones(N-1,1);-2]];
Mesh = CreateMeshTriangle('circle',xy,0.01);
%Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

a = 1; b0 = 0; bx = 0; by = 0; u0 = 0; c = 1.0; gD = +0.2;
f = {@(xy,u)exp(c*u);   @(xy,u)c*exp(c*u)};
disp('solving the 2D problem')
u = BVP2DNL(Mesh,a,b0,bx,by,f,gD,0,0,u0,'tol',1e-10,'Display','iter');
figure(1); FEMtrimesh(Mesh,u); xlabel('x'); ylabel('y'); zlabel('u')

Interval = linspace(0,R,201);
f_r = {@(r,u)r.*exp(c*u), @(r,u)c*r.*exp(c*u)};
disp('solving the 1D problem')
```

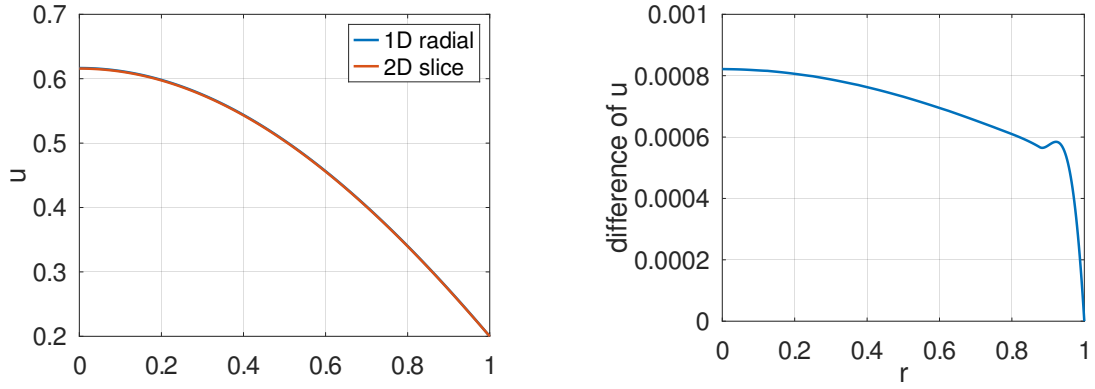


Figure 117: The solution of a Liouville equation in $\mathbb{R}$ and the difference to the solution in $\mathbb{R}^2$

```
[r,u1] = BVP1DNL(Interval,@(r)r,0,0,1,f_r,[0,0],gD,u0,'tol',1e-10,'Display','iter');
u2d = FEMgriddata(Mesh,u,r,zeros(size(r)));
figure(2); plot(r,u1,r,u2d); xlabel('r'); ylabel('u')
           legend('1D radial','2D slice')
figure(3); plot(r,u1-u2d); xlabel('r'); ylabel('difference of u')
-->
solving the 2D problem
iteration 1, RMS(correction) = 8.356644e-02
iteration 2, RMS(correction) = 1.917466e-03
iteration 3, RMS(correction) = 1.158775e-06
iteration 4, RMS(correction) = 4.310897e-13
solving the 1D problem
iteration 1, RMS(correction) = 1.163513e-01, RMS(phi) = 1.143953e-01
iteration 2, RMS(correction) = 7.939812e-04, RMS(phi) = 7.938795e-04
iteration 3, RMS(correction) = 4.092488e-08, RMS(phi) = 4.092535e-08
iteration 4, RMS(correction) = 3.247351e-14, RMS(phi) = 5.570054e-13
```

The above output confirms the quadratic convergence on Newton's algorithm, i.e. the number of stable digits is (approximately) doubled by each iteration.

9.7 Solving Bratu's equation

There are a few applications for the Bratu equation[40], see e.g. [Mohs14] or [JacoSchm02]. Below find two setups examined: a 1D version on the interval $[0,1]$ and a 2D setup on the square $[0,1]\times[0,1]$.

9.7.1 Solving the Bratu equation on $[0,1]$

The BVP to be examined is

$$\begin{aligned} -u''(x) &= C\,e^{u(x)} \qquad \text{for } 0<x<1 \\ u(0) &= u(1) = 0 \end{aligned} \tag{116}$$

There are known exact solutions for this equation.

- For a given value of C solve for α and find an exact solution, given by

$$u(x) = 2\,\ln\left(\frac{\cosh(\alpha)}{\cosh(\alpha\,(1-2\,x)}\right) \quad \text{where} \quad \cosh(\alpha) = \frac{4\,\alpha}{\sqrt{2\,C}}\,.$$

  Soving the equation for C and α can be done with the numerical solver of your choice. The code below generates the image of all solutions of $\sqrt{2\,C}\,\cos(\alpha) - 4\,\alpha = 0$ on the left in Figure 118.

[40] Also called Liouville–Bratu–Gelfand equation. The similarity to the Liouville equation $-\Delta u = \exp(c\,u)$ in the previous section is obvious.

```
[alpha,C] = meshgrid(linspace(0,5,201),linspace(0,5,201));
z = sqrt(2*C).*cosh(alpha)-4*alpha;
figure(1); contour(alpha,C,z,[0,0],'k'); xlabel('\alpha'); ylabel('C')
```

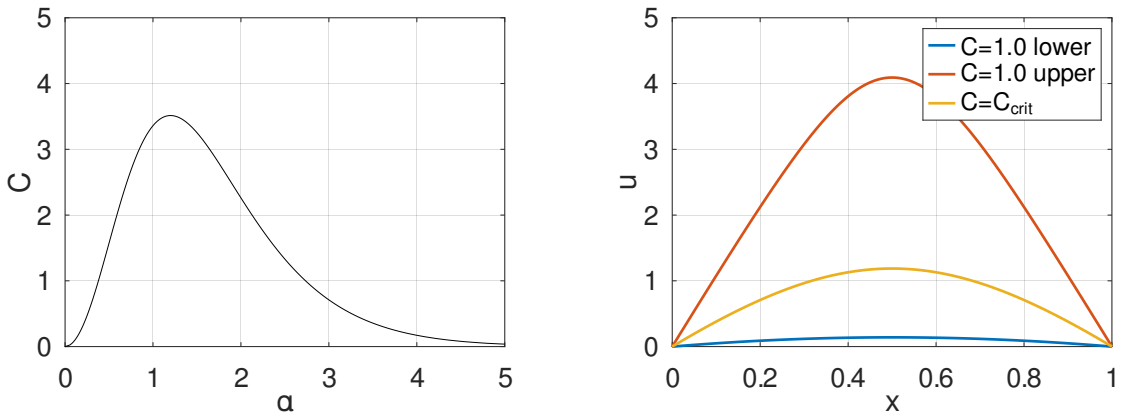


Figure 118: Solutions for the parameters α and C and three solutions

- With Figure 118 conclude that for $C < C_{crit} \approx 3.5138$ two solutions exist:
  - The smaller solution with $\|u\|_\infty = u(\frac{1}{2}) < u_{crit} \approx 1.2277$
  - The larger solution with $\|u\|_\infty > u_{crit}$

  Using the command `BVP1DNL()` these solutions can be determined. The code `Bratu.m` generates the two solutions for $C = 1$ and the only solution for the critical value C_{crit}.

Bratu.m

```
C = 1.0; C_crit = 3.513830719;
N = 201;
Interval = linspace(0,1,N)';
f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
u0 = @(x)0.8*sin(pi*x); %% BVP1DNL, one solution, lower branch
[x,u_low] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = [max(u_low),pwquadinterp(x,u_low,0.5)]

u0 = @(x)4*sin(pi*x); %% BVP1DNL, one solution, upper branch
[x,u_upp] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = [max(u_upp),pwquadinterp(x,u_upp,0.5)]

u0 = @(x)1.2*sin(pi*x);
f = {@(x,u)C_crit*exp(u),@(x,u)C_crit*exp(u)};
[x,u_crit] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter','maxiter',20);
u_max = [max(u_crit),pwquadinterp(x,u_crit,0.5)]
figure(1); plot(x,[u_low,u_upp,u_crit]); xlabel('x'); ylabel('u')
           legend('C=1.0 lower','C=1.0 upper','C=C_{crit}')
```

Observe that the convergence for the critical case C_{crit} is not quadratic, since the solution is not nicely isolated. Thus Newton's algorithm might have problems to converge.

The next goal is to construct upper and lower solutions for (almost) all parameter values.

- Start with $C = 1$, construct the lower solution as above by using the initial function $u_0(x) = 0.8\ \sin(\pi\, x)$. Then increase the values of C by small increments up to C_{crit}, always using the previous solution as starting function for Newton's algorithm for the next solution. At each step store the values of C and $\|u\|_\infty = u(\frac{1}{2})$.

- Repeat the above with initial function $u_0(x) = 4\ \sin(\pi\, x)$ to construct the upper solutions.
- Start with $C = 1$, construct the lower solution as above by using the initial function $u_0(x) = 0.8\ \sin(\pi\, x)$. Then decrease the values of C by small increments down to 0.1 and store the results.
- Restart with $C = 1$, construct the upper solution as above by using the initial function $u_0(x) = 0.8\ \sin(\pi\, x)$. Then decrease the values of C by small increments down to 0.1. Store the results only if the algorithm converged. For small values of C the algorithm will not converge. To fix this smaller steps of C would be required.
- Plot all four results in one graph, visible in Figure 119. The result is very similar to [Mohs14, Figure 1].

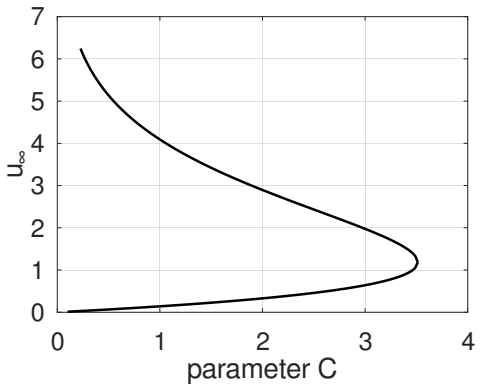


Figure 119: The branch of all solutions to the the Bratu equation

BratuParametric.m

```
C_crit = 3.513830719;
N_C = 101; N = 201;
Interval= linspace(0,1,N)';
%% BVP1DNL, lower branch, up
C_list = linspace(1,C_crit,N_C);
u0 = @(x)0.8*sin(pi*x);
C = 1;
f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
[x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = max(u)
for C = C_list(2:end)
  disp(sprintf('parameter C = %f',C))
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  [x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u,'display','iter','maxiter',20);
  u_max = [u_max,max(u)];
endfor
u_max_low = u_max;

%% BVP1DNL, upper branch, up
u0 = @(x)4*sin(pi*x);
C = 1;
f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
[x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = max(u)
for C = C_list(2:end)
  disp(sprintf('parameter C = %f',C))
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  [x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u,'display','iter','maxiter',20);
  u_max = [u_max,max(u)];
endfor
u_max_upp = u_max;
```

```
%% BVP1DNL, lower branch, down
C_list_down = linspace(1,0.1,N_C);
u0 = @(x)0.8*sin(pi*x);
C = 1;
f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
[x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = max(u)
for C = C_list_down(2:end)
  disp(sprintf('parameter C = %f',C))
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  [x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u,'display','iter','maxiter',20);
  u_max = [u_max,max(u)];
endfor
u_max_low_down = u_max;
u0 = @(x)4*sin(pi*x);
C = 1;
f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
[x,u] = BVP1DNL(Interval,1,0,0,1,f,0,0,u0,'display','iter');
u_max = max(u)
for C = C_list_down(2:end)
  disp(sprintf('parameter C = %f',C))
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  [x,u,inform] = BVP1DNL(Interval,1,0,0,1,f,0,0,u,'display','none','maxiter',20);
  if inform.info ==1
    u_max = [u_max,max(u)];
  else
    u_max = [u_max,NaN];
  endif
endfor
u_max_upp_down = u_max;

%% plot all
figure(5); plot(C_list_down,[u_max_low_down',u_max_upp_down'],'k',
                C_list,     [u_max_low',u_max_upp'],'k');
           xlabel('parameter C'); ylabel('u_\infty')
```

9.7.2 Solving the Bratu equation on $[0,1] \times [0,1] \subset \mathbb{R}^2$

For the domain $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$ one possible version of Bratu's equation is given by

$$\begin{aligned} -\Delta u(x,y) &= C\, e^{u(x,y)} && \text{for } 0 < x, y < 1 \\ u(x,y) &= 0 && \text{on the boundary} \end{aligned} \tag{117}$$

The goal is to obtain results similar to the 1D situation.

Due to symmetry for the solution of (117) there are different options to generate the domain and mesh.

1. Use `CreateMeshRect()` to generate $[0,1] \times [0,1]$ with Dirichlet conditions on the boundary.
2. Use `CreateMeshRect()` to generate $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$ and use Neumann boundary conditions for $x, y = \frac{1}{2}$.
3. Use `CreateMeshTriangle()` to generate $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$ and use Neumann boundary conditions for $x, y = \frac{1}{2}$.
4. Work with the triangle given by the points $(0,0)$, $(\frac{1}{2}, \frac{1}{2})$ and $(0, \frac{1}{2})$. Generate the mesh with `triangle` and apply the correct boundary conditions.

Bratu2D.m

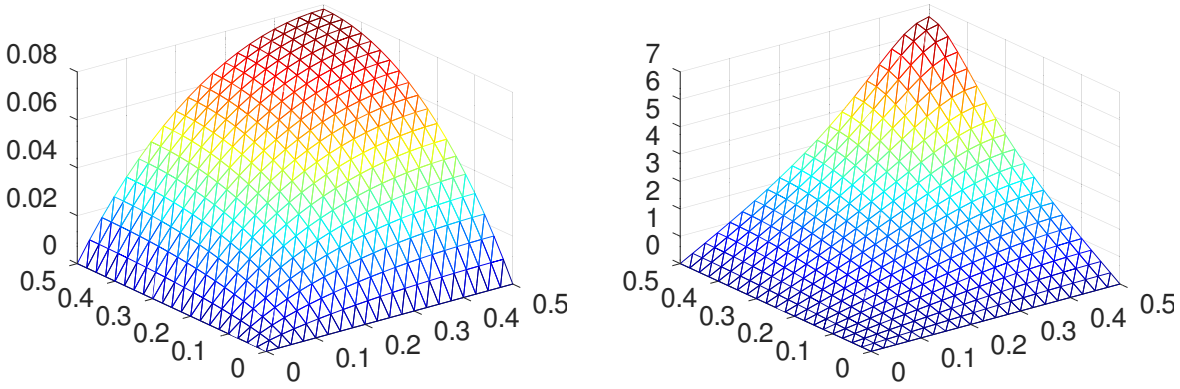
```
MeshCase  = 2;  N = 10;
switch MeshCase
case 1  %% full square
  Mesh = CreateMeshRect(linspace(0,1,2*N+1),linspace(0,1,2*N+1),-1,-1,-1,-1);
case 2  %% quarter of the square
```

```
  Mesh = CreateMeshRect(linspace(0,0.5,N+1),linspace(0,0.5,N+1),-1,-2,-1,-2);
case 3  %% quarter of the square, triangle
  Mesh = CreateMeshTriangle('Shape',[0,0,-1;0.5,0,-2;0.5,0.5,-2;0,0.5,-1],0.2/N^2);
case 4  %% eigth of the square, triangle
  Mesh = CreateMeshTriangle('Shape',[0,0,-2;0.5,0.5,-2;0,0.5,-1],0.2/N^2);
endswitch
Mesh = MeshUpgrade(Mesh,'cubic');
```

The size of the individual elements (triangles) is similar for the four setups. Smaller domains lead to smaller systems of equations to be solved.

The solutions are then evaluated step by step, selectable by the variable `CASE` in the code `Bratu2D.m`.

To determine the lower solution with the parameter $C = 1$ use an initial function $u_0(x, y) = 0.8 \sin(\pi\, x) \sin(\pi\, y)$ and the command `BVP2DNL()`. To obtain the upper solution use $u_0(x, y) = 5.5 \sin(\pi\, x) \sin(\pi\, y)$. This choice was rather delicate and the solution on the right in Figure 120 is clearly not sin-shaped.



(a) the lower solution

(b) the upper solution

Figure 120: Two solution of Bratu's equation for $C = 1$

Bratu2D.m

```
CASE = 1;
switch CASE
case 1  %% BVP2DNL, one solution, lower branch
  C = 1.0;
  u0 = @(xy)0.8*sin(pi*xy(:,1)).*sin(pi*xy(:,2));
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  u = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'display','iter');
  figure(1); FEMtrimesh(Mesh,u)
  u_max = [max(u),FEMgriddata(Mesh,u,0.5,0.5)]

case 2  %% BVP2DNL, one solution, upper branch
  C = 1.0;
  u0 = @(xy)5.5*sin(pi*xy(:,1)).*sin(pi*xy(:,2));
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  u = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'maxiter',20);
  figure(2); FEMtrimesh(Mesh,u)
  u_max = [max(u),FEMgriddata(Mesh,u,0.5,0.5)]
```

To generate the branch of all solutions in the (C, u_∞) plane (similar to Figure 119) the additional problem is to determine the maximal value C_{crit} for the parameter.

- Start with $C = 1$ and generate the lower solution as above, i.e. the results on the left in Figure 120.

- Increase the value of C by step $= 0.2$ and try to solve. If the algorithm in `BVP2DNL()` converges, store the results and apply the next step, i.e. $C \rightarrow C + \text{step}$.
- If the algorithm does not converge fast enough, divide the step by 2 and retry.
- If the step is small enough the maximal value for $C \approx C_{crit}$ is attained. Store and display the results.
- Proceed similarly for the upper branch, but with a smaller initial step size step $= 0.025$, since the convergence is more delicate.
- As final step display both results, leading to Figure 121.

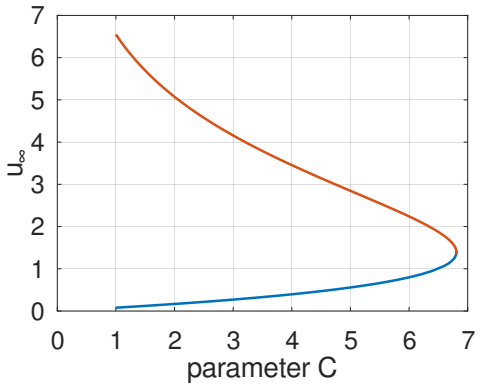


Figure 121: The branch of solutions for Bratu's equation in 2D

Bratu2D.m

```
case 3  %% BVP2DNL, parameter moving up, lower branch
  C = 1.0;  u0 = @(xy)0.8*sin(pi*xy(:,1)).*sin(pi*xy(:,2));
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  u = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'maxiter',20);
  C_list = C;  u_max_list = max(u);
  step = 0.2; half_counter = 0; iterations = 0;
  do
    u0 = u; C = C+step;
    f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
    [u,inform] = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'maxiter',20);
    iterations++;
    if (inform.info == 1)&&(inform.iter<5)
      u_max_list = [u_max_list,max(u)]; C_list = [C_list,C];
      disp(sprintf("%i, C = %f, max(u) = %f, iterations = %i",...
                   iterations,C,u_max_list(end),inform.iter))
      u0 = u;
    else
      C = C-step; step = step/2; half_counter++;
    endif
  until or(iterations>=150,half_counter>15)
  figure(3); plot(C_list,u_max_list,'.-'); xlabel('parameter C'); ylabel('u_\infty')
  C_list3 = C_list; u_max_list3 = u_max_list;
case 4  %% BVP2DNL, parameter moving up, upper branch
  C = 1.0;  u0 = @(xy)5.5*sin(pi*xy(:,1)).*sin(pi*xy(:,2));
  f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
  [u,inform] = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'maxiter',20);
  C_list = C;  u_max_list = max(u);
  disp(sprintf("%i, C=%f, max(u)= %f, iterations = %i",...
               0,C,u_max_list(end),inform.iter))
  step = 0.025; half_counter = 0; iterations = 0;
  do
    u0 = u; C = C+step;
```

```
    f = {@(x,u)C*exp(u),@(x,u)C*exp(u)};
    [u,inform] = BVP2DNL(Mesh,1,0,0,0,f,0,0,0,u0,'maxiter',20);
    iterations++;
    if (inform.info == 1)&&(inform.iter<5)
      u_max_list = [u_max_list,max(u)]; C_list = [C_list,C];
      disp(sprintf("%i, C = %f, max(u) = %f, iterations = %i",...
                   iterations,C,u_max_list(end),inform.iter))
      u0 = u;
    else
      C = C-step; step = step/2; half_counter++;
    endif
  until or((iterations>=600),(half_counter>15))
  figure(4); plot(C_list,u_max_list,'.-'); xlabel('parameter C'); ylabel('u_\infty')
  C_list4 = C_list; u_max_list4 = u_max_list;
case 5
  figure(5); plot(C_list3,u_max_list3,C_list4,u_max_list4);
             xlabel('parameter C'); ylabel('u_\infty'); xlim([0,7])
endswitch
```

9.8 Poiseuille flow through a pipe

The Poiseuille flow[41] of a viscous liquid with velocity $u(x, y)$ through a pipe with cross section $\Omega \subset \mathbb{R}^2$, a pressure gradient $G = -\frac{\partial p}{\partial z}$ and dynamic viscosity μ can be modeled by the BVP

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = \frac{G}{\mu} \text{ in } \Omega \text{ and } u = 0 \text{ on the boundary } \partial\Omega. \tag{118}$$

9.8.1 Poiseuille flow through an annular tube, 1D solution

The situation of an annular tube $R_1 < r = \sqrt{x^2 + y^2} < R_2$ can be examined using polar coordinates. In this case (118) simplifies to the ODE

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\,\frac{\partial\, u(r)}{\partial r}\right) = \frac{G}{\mu} \text{ for } R_1 < r < R_2 \text{ and } u(R_1) = u(R_2) = 0\ .$$

For a full, circular pipe $0 \le r < R_2$ obtain the well known parabolic profile $u(r) = \frac{G}{4\,\mu}\,(R_2^2 - r^2)$. For $0 < R_1 < r < R_2$ it is possible to determine the exact solution of this boundary value problem.

$$u_{exact}(r) = \frac{G}{4\,\mu}\,(R_1^2 - r^2) + \frac{G}{4\,\mu}\,\frac{R_2^2 - R_1^2}{\ln\frac{R_2}{R_1}}\,\ln\frac{r}{R_1}$$

This BVP can be solved using `BVP1D()` too. The code `Annulus.m` determines a numerical approximation and shows the solution and the difference to the exact solution in Figure 122. The total flow through this pipe can be evaluated by

$$\text{Flow } = \iint_\Omega u\ dA = 2\,\pi \int_{r=R_1}^{R_2} u(r)\,r\ dr \approx 0.9711$$

Annulus.m

```
G = 1; mu = 1; R1 = 1; R2 = 2;
Interval = linspace(R1,R2,21)';
[r,u] = BVP1D(Interval,@(r)r,0,0,1,@(r)G/mu*r,0,0);

figure(1); plot(r,u); xlabel( 'radius r'); ylabel('velocity u')

u_exact = G/(4*mu)*((R1^2-r.^2)+(R2^2-R1^2)*ln(r/R1)/ln(R2/R1));
figure(2); plot(r,u-u_exact); xlabel( 'radius r'); ylabel('u-u_{exact}')

Flow = FEM1DIntegrate(r,2*pi*r.*u)
-->
Flow = 0.9716
```

[41] Use en.wikipedia.org/wiki/Hagen-Poiseuille_equation or [Tref75, p. 127].

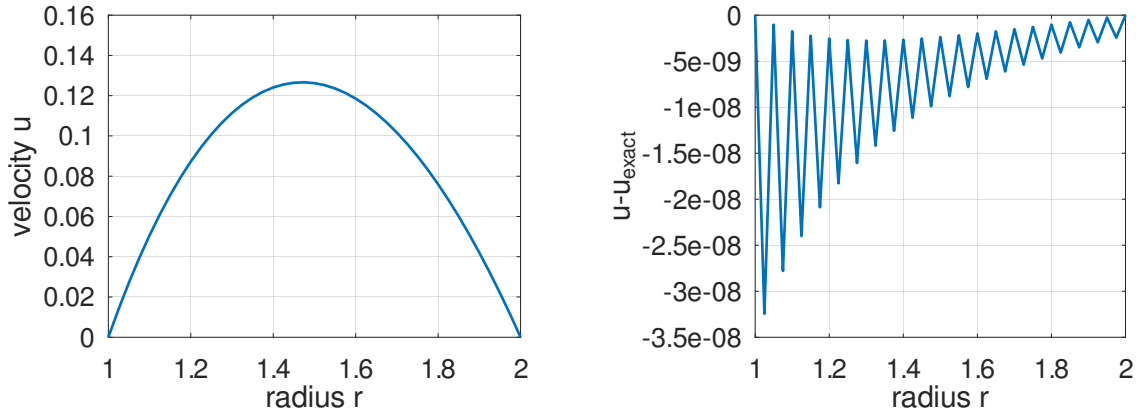


Figure 122: The velocity profile $u(r)$ for a Poiseuille flow through a pipe $1 < r < 2$

9.8.2 Poiseuille flow through an annular tube, 2D solution

On the same annulus Ω, i.e. $R_1 < r < R_2$, the BVP (118) can be examined as a boundary value problem on the domain $\Omega \subset \mathbb{R}^2$. The code `Annulus2D.m` uses the commands `BVP2D()` and `FEMIntegrate()` to generate Figure 123 and estimate the flow by ≈ 0.9713 .

Annulus2D.m

```
G = 1; mu = 1; R1 = 1; R2 = 2;
angles = linspace(0,2*pi,201)'; angles = angles(1:end-1);
xy = [R2*cos(angles),R2*sin(angles),-ones(size(angles))];
Hole.name   = 'Hole';
Hole.border = [R1*cos(angles),R1*sin(angles),-ones(size(angles))];
Hole.point  = [0,0];
Mesh = CreateMeshTriangle('Annulus',xy,0.01,Hole);
Mesh = MeshUpgrade(Mesh,'quadratic');

u = BVP2D(Mesh,1,0,0,0,G/mu,0,0,0);
figure(1); FEMtrimesh(Mesh,u); xlabel('x'); ylabel('y'); zlabel('velocity u')

Flow = FEMIntegrate(Mesh,u)
-->
Flow = 0.9713
```

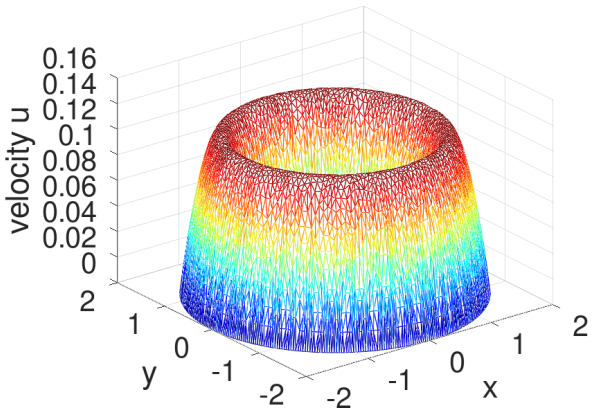


Figure 123: The velocity profile $u(x, y)$ for a Poiseuille flow through a pipe $1 < r < 2$

9.8.3 Poiseuille flow through an annular tube with an offset

The annulus $\Omega \subset \mathbb{R}^2$ can be modified by moving the inner circle, e.g.

$$R_1^2 < (x + \frac{R_1}{4})^2 + y^2 \quad \text{and} \quad x^2 + y^2 < R_2^2 \, .$$

The code `Annulus2Doffset.m` shows the resulting, non–symmetric velocity profile in Figure 124. The flow ≈ 0.8609 is reduced, caused by the offset center. The maximal velocity is higher in the wider section of the annulus.

Annulus2Doffset.m

```
G = 1; mu = 1; R1 = 1; R2 = 2;
angles = linspace(0,2*pi,201)'; angles = angles(1:end-1);
xy = [R2*cos(angles),R2*sin(angles),-ones(size(angles))];
Hole.name   = 'Hole';
Hole.border = [-0.25*R1+R1*cos(angles),R1*sin(angles),-ones(size(angles))];
Hole.point  = [0,0];
Mesh = CreateMeshTriangle('Annulus',xy,0.01,Hole);
Mesh = MeshUpgrade(Mesh,'quadratic');

u = BVP2D(Mesh,1,0,0,0,G/mu,0,0,0);

figure(1); FEMtrimesh(Mesh,u);    xlabel('x'); ylabel('y'); zlabel('velocity u')
figure(2); FEMtricontour(Mesh,u); xlabel('x'); ylabel('y'); axis equal

Flow = FEMIntegrate(Mesh,u)
-->
Flow = 0.8609
```

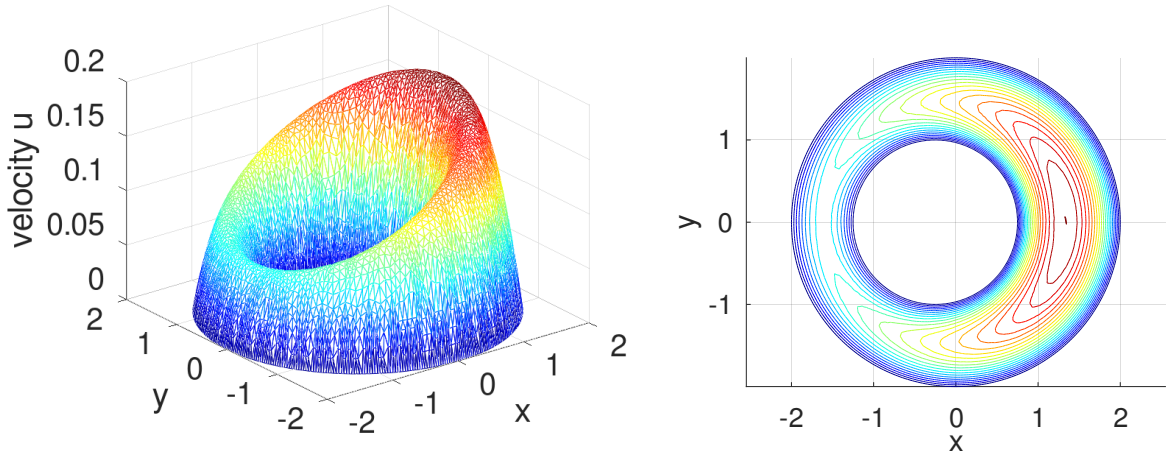


Figure 124: The velocity profile $u(x, y)$ for a Poiseuille flow through a pipe with an offset center

9.9 A potential flow problem

Consider a laminar flow between two plates with an obstacle between the two plates. Assume that the situation is independent on one of the spatial variables and consider a cross section shown in Figure 125. The goal is to find the velocity field $\vec{v}$ of the fluid.

This problem can be solved by introducing a velocity potential $\Phi(x, y)$. The velocity vector $\vec{v}$ is then given by

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = - \begin{pmatrix} \frac{\partial \Phi}{\partial x} \\ \frac{\partial \Phi}{\partial y} \end{pmatrix} .$$

The flow is assumed to be uniform far away from the obstacle. Thus set the potential to $\Phi = 1$ (resp. $\Phi = 0$) at the left (resp. right) end of the plates. Since the fluid can not flow through the boundaries of the plates use that the normal

component of the velocity has to vanish at the upper and lower boundary. The differential equation to be satisfied by Φ is

$$\Delta\Phi = \operatorname{div}(\operatorname{grad}\Phi) = 0$$

In Figure 126 the resulting flow is visualized. Observe the unrealistic velocities at the corners of the domain. The model of laminar flow is not appropriate in this situation. Selecting a finer mesh is no solution to this problem. Mathematically the effect is related to the effect illustrated in Section 9.4.

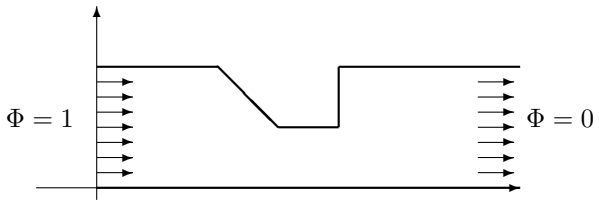


Figure 125: Fluid flow between two plates, the setup

The results are generated by the code below.

PotentialFlow.m

```
%% define the domain
xy = [0 0 -2; 5 0 -1;5 2 -2; 3 2 -2; 3 0.5 -2; 2 0.5 -2; 1 2 -2; 0 2 -1];
if 1      %% linear elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,0.003);
elseif 1  %% quadratic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,4*0.003);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
else      %% cubic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,9*0.003);
  FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endif

x = FEMmesh.nodes(:,1);  y = FEMmesh.nodes(:,2);
function res = gD(xy)   res = 1-xy(:,1)/5;  endfunction
u = BVP2Dsym(FEMmesh,1,0,0,'gD',0,0);
figure(1); FEMtrimesh(FEMmesh,u)
           xlabel('x'); ylabel('y'); zlabel('potential')

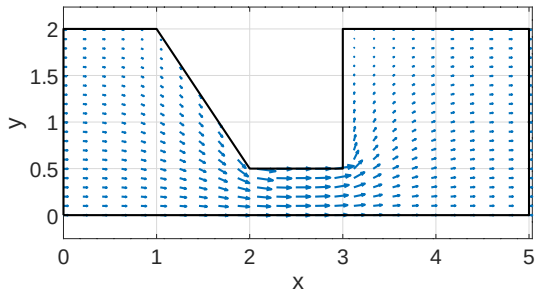
[xx,yy] = meshgrid(linspace(0,5-0.01,25),linspace(0,2-0.01,21));
[u_int,ux_int,uy_int] = FEMgriddata(FEMmesh,-u, xx, yy);

figure(2); quiver(xx,yy,ux_int,uy_int)
           xlabel('x'); ylabel('y');
           hold on; plot([xy(:,1);0],[xy(:,2);0],'k'); hold off; axis equal

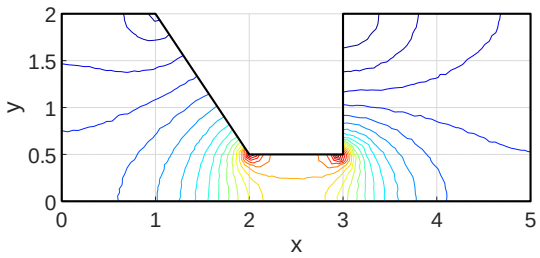
xx = linspace(0,5,101); yy = 0.25*ones(101,1);
[u_int,ux_int,uy_int] = FEMgriddata(FEMmesh,-u,xx,yy);
figure(3); plot(xx,ux_int)
           xlabel('x'); ylabel('horizontal velocity'); ylim([0 0.5])

[ux,uy] = FEMEvaluateGradient(FEMmesh,u);
figure(4); FEMtrimesh(FEMmesh,sqrt(ux.^2+ uy.^2))
           xlabel('x'); ylabel('y'); zlabel('v=|grad u|'); zlim([0 0.5])

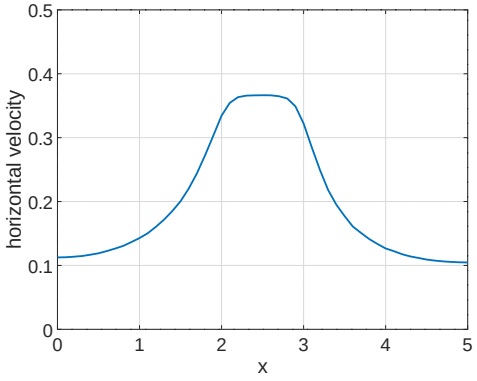
figure(5); FEMtricontour(FEMmesh,sqrt(ux.^2+ uy.^2),21)
           xlabel('x'); ylabel('y'); zlabel('| grad u|')
           hold on; plot([xy(:,1);0],[xy(:,2);0],'k'); hold off
           xlim([0 5]); ylim([0 2]); axis equal
```



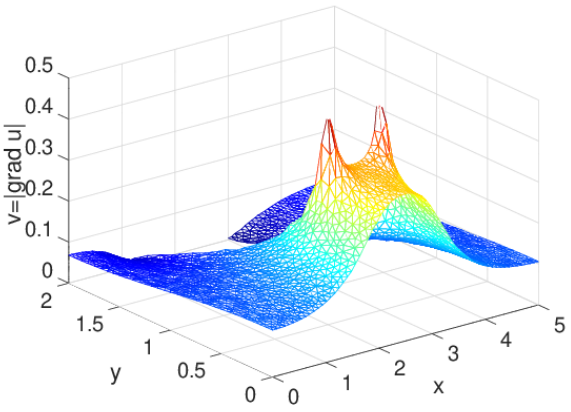
(a) field of velocity vectors



(b) velocity contours



(c) horizontal speed profile along $y = 0.25$



(d) the velocity

Figure 126: Velocity field of an ideal fluid

By integrating the horizontal velocities along vertical cuts observe the flux conservation, i.e whats coming in on the left has to flow through the canal and leave on the right.

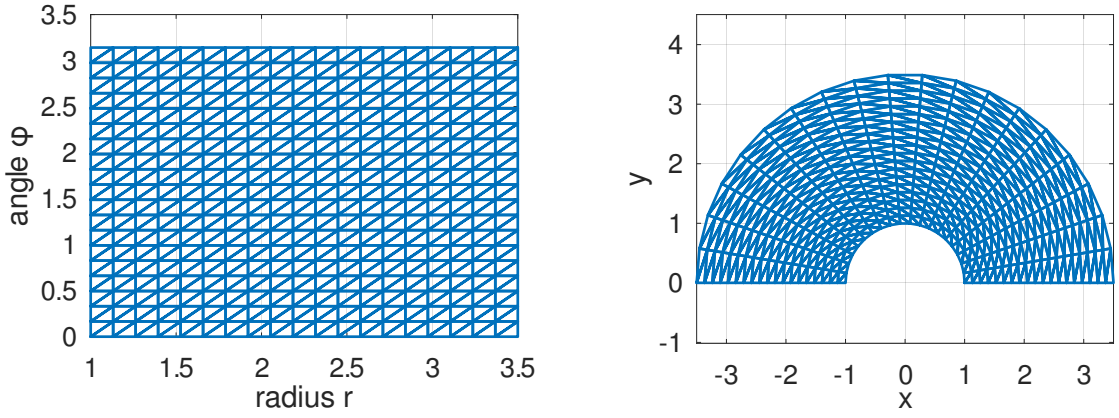
$$\begin{aligned} \text{flux at inlet } x = 0.0 &\approx 0.18337 \\ \text{flux in middle } x = 2.5 &\approx 0.18328 \\ \text{flux at outlet } x = 5.0 &\approx 0.18333 \end{aligned}$$

Selecting a finer mesh or using quadratic elements will make the differences smaller.

```
yy = linspace(0,2); xx = zeros(size(yy));
vx = FEMgriddata(FEMmesh,-ux, xx, yy);          Flux_inlet_ = trapz(yy,vx)
yy = linspace(0,0.5); xx = 2.5*ones(size(yy));
vx = FEMgriddata(FEMmesh,-ux,xx,yy);            Flux_middle = trapz(yy,vx)
yy = linspace(0,2); xx = 5*ones(size(yy));
vx = FEMgriddata(FEMmesh,-ux, xx, yy);          Flux_outlet = trapz(yy,vx)
```

9.10 A potential flow around a cylinder, using polar coodinates

In this section the potential flow around a cylinder is examined, using polar coordinates (r, φ) instead of the commonly used Cartesian coordinates (x, y). The domain to be examined is shown in Figure 127. The flow to be examined far away from the cylinder is moving with speed 1 from left to right, in Cartesian coordinates. This corresponds to a potential $\Phi(x, y) = -x$. Using symmetry only the upper half of the configuration is used for the computations.



(a) polar coordinates (b) Cartesian coordinates

Figure 127: The domains for a potential flow around a cylinder

The potential equation $\Delta\Phi = 0$ is used with polar coordinates (r, φ), i.e.

$$\begin{aligned} 0 &= \Delta\Phi = \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial\Phi}{\partial r}) + \frac{1}{r^2}\frac{\partial^2\Phi}{\partial\varphi^2} \\ 0 &= \frac{\partial}{\partial r}(r\frac{\partial\Phi}{\partial r}) + \frac{1}{r}\frac{\partial^2\Phi}{\partial\varphi^2} \\ 0 &= \begin{bmatrix} \frac{\partial}{\partial r} & \frac{\partial}{\partial\varphi} \end{bmatrix} \begin{bmatrix} r & 0 \\ 0 & \frac{1}{r} \end{bmatrix} \begin{pmatrix} \frac{\partial\Phi}{\partial r} \\ \frac{\partial\Phi}{\partial\varphi} \end{pmatrix} = \nabla_{r\varphi}\left(\begin{bmatrix} r & 0 \\ 0 & \frac{1}{r} \end{bmatrix} \nabla_{r\varphi}\Phi \right) \end{aligned}$$

On the domain in polar coordinates $1 = R_0 \le r \le R_1 = 3.5$ and $0 \le \phi \le \pi$ the boundary conditions are

- Along the outer boundary at $r = R_1 = 3.5$ and $0 \le \varphi \le \pi$ use $\Phi(r, \varphi) = -x = -r\,\cos(\varphi)$.
- Along all other borders use zero Neumann conditions.

- Along $r = R_0 = 1$ and $0 \leq \varphi \leq \pi$ this leads to $\frac{\partial \Phi}{\partial r} = 0$, i.e. tangential flow.
- Along $R_0 \leq r \leq R_1$ and $\varphi = 0$ or $\varphi = \pi$ this leads to $\frac{\partial \Phi}{\partial \varphi} = 0$, based on symmetry.

With this information the boundary value problem can be solved for the potential $\Phi(r, \varphi)$.

PotentialFlowPolar.m

```
R0 = 1; R1 = 3.5; N = 20;
%% create the mesh
FEMmesh = CreateMeshRect(linspace(R0,R1,N),linspace(0,pi,N),-2,-2,-2,-1);
figure(1); FEMtrimesh(FEMmesh); xlabel('radius r'); ylabel('angle \phi')
function xy = Deform(r_phi)
  r = r_phi(:,1); phi = r_phi(:,2);
  xy = [r.*cos(phi),r.*sin(phi)];
endfunction
figure(11); FEMtrimesh(MeshDeform(FEMmesh,'Deform'));
            axis equal; xlabel('x'); ylabel('y');
FEMmesh = MeshUpgrade(FEMmesh,"cubic");

%% define the functions
function res = gD(r_phi)
  res = -r_phi(:,1).*cos(r_phi(:,2));
endfunction

function res = a(r_phi)
   r = r_phi(:,1);
   res = [r,1./r,0*r];
endfunction

%% solve the BVP
Phi = BVP2Dsym(FEMmesh,"a",0,0,"gD",0,0);
```

Using `FEMEvaluateGradient()` the partial derivatives $\frac{\partial \Phi}{\partial r}$ and $\frac{\partial \Phi}{\partial \varphi}$ can be evaluated. The gradient is given by

$$-\vec{v} = \nabla \Phi = \frac{\partial \Phi}{\partial r} \vec{e}_r + \frac{1}{r} \frac{\partial \Phi}{\partial \varphi} \vec{e}_\phi = \frac{\partial \Phi}{\partial r} \begin{pmatrix} +\cos(\varphi) \\ +\sin(\varphi) \end{pmatrix} + \frac{1}{r} \frac{\partial \Phi}{\partial \varphi} \begin{pmatrix} -\sin(\varphi) \\ +\cos(\varphi) \end{pmatrix}$$

This leads to the velocity $v = \|\vec{v}\|$.

$$v^2 = \|\nabla \Phi\|^2 = \left(\frac{\partial \Phi}{\partial r} \right)^2 + \frac{1}{r^2} \left(\frac{\partial \Phi}{\partial \varphi} \right)^2$$

The values are on the nodes of the mesh with polar coordinates in Figure 127(a). Convert the mesh with cubic elements back to a linear mesh and deform it to Cartesian coordinates in Figure 127(b). On this mesh display the velocity v.

PotentialFlowPolar.m

```
figure(2); FEMtrimesh(FEMmesh,Phi)
           xlabel('r'); ylabel('\phi'); zlabel('potential \Phi')
[u_r,u_phi] = FEMEvaluateGradient(FEMmesh,Phi);
v = sqrt(u_r.^2+(u_phi./FEMmesh.nodes(:,1)).^2);
figure(3); FEMtrimesh(FEMmesh,v)
           xlabel('r'); ylabel('\phi'); zlabel('velocity');
FEMmeshCartesian = MeshDeform(MeshCubic2Linear(FEMmesh),'Deform');
figure(33); FEMtrimesh(FEMmeshCartesian,v)
            xlabel('x'); ylabel('y'); zlabel('speed');
            xlim([-R1,+R1]);ylim([0,R1]); zlim([0 2]);
figure(34); FEMtrimesh(FEMmeshCartesian,v); view([0,90])
            xlabel('x'); ylabel('y'); colorbar; axis equal

%% evaluate the  gradient as vector field
r = FEMmesh.nodes(:,1); phi = FEMmesh.nodes(:,2);
```

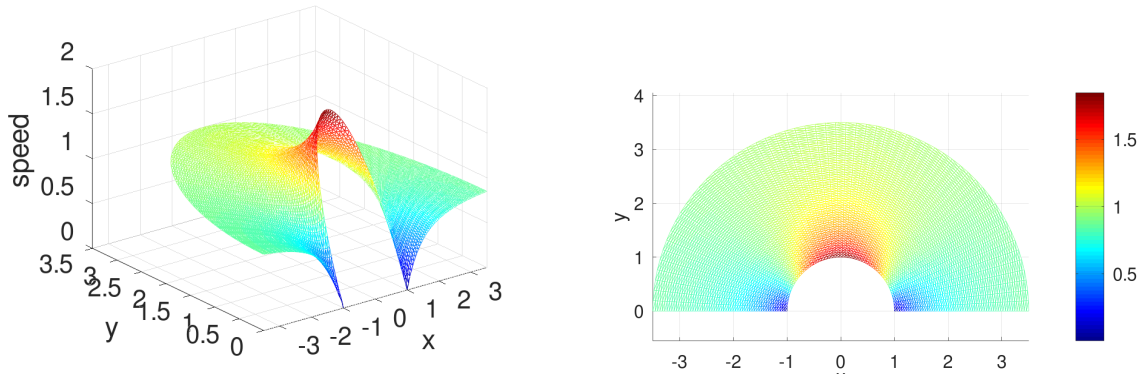


Figure 128: The speed of the flow aound a cylinder

```
x = r.*cos(phi); y = r.*sin(phi);
u_x = +cos(phi).*u_r - sin(phi).*u_phi./r;
u_y = +sin(phi).*u_r + cos(phi).*u_phi./r;
figure(4); quiver(x,y,-u_x,-u_y)
           xlabel('x'); ylabel('y'); xlim([-2,2]); ylim([0,2])
```

To evaluate the flow lines the gradient $\nabla\Phi$ has to be evaluated on a regular mesh, generated by `meshgrid()`. The occurring `NaN` values for $\frac{\partial\Phi}{\partial r}$ and $\frac{\partial\Phi}{\partial\varphi}$ are replaced by 0, thus the flow does not pass through. In Figure 129 find the vector field and some flow lines.

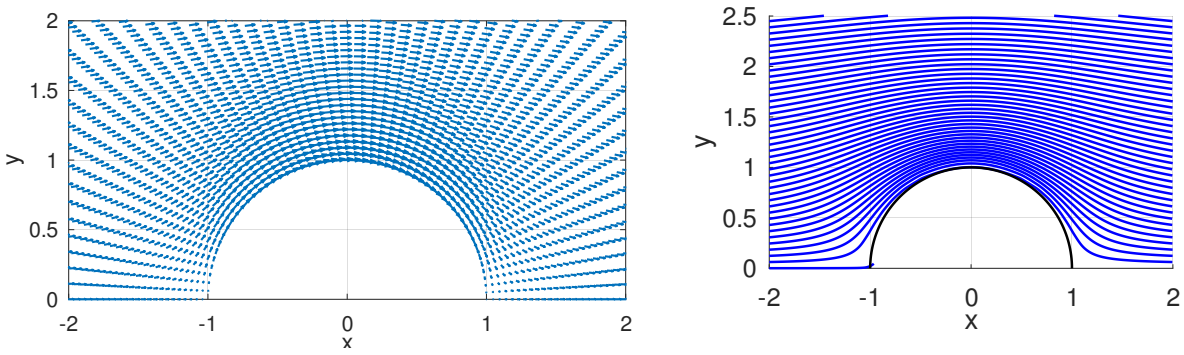


Figure 129: The vector field and flow lines for a flow around a cylinder

PotentialFlowPolar.m

```
%% find the flow lines
[xx,yy] = meshgrid(linspace(-2,2,51),linspace(0,3,25));
r_int = sqrt(xx.^2+yy.^2); phi_int = atan2(yy,xx);
[u_int,ur_int,uphi_int] = FEMgriddata(FEMmesh,-Phi,r_int,phi_int);
ux_int = +cos(phi_int).*ur_int - sin(phi_int).*uphi_int./r_int;
uy_int = +sin(phi_int).*ur_int + cos(phi_int).*uphi_int./r_int;
ux_int(find(isnan(ux_int))) = 0;  uy_int(find(isnan(uy_int))) = 0;

figure(5); clf; NN = 50; alpha = linspace(0,pi);
           streamline(xx,yy,ux_int,uy_int,-2*ones(1,NN),...
                      linspace(0.001,3,NN),[0.1,10000]);
         hold on; plot(cos(alpha),sin(alpha),'k'); hold off; axis equal
         xlabel('x'); ylabel('y'); xlim([-2 2]); ylim([0,2.5]);
```

The above problem can be solved on the domain in Figure 127(b) using Cartesian coordinates. The FEMoctave code is very similar. It is in fact a bit shorter, since no transformation from polar to Cartesian coordinates is necessary.

PotentialFlowCartesian.m

```
R0 = 1; R1 = 3.5; N = 20;
%% create the mesh
FEMmesh = CreateMeshRect(linspace(R0,R1,N),linspace(0,pi,N),-2,-2,-2,-1);
function xy = Deform(r_phi)
  r = r_phi(:,1); phi = r_phi(:,2);
  xy = [r.*cos(phi),r.*sin(phi)];
endfunction
FEMmesh = MeshDeform(FEMmesh,'Deform');
figure(1); FEMtrimesh(FEMmesh); axis equal; xlabel('x'); ylabel('y');
FEMmesh = MeshUpgrade(FEMmesh,"cubic");

%% define the function
function res = gD(xy) res = -xy(:,1); endfunction

Phi = BVP2Dsym(FEMmesh,1,0,0,"gD",0,0); %% solve the BVP

figure(2); FEMtrimesh(FEMmesh,Phi)
           xlabel('r'); ylabel('\phi'); zlabel('potential \Phi')
[u_x,u_y] = FEMEvaluateGradient(FEMmesh,Phi);
v = sqrt(u_x.^2+u_y.^2);
figure(3); FEMtrimesh(FEMmesh,v)
           xlabel('x'); ylabel('y'); zlabel('velocity');
figure(34); FEMtrimesh(FEMmesh,v); view([0,90])
            xlabel('x'); ylabel('y'); colorbar; axis equal
%% evaluate the  gradient as vector field
x = FEMmesh.nodes(:,1); y = FEMmesh.nodes(:,2);
figure(4); quiver(x,y,-u_x,-u_y)
           xlabel('x'); ylabel('y'); xlim([-2,2]); ylim([0,2]); axis equal
%% find the flow lines
[xx,yy] = meshgrid(linspace(-2,2,51),linspace(0,3,25));
[u_int,ux_int,uy_int] = FEMgriddata(FEMmesh,-Phi,xx,yy);
ux_int(find(isnan(ux_int))) = 0;  uy_int(find(isnan(uy_int))) = 0;
figure(5); clf; NN = 50; alpha = linspace(0,pi);
         streamline(xx,yy,ux_int,uy_int,-2*ones(1,NN),...
                    linspace(0.001,3,NN),[0.1,10000]);
         hold on; plot(cos(alpha),sin(alpha),'k'); hold off; axis equal
         xlabel('x'); ylabel('y'); xlim([-2 2]); ylim([0,2.5]);
```

9.11 A potential flow problem in a circular pipe

An ideal liquid is flowing through a circular pipe with diminished radius in a central section. The outer radius is given by

$$R(z) = \begin{cases} 2 & \text{for} \quad |z| \geq 1 \\ 2 - \cos^2(\frac{\pi}{2}\, z) & \text{for} \quad |z| \leq 1 \end{cases} .$$

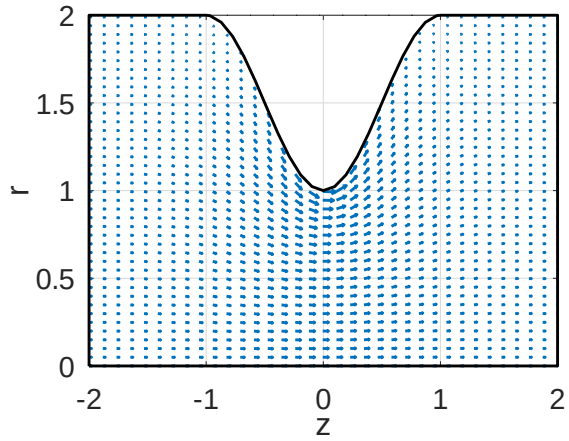
The upper half of a section is visible in Figure 130. Assuming that the solution is independent on the angle θ the equation $\Delta\Phi = 0$ has to be reformulated in cylindrical coordinates and simplified.

$$\begin{aligned} 0 &= \Delta\Phi = \operatorname{div}(\operatorname{grad}\Phi) = \Phi_{rr} + \frac{1}{r}\,\Phi_r + \frac{1}{r^2}\,\Phi_{\theta\theta} + \Phi_{zz} \\ 0 &= r\left(\Phi_{rr} + \frac{1}{r}\,\Phi_r + \Phi_{zz}\right) = r\,\Phi_{rr} + \Phi_r + r\,\Phi_{zz} = \frac{\partial}{\partial r}\,(r\,\Phi_r) + \frac{\partial}{\partial z}\,(r\,\Phi_z) \,. \end{aligned}$$

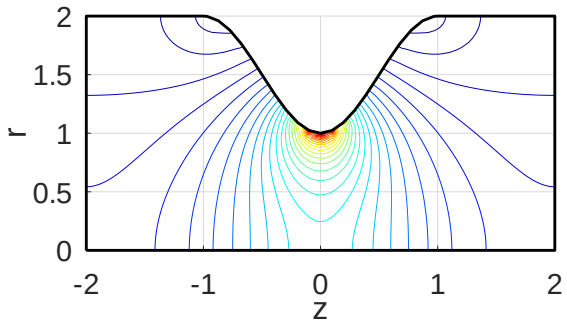
Setting $\Phi = +1$ at the left edge and $\Phi = -1$ at the right edge, the BVP can be solved for the potential $\Phi(z, r)$ with the help of FEMoctave. The velocity vector is again given by the gradient

$$\vec{v} = \begin{pmatrix} v_z \\ v_r \end{pmatrix} = -\begin{pmatrix} \frac{\partial\,\Phi}{\partial z} \\ \frac{\partial\,\Phi}{\partial r} \end{pmatrix} .$$

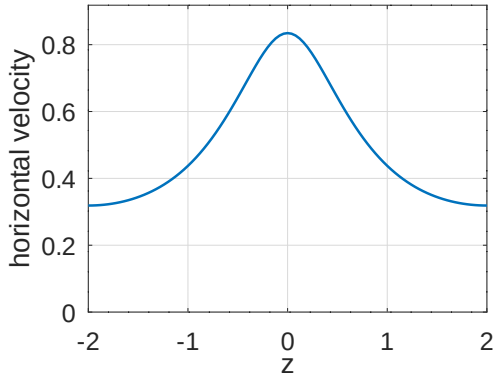
Observe that there are no singularities for the velocities, compared to the previous section 9.9, since there are no sharp corners in the domain.



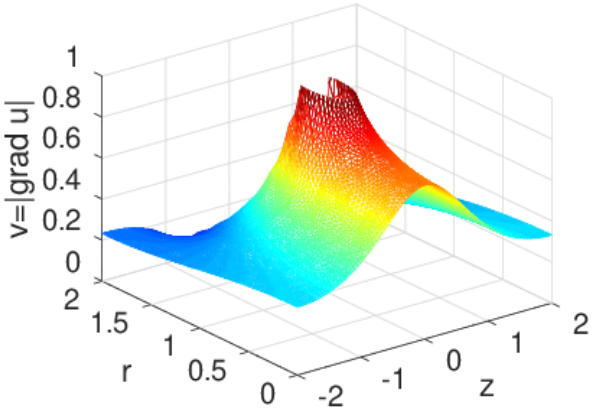
(a) field of velocity vectors



(b) velocity contours



(c) horizontal speed profile along $r = 0.5$



(d) the velocity

Figure 130: Velocity field of a ideal fluid in a circular pipe

PotentialFlowCircular.m

```
%% define the domain and mesh
R = 2; R_in = 1.0; area = 0.001;
z = linspace(-1+sqrt(area),1-sqrt(area),21)'; r = R-R_in*cos(pi/2*z).^2;
b = -2*ones(size(z));
zr = [-2 0 -1; -2 R -2; -1 R -2; [z,r,b]; 1 R -2; 2 R -1; 2 0 -2];
if 0      %% linear elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',zr,area);
elseif 0 %% quadratic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',zr,4*area);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
else    %% cubic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',zr,9*area);
  FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endif

z = FEMmesh.nodes(:,1);  z = FEMmesh.nodes(:,2);
function res = gD(zr)       res = -zr(:,1)/2;  endfunction
function res = a_coeff(zr)  res = zr(:,2);     endfunction
```

```
u = BVP2Dsym(FEMmesh,'a_coeff',0,0,'gD',0,0);

[zz,rr] = meshgrid(linspace(-2,2-0.01,35),linspace(0,R-0.01,41));
[u_int,uz_int,ur_int] = FEMgriddata(FEMmesh,-u, zz, rr);

figure(1); quiver(zz,rr,uz_int,ur_int)
           xlabel('z'); ylabel('r');
           hold on; plot([zr(:,1);-2],[zr(:,2);0],'k'); hold off
           xlim([-2,2]); ylim([0,R]);

[uz,ur] = FEMEvaluateGradient(FEMmesh,u);
figure(2); FEMtrimesh(FEMmesh,sqrt(uz.^2+ ur.^2))
           xlabel('z'); ylabel('r'); zlabel('v=|grad u|')
           zlim([0 1]); caxis([0,1])

zz = linspace(-2,2,101); rr = 0.5*ones(101,1);
[u_int,uz_int,ur_int] = FEMgriddata(FEMmesh,-u,zz,rr);
figure(3); plot(zz,uz_int)
           xlabel('z'); ylabel('horizontal velocity');
           ylim([0 1.1*max(uz_int)]

figure(4); FEMtricontour(FEMmesh,sqrt(uz.^2+ ur.^2),31)
           xlabel('z'); ylabel('r'); zlabel('|grad u|')
           hold on; plot([zr(:,1);-2],[zr(:,2);0],'k'); hold off
           xlim([-2 2]); ylim([0 R]); axis equal
```

The total flux accross a vertical line $z = \text{const}$ can be determined by the integral

$$\text{flux} = \int_0^{R(z)} v_z(r,z)\, 2\,\pi\, r\; dr = 2\,\pi \int_0^{R(z)} -\frac{\partial\,\Phi(z,r)}{\partial z}\, r\; dr\;.$$

```
rr = linspace(0,R); zz = -1.9*ones(size(rr)); vz = FEMgriddata(FEMmesh,-uz, zz, rr);
Flux_inlet  = trapz(rr,rr.*vz)*2*pi
rr = linspace(0,R-R_in); zz = 0*ones(size(rr)); vz = FEMgriddata(FEMmesh,-uz,zz,rr);
Flux_middle = trapz(rr,rr.*vz)*2*pi
rr = linspace(0,R); zz = 1.9*ones(size(rr)); vz = FEMgriddata(FEMmesh,-uz, zz, rr);
Flux_outlet = trapz(rr,rr.*vz)*2*pi
-->
Flux_inlet  = 3.3115
Flux_middle = 3.2897
Flux_outlet = 3.3115
```

The accurracy of the numerical results

$$\begin{aligned} \text{flux at inlet } z = -1.9 &\approx 3.3115 \\ \text{flux in middle } x = +0.0 &\approx 3.2897 \\ \text{flux at outlet } x = +1.9 &\approx 3.3115 \end{aligned}$$

could be improved by a finer mesh. This would verify the conservation of flux at different z–levels.

9.12 A potential flow around a wing profile

For many years it was common knowledge that the Bernoulli's law leads to the main force allowing planes to fly. See e.g. https://www.scientificamerican.com/article/no-one-can-explain-why-planes-stay-in-the-air/. Now one should know differently, but the setup is an interesting problem to examine. Thus examine the air flow around a wing profile. The velocity v along the surface of the wing leads to a pressure proportional to v^2.

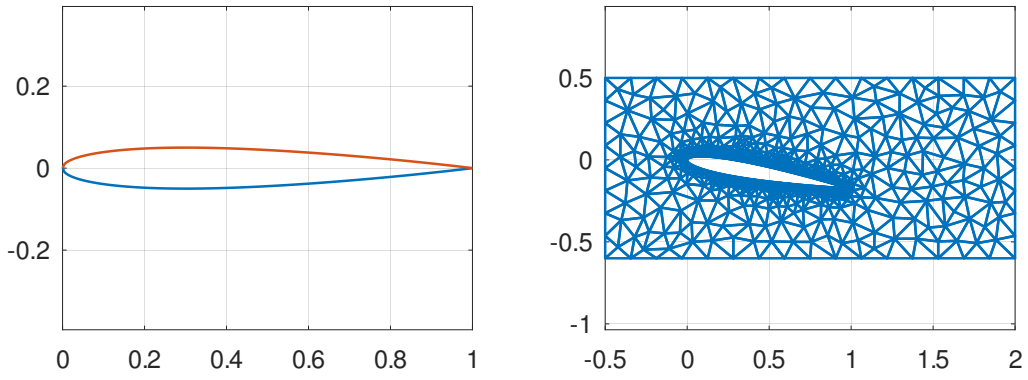
- Examine a potential flow (https://en.wikipedia.org/wiki/Potential_flow) around a wing profile. Find examples of wing profiles at https://en.wikipedia.org/wiki/NACA_airfoil. For an NACAxx profile for $0 \le x \le 1$ with maximal height T the curves for the upper and lower part are given by

$$h = \pm 5 \cdot T \cdot (0.2969\sqrt{x} - 0.1260\,x - 0.3516\,x^2 + 0.2843\,x^3 - (0.1015 - 0.0021) * x^4) \; .$$

  The correction of the last term assures that the profile satisfies $h(1) = 0$, i.e. it is a closed curve. This is implemented as function `FoilThickness()` and a sample foil is shown in Figure 131. This foils is then rotated by an angle, e.g. by $10°$ with `Angle = -10/180*pi`. Using the slopes of the foil function the angles of the normal vectors are computed along both edges of the foil.

- On a domain $-0.5 \le x \le 2$ and $-0.6 \le y \le 0.5$ with the hole given by the wing profile the potential equation $\Delta\varphi = 0$ is solved with boundary conditions $\varphi(x, y) = -x$ along the two edges at $x = -0.5$ and $x = +2$. On the other boundaries a Neumann condition $\frac{\partial \varphi}{\partial \vec{n}} = 0$ is used. Thus the gradient $\nabla\varphi$ will be tangential to the boundaries.

- With `BVP2Dsym()` and `FEMEvaluateGradient()` the potential $\varphi(x, y)$ and its partial derivatives are evaluated.



(a) the wing profile (b) the mesh with the rotated wing

Figure 131: The wing profile and the mesh with the rotated wing

Wing.m

```
x = linspace(0,1,200)';
function [height,slope] = FoilThickness(x)
  FoilT  = 0.10; %% thickness of profile
  height = 5*FoilT*(0.2969*sqrt(x)-0.1260*x-0.3516*x.^2+0.2843*x.^3-0.1015*x.^4);
  fix = height(end)/(5*FoilT);
  height = 5*FoilT*(0.2969*sqrt(x)-0.1260*x-0.3516*x.^2+0.2843*x.^3-(0.1015+fix)*x.^4);
  slope = 5*FoilT*(0.2969*0.5./sqrt(x)-0.1260-2*0.3516*x+3*0.2843*x.^2-4*(0.1015+fix)*x.^3);
  slope(1) = (2*sqrt(2)-1)*slope(2);
endfunction
[height,slope] = FoilThickness(x);
figure(1); plot(x,[-height,height]); axis equal
Angle = -10/180*pi;  %% angle of attack
R = [cos(Angle),-sin(Angle);+sin(Angle),cos(Angle)];
Upper = (R*[x,height]')';  Lower = (R*[x,-height]')';
AngleNormalUpper = atan2(+1,-slope)+Angle;
AngleNormalLower = atan2(-1,-slope)+Angle;

DomainHole = [flipud(Upper);Lower(1:end-1,:)];
Hole.name = 'hole';
Hole.border = [DomainHole,-2*ones(length(DomainHole),1)];
Hole.point = [0.1,0];
Borders = [-0.5,-0.6,-2;2,-0.6,-1;2,0.5,-2;-0.5,0.5,-1];
```

```
Mesh = CreateMeshTriangle('Wing',Borders,1e-2,Hole);
figure(2); FEMtrimesh(Mesh); axis equal
Mesh = MeshUpgrade(Mesh,'cubic');

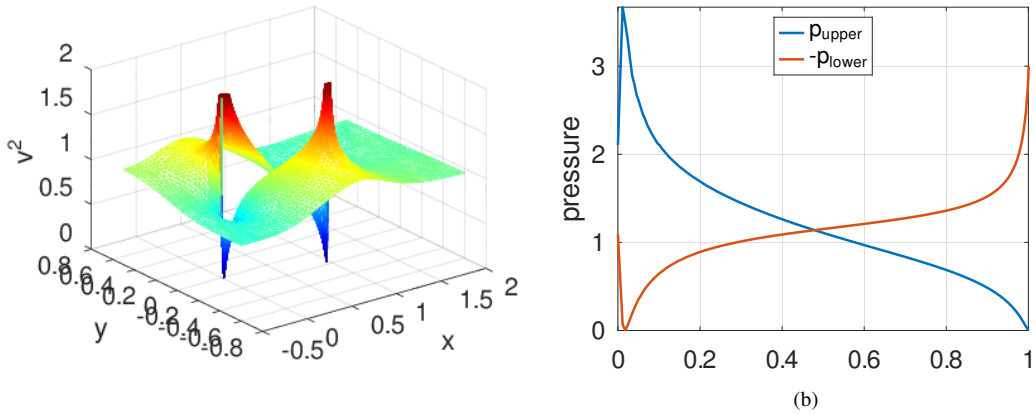
function res = gD(xy)
  res = -xy(:,1);
endfunction

u = BVP2Dsym(Mesh,1,0,0,'gD',0,0);
[ux,uy] = FEMEvaluateGradient(Mesh,u);
```

- With the above result the speed v can be evaluated, using $v^2 = (\frac{\partial \varphi}{\partial x})^2 + (\frac{\partial \varphi}{\partial y})^2$. According to Bernoulli's law the change of pressure p is proportional to $-v^2$. Thus the graphs (Figure 132) and contour lines (Figure 133) of v^2 provide information on the pressure distribution.
- To find the vertical force density on the wing determine the vertical component of the pressure based force by using the angle of the normal vectors. Find the graphs in Figure 132. For the total vertical force use the normal angle α and evaluate the vertical component of force (per length) with $p\ \cos(\alpha)$. Then integrate along the upper and lower edge of the foil using the arc length elements.

$$F_{upper} = \int_0^1 p\ \cos(\alpha_{upper}(x))\ \sqrt{1 + (y'(x))^2}\ dx\ = \int_0^1 p\ \cos(\alpha_{upper}(x))\, ds$$

Use $ds \approx \Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2}$ and the trapezoidal rule `trapz()` to evaluate the integrals. The difference $F_{upper} - F_{lower}$ is an estimate for the total lift.



(b)

Figure 132: The surface for v^2 and the vertical pressure along the upper and lower edge of the wing

Wing.m

```
v = sqrt(sum([ux.^2,uy.^2],2));
figure(3); FEMtrimesh(Mesh,v.^2); xlabel('x'); ylabel('y'); zlabel('v^2');
           zlim([0,2]); caxis([min(v.^2),2])
Levels = [0.5:0.1:2];
figure(4); clf; FEMtricontour(Mesh,v.^2,Levels); xlabel('x'); ylabel('y');
           title('contours of v^2'); colorbar(); axis equal
           hold on; plot(DomainHole(:,1),DomainHole(:,2),'k'); hold off

x = x(1:end-1); Upper = Upper(1:end-1,:); Lower = Lower(1:end-1,:);
AngleNormalUpper = AngleNormalUpper(1:end-1);
AngleNormalLower = AngleNormalLower(1:end-1);

function res = ArcLength(x,y);
  dx = diff(x); dy = diff(y); ds = sqrt(dx.^2+dy.^2);
```

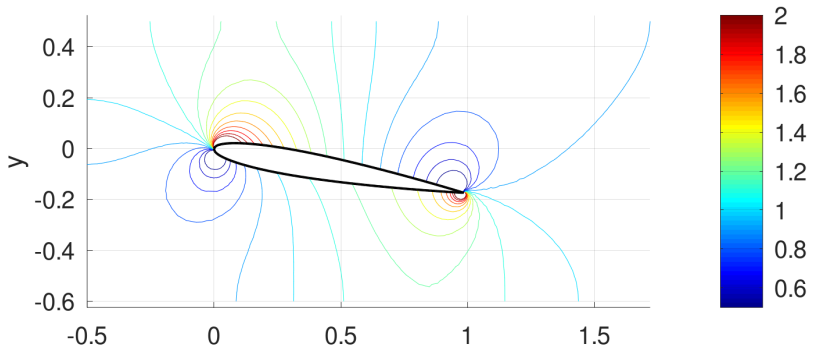


Figure 133: The contour plot for the pressure

```
  res = [0;cumsum(ds)];
endfunction

dsUpper = ArcLength(Upper(:,1),Upper(:,2));
pUpper = FEMgriddata(Mesh,v.^2,Upper(:,1),Upper(:,2)).*cos(AngleNormalUpper);
ForceUpper = trapz(dsUpper,pUpper)
dsLower = ArcLength(Lower(:,1),Lower(:,2));
pLower = FEMgriddata(Mesh,v.^2,Lower(:,1),Lower(:,2)).*cos(AngleNormalLower);
ForceLower = trapz(dsLower,pLower)

figure(5); plot(dsUpper,pUpper,dsLower,-pLower); ylabel('pressure');
           legend('p_{upper}','-p_{lower}','location','north');
           xlim([0,1]); ylim([0,max(pUpper)])
```

- Using the vector field generated by $\nabla\varphi$ the command `streamline()` will generate the streamlines, leading to Figure 134.

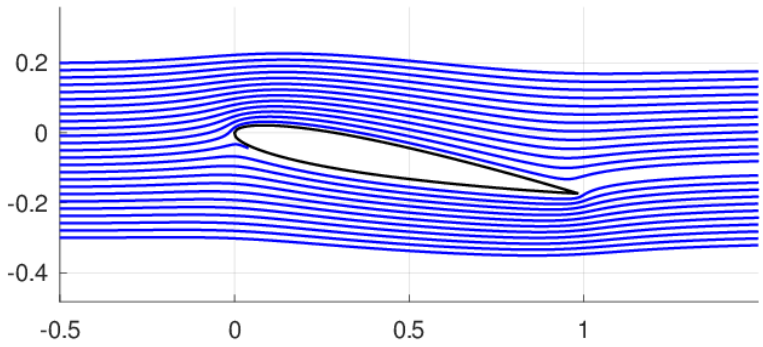


Figure 134: The flow lines around the wing

Wing.m

```
[xx,yy] = meshgrid(linspace(-0.5,1.5,201),linspace(-0.5,0.5,201));
[ui,uxi,uyi] = FEMgriddata(Mesh,u,xx,yy);

figure(6); clf; NN = 25;
           streamline(xx,yy,-uxi,-uyi,-0.5*ones(1,NN),linspace(-0.3,0.2,NN),[0.1,10000]);
           hold on; plot(DomainHole(:,1),DomainHole(:,2),'k'); hold off; axis equal
```

9.13 A minimal surface problem

Let $u(x,y)$ be the hight of a surface above the border of a 2-dimensional domain Ω is given by a function $g(x,y)$. Then the function u representing the surface of minimal with has to solve a nonlinear PDE.

$$\begin{aligned} \nabla\left(\frac{1}{\sqrt{1+|\nabla u|^2}}\nabla u\right) &= 0 \qquad \text{in domain} \quad \Omega \\ u &= g \qquad \text{on} \quad \Gamma = \partial\Omega \end{aligned}$$

FEMoctave is not directly capable of solving non linear problems, but a simple iteration will lead to an approximation

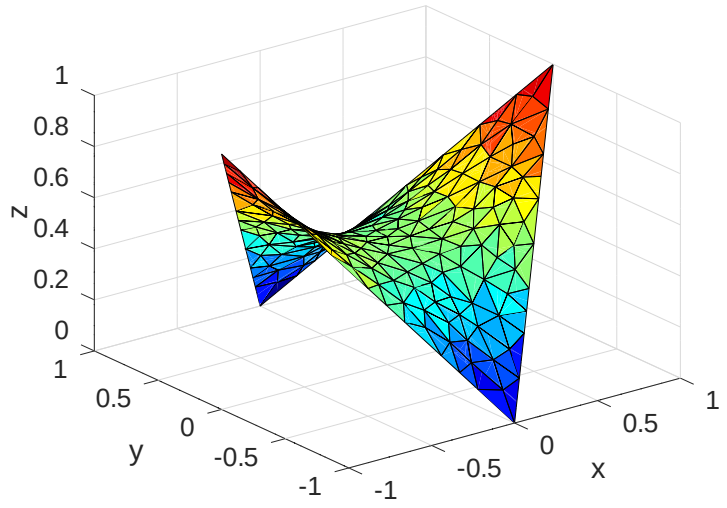


Figure 135: A minimal surface

of the solution.

- start with an initial solution $u_0(x,y) = 0$
- repeat until the change in solution is small enough
  - compute the coefficient function
$$a(x,y) = \frac{1}{\sqrt{1+\|\nabla u(x,y)\|^2}}$$
  - Solve the boundary value problem

$$\begin{aligned} \operatorname{div}(a(x,y)\ \operatorname{grad} u) &= 0 \qquad \text{in the domain} \quad \Omega \\ u &= g \qquad \text{on} \quad \Gamma = \partial\Omega \end{aligned}$$

The code below implements this algorithm for a square Ω and leads to the result in Figure 135. While iterating the area of each surface is determined by integrating

$$\text{area} = \iint_\Omega \sqrt{1+\|\nabla u\|^2}\, dA$$

and the average difference of subsequent solutions is computed.

MinimalSurface.m

```
xy = [1,0,-1;0,1,-1;-1,0,-1;0,-1,-1];
FEMmesh = CreateMeshTriangle("square",xy,0.01);
%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

x = FEMmesh.nodes(:,1); y = FEMmesh.nodes(:,2);
function res = BC(xy)  res = abs(xy(:,1));  endfunction

u = BVP2Dsym(FEMmesh,1,0,0,'BC',0,0);
difference = zeros(5,1); area = difference;
for ii = 1:5
  [~,grad] = FEMEvaluateGP(FEMmesh,u);
  coeff = sqrt(1+grad(:,1).^2+ grad(:,2).^2);
  area(ii) = FEMIntegrate(FEMmesh,coeff);
  u_new = BVP2Dsym(FEMmesh,coeff,0,0,'BC',0,0);
  difference(ii) = mean(abs(u_new-u));
  u = u_new;
endfor

Area_Difference = [area,difference]
figure(1); FEMtrisurf(FEMmesh,x,y,u)
           xlabel('x'); ylabel('y'); zlabel('z')
-->
Area_Difference =    2.30454229746   0.00271116350
                     2.30609424101   0.00030136719
                     2.30586894444   0.00003705316
                     2.30589632291   0.00000508928
                     2.30589260378   0.00000078521
```

By choosing quadratic or cubic elements, or a finer mesh, one can observe that the computed minimal area will be smaller. This should not come as a surprise, the better the resolution, the smaller the minimal area.

9.14 Computing a capacitance

9.14.1 State the problem

Examine a circular plate capacitance as shown in Figure 136. Based on the radial symmetry one should be able to consider a two dimensional section only for the computations.

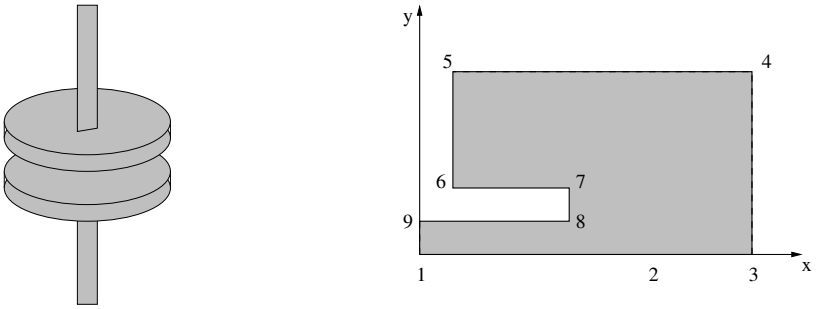


Figure 136: The capacitance and the section used for the modeling

Consider the voltage u as unknown. On the upper conductor assume $u = 1$ and on the lower conductor $u = -1$. Based on the symmetry consider a section only and use $u = 0$ in the plane centered between the conductors. Use the Laplace operator in cylindrical coordinates. Thus the following boundary value problem has to be solved.

$$\begin{aligned} \operatorname{div}(x\ \operatorname{grad} u(x,y)) &= 0 && \text{in domain} \\ u(x,0) &= 0 && \text{along edge } y = 0 \\ u(x,y) &= 1 && \text{along edges of upper conductor} \\ \tfrac{\partial}{\partial n} u(x,y) &= 0 && \text{on remaining boundary} \end{aligned} \tag{119}$$

Assume that the domain is embedded in the rectangle $0 \le x \le R$ and $0 \le y \le H$. The lower edge of the conductor is at $y = h$ and $0 \le x \le r$. If $h \ll r$ expect the gradient of u to be $1/h$ between the plates and zero away from the plates. Thus

$$\text{flux} = \iint\limits_{\text{disk}} \vec{n} \cdot \operatorname{grad} u \, dA = 2\,\pi \int_0^R x \, \frac{\partial\, u}{\partial y} \, dx \approx 2\,\pi \int_0^r x \, \frac{1}{h} \, dx = \frac{\pi\, r^2}{h} \, .$$

Because the electric field will not be homogeneous around the boundaries of the disk expect deviations from the result of an idealized circular disk. With the divergence theorem and a physical argument one can verify that the flux trough the midplane is proportional to the capacitance. By applying the following steps compute the capacitance by analyzing the solution of a boundary value problem.

1. Create a mesh for the domain in question.
2. Define parameters and boundary conditions.
3. Solve the partial differential equation and visualize the solution.
4. Compute the flux through the midplane as an integral to determine the capacitance.

9.14.2 Create the mesh and solve the BVP

According to Figure 136 create a mesh with the following data.

$h = 0.2$	distance between midplane and lower edge of capacitance
$r = 1.0$	radius of disk of the capacitance
$H = 0.5$	height of the enclosing rectangle
$R = 2.5$	radius of the enclosing rectangle

As input for the mesh generating code `triangle` (see [www:triangle]) use

- the coordinates of the corner points, numbered according to Figure 136
- a list of all the connecting edges and the type of boundary conditions to be used
- information of the desired area of the triangles to be generated

Then use two different sizes of the triangles since a finer mesh between the plates is required, expecting large variations in the solution. The file `capacitance.poly` provides this information. The numbering of the nodes is visible in Figure 136. With the above use the program `triangle` to generate a mesh.

```
triangle -pqa capacitance.poly
```

The mesh consists of 2189 nodes, forming 4036 triangles.

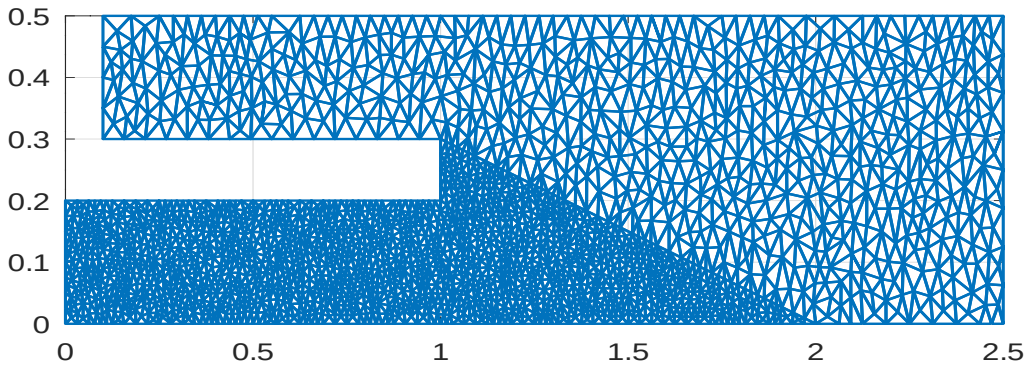


Figure 137: A mesh on the domain

To solve the BVP (119) one needs a definition of the coefficient function and the Dirichlet boundary function. Then set up and solve the system of linear equations. This leads to a system for 1937 unknowns. Then generate a plot of the voltage $u(x, y)$ and its level curves. Find the results in Figure 139.

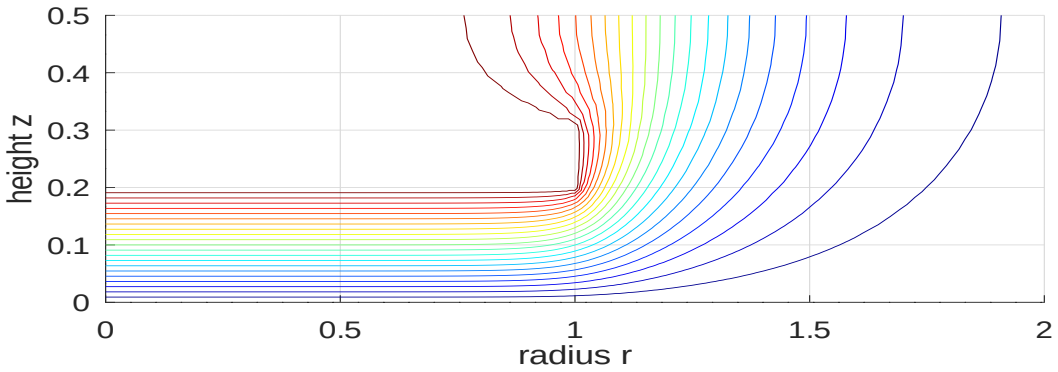
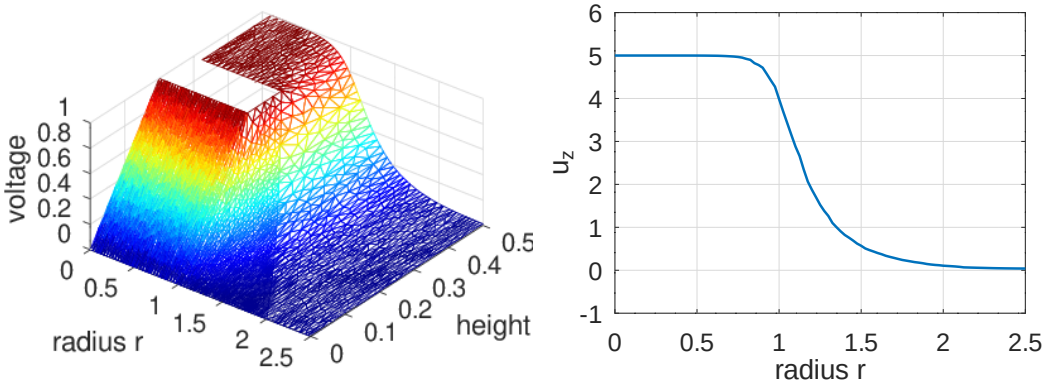


Figure 138: The contour lines of the resulting voltage



(a) the voltage

(b) vertical field along the horizontal axis

Figure 139: Voltage plot and electric field between the plates of the capacitance

9.14.3 Compute the capacitance

It remains to compute the flux through the midplane. For this start out by computing the gradient of the voltage u along the line $y = 0$. Find the plot of the normal component in Figure 139. The graph confirms that between the plates the gradient is approximately $1/h = 1/0.2 = 5$ and vanishes away from the plate. Then a trapezoidal rule is used to determine the flux accross the midplane with the integral.

$$\text{flux} = \iint_{\text{disk}} \vec{n} \cdot \operatorname{grad} u \, dA = 2\,\pi \int_0^R x \, \frac{\partial\, u}{\partial y} \, dx$$

For the selected values of h, H, r and R obtain a factor of 1.5 between result of the boundary value problem and the idealized approximation $\pi\, r^2/h$. Thus the simple formula is not a good approximation, the distance h is too large compared to the radius r.

Capacitance.m

```
FEMmesh = ReadMeshTriangle('capacitance.1');
%% FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment fora quadratic mesh
figure(1); FEMtrimesh(FEMmesh)                  %% display the generated mesh

function res = a(xy,dummy)     res = xy(:,1);     endfunction
function res = Volt(xy,dummy)  res = xy(:,2)>0.1; endfunction

u = BVP2Dsym(FEMmesh,'a',0,0,'Volt',0,0);
figure(2); FEMtrimesh(FEMmesh,u);
           view([38,48]); xlabel('radius r'); ylabel('height z'); zlabel('voltage')
figure(3); FEMtricontour(FEMmesh,u,21);
           xlabel('radius r'); ylabel('height z');

[ux,uy] = FEMEvaluateGradient(FEMmesh,u);
xi = linspace(0,2.5,101)'; yi = zeros(101,1);
uy_i = FEMgriddata(FEMmesh,uy,xi,yi);
figure(4); plot(xi,uy_i)
           xlabel('radius r'); ylabel('u_z'); ylim([-1,6])
Integral = [2*pi*trapz(xi,xi.*uy_i), pi*1^2/0.2]
-->
Integral =  23.782   15.708
```

9.15 Torsion of beams, Prandtl stress function

Examine the torsion of a shaft with constant cross section. Based on a few assumtions determine the deformation of the shaft under torsion. The problem is presented in [VarFEM] and find more details in [Sout73, §12].

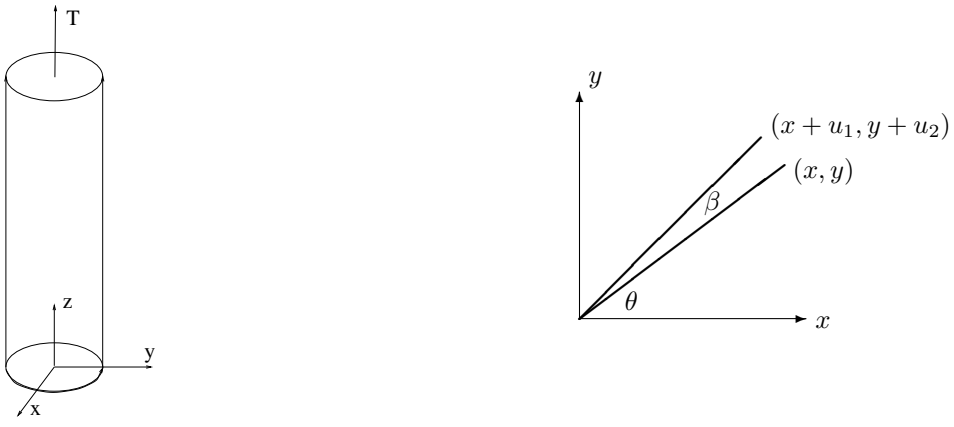


Figure 140: Torsion of a shaft

9.15.1 The setup with the warp function and the Prandtl stress function

Consider a vertical shaft with constant cross section. The centers of gravity of the cross section are along the z axis and the bottom of the shaft is fixed. The top surface is twisted by a total torque T. The situation of a circular cross section is shown in Figure 140. There is no exact specification of the forces and twisting moments applied to the two ends. Based on Saint-Venant principle (see [Sout73, §5.6]) assume that the stress distribution in the cross sections does not depend on z, except very close to the two ends. The twisting leads to a rotation of each cross section by an angle β where $\beta = z \cdot \alpha$. The constant α is a measure of the change of angle per unit length of the shaft. Its value α has to be determined, using the moment T. Based on this determine the horizontal displacements for small angles β by the right part of Figure 140 and a linear approximation

$$\begin{aligned} u_1(x,y) &= r\,\cos(\beta+\theta) - r\,\cos(\theta) \approx -\beta\, r\,\sin\theta = -y\,\beta = -y\,z\,\alpha \\ u_2(x,y) &= r\,\sin(\beta+\theta) - r\,\sin(\theta) \approx +\beta\, r\,\cos\theta = +x\,\beta = +x\,z\,\alpha \end{aligned} \quad .$$

It is assumed that the vertical displacement is independent of z and given by a warping function $\phi(x,y)$. This leads to the displacements

$$u_1 = -y\,z\,\alpha \quad , \quad u_2 = x\,z\,\alpha \quad , \quad u_3 = \alpha\,\phi(x,y)$$

and thus the strain components

$$\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{zz} = \varepsilon_{xy} = 0 \quad , \quad \varepsilon_{xz} = -\frac{1}{2}\,\alpha\,y + \frac{1}{2}\,\alpha\,\frac{\partial\,\phi}{\partial x} \quad , \quad \varepsilon_{yz} = \frac{1}{2}\,\alpha\,x + \frac{1}{2}\,\alpha\,\frac{\partial\,\phi}{\partial y}\,.$$

Using Hooke's law find the stress components

$$\sigma_x = \sigma_y = \sigma_z = \tau_{xy} = 0 \quad , \quad \tau_{xz} = \frac{E\,\alpha}{2\,(1+\nu)}\,(-y + \frac{\partial\,\phi}{\partial x}) \quad , \quad \tau_{yz} = \frac{E\,\alpha}{2\,(1+\nu)}\,(x + \frac{\partial\,\phi}{\partial y})\,.$$

The problem is neither plane stress ($\tau_{xz} \neq 0$, $\tau_{yz} \neq 0$) nor plane strain ($\phi \neq 0$). Using the stresses determine the horizontal forces and the torsion along a hypothetical horizontal cross section. Since the origin is the center of gravity of the cross section Ω the first moments vanish and

$$\begin{aligned} T &= \iint\limits_{\Omega} x\,\tau_{yz} - y\,\tau_{yz}\,dA = \frac{E\,\alpha}{2\,(1+\nu)} \iint\limits_{\Omega} x\,(x + \frac{\partial\,\phi}{\partial y}) - y\,(-y + \frac{\partial\,\phi}{\partial x})\,dA \\ &= \frac{E\,\alpha}{2\,(1+\nu)} \iint\limits_{\Omega} x^2 + y^2 + x\,\frac{\partial\,\phi}{\partial y} - y\,\frac{\partial\,\phi}{\partial x}\,dA = \frac{E\,\alpha}{1+\nu}\,J\,. \end{aligned}$$

Using the **torsional rigidity** J with

$$J = \iint\limits_{\Omega} x^2 + y^2 + x\,\frac{\partial\,\phi}{\partial y} - y\,\frac{\partial\,\phi}{\partial x}\,dA$$

determine the constant α by

$$\alpha = \frac{2\,(1+\nu)}{J\,E}\,T$$

and thus for a shaft of height H the total change of angle β as

$$\beta = H \cdot \alpha = \frac{2\,(1+\nu)}{J\,E}\,H \cdot T\,.$$

The only difficult part is to determine the function ϕ, then J is determined by an integration.

The above computations allow to compute the energy E in one cross section Ω by

$$\begin{aligned} E &= \iint\limits_{\Omega} \sigma_{xz}\,\tau_{xz} + \sigma_{yz}\,\tau_{yz}\,dA = \frac{E\,\alpha^2}{4\,(1+\nu)} \iint\limits_{\Omega} (-y + \frac{\partial\,\phi}{\partial x})^2 + (x + \frac{\partial\,\phi}{\partial y})^2\,dA \\ &= \frac{E\,\alpha^2}{4\,(1+\nu)} \iint\limits_{\Omega} (\frac{\partial\,\phi}{\partial x})^2 + (\frac{\partial\,\phi}{\partial y})^2 - 2\,y\,\frac{\partial\,\phi}{\partial x} + 2\,x\,\frac{\partial\,\phi}{\partial y} + x^2 + y^2\,dA\,. \end{aligned}$$

The warp function ϕ has to minimize this expression. Using calculus of variations (e.g. [VarFEM]) one can show that ϕ has to solve the boundary value problem

$$\begin{aligned} \operatorname{div}(\nabla\phi) = \Delta\phi &= 0 && \text{in the cross section} \quad \Omega \\ \vec{n}\cdot\nabla\phi &= \begin{pmatrix} y \\ -x \end{pmatrix}\cdot\vec{n} && \text{on the boundary} \quad \partial\Omega \end{aligned} \quad . \tag{120}$$

Since the stress components are given by

$$\sigma_x = \sigma_y = \sigma_z = \tau_{xy} = 0 \quad , \quad \tau_{xz} = \frac{E\,\alpha}{2\,(1+\nu)}\,(-y + \frac{\partial\,\phi}{\partial x}) \quad , \quad \tau_{yz} = \frac{E\,\alpha}{2\,(1+\nu)}\,(x + \frac{\partial\,\phi}{\partial y})$$

the boundary condition can be written as

$$\begin{pmatrix} \tau_{xz} \\ \tau_{yz} \end{pmatrix}\cdot\vec{n} = 0\,.$$

This equation implies that there is no stress on the lateral surface of the shaft. This condition is consistent with the mechanical setup.

The Prandtl stress funktion χ is characterized by

$$\frac{\partial\chi}{\partial y} = -y + \frac{\partial\phi}{\partial x} = \frac{2\,(1+\nu)}{E\,\alpha}\,\tau_{xz} \quad \text{and} \quad -\frac{\partial\chi}{\partial x} = x + \frac{\partial\phi}{\partial y} = \frac{2\,(1+\nu)}{E\,\alpha}\,\tau_{yz}\,.$$

By differentiating the above equations by y (resp. x) and subtracting and using $\frac{\partial}{\partial x}\frac{\partial\,\phi}{\partial y} = \frac{\partial}{\partial y}\frac{\partial\,\phi}{\partial x}$ find

$$\Delta\chi = \frac{\partial^2\chi}{\partial x^2} + \frac{\partial^2\chi}{\partial y^2} = -2\,.$$

To determine the boundary conditions for χ assume that there are no external forces on the boundary.

$$\begin{pmatrix} \tau_{xz} \\ \tau_{yz} \end{pmatrix}\cdot\vec{n} = 0 \qquad \Longrightarrow \qquad \begin{pmatrix} \frac{\partial\chi}{\partial y} \\ -\frac{\partial\chi}{\partial x} \end{pmatrix}\cdot\vec{n} = \nabla\chi\cdot\vec{t} = 0\,,$$

where $\vec{t}$ is a tangential vector of the boundary curve. Assuming that there are no holes[42], this implies that one can work with $\chi = 0$ on the boundary Γ. Thus the Pandtl stress function is a solution of the boundary value problem

$$\begin{aligned} -\Delta\chi &= 2 && \text{in} \quad \Omega \\ \chi &= 0 && \text{on} \quad \Gamma \end{aligned} \quad . \tag{121}$$

The torsional rigidity is determined by

$$J = \iint_\Omega x^2 + y^2 + x\,(-\frac{\partial\,\chi}{\partial x} - x) - y\,(+\frac{\partial\,\chi}{\partial y} + y)\;dA = -\iint_\Omega x\,\frac{\partial\,\chi}{\partial x} + y\,\frac{\partial\,\chi}{\partial y}\;dA\,.$$

For ductile materials the von Mises stress indicates the possible fractures in the material. In this case it is given by

$$\sigma_{vM} = \sqrt{\frac{3}{2}\,(\tau_{xz}^2 + \tau_{yz}^2)} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\sqrt{(\frac{\partial\,\chi}{\partial x})^2 + (\frac{\partial\,\chi}{\partial y})^2} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\|\nabla\chi\|\,.$$

9.15.2 On a disk with radius R

On a disk with radius R the solution is given by $\chi(x,y) = \frac{1}{2}\,(R^2 - x^2 - y^2)$. Thus the nonzero stresses are

$$\tau_{xz} = +\frac{E\,\alpha}{2\,(1+\nu)}\,\frac{\partial\,\chi}{\partial y} = -\frac{E\,\alpha}{2\,(1+\nu)}\,y \quad \text{and} \quad \tau_{yz} = -\frac{E\,\alpha}{2\,(1+\nu)}\,\frac{\partial\,\chi}{\partial x} = +\frac{E\,\alpha}{2\,(1+\nu)}\,x\,.$$

[42] This restriction can be removed.

The BVP (120) for the warp function ϕ is

$$\begin{aligned} \operatorname{div}(\nabla\phi) = \Delta\phi &= 0 && \text{in the cross section} \quad \Omega \\ \vec{n}\cdot\nabla\phi &= \frac{1}{\sqrt{x^2+y^2}} \begin{pmatrix} y \\ -x \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = 0 && \text{on the boundary} \quad \partial\Omega \end{aligned}$$

with the unique solution $\phi(x,y) = 0$, i.e. no warping. The torsional rigidity is given by

$$J = \iint\limits_{\Omega} x^2 + y^2 \, dA = 2\,\pi \int_0 r^2\, r \, dr = \frac{\pi}{2} R^4$$

and the von Mises stress is given by

$$\sigma_{vM} = \frac{E\,\alpha}{2\,(1+\nu)} \sqrt{\frac{3}{2}} \sqrt{(\frac{\partial\,\chi}{\partial x})^2 + (\frac{\partial\,\chi}{\partial y})^2} = \frac{E\,\alpha}{2\,(1+\nu)} \sqrt{\frac{3}{2}}\, \sqrt{x^2+y^2} = \frac{E\,\alpha}{2\,(1+\nu)} \sqrt{\frac{3}{2}}\, r\,.$$

9.15.3 On a square

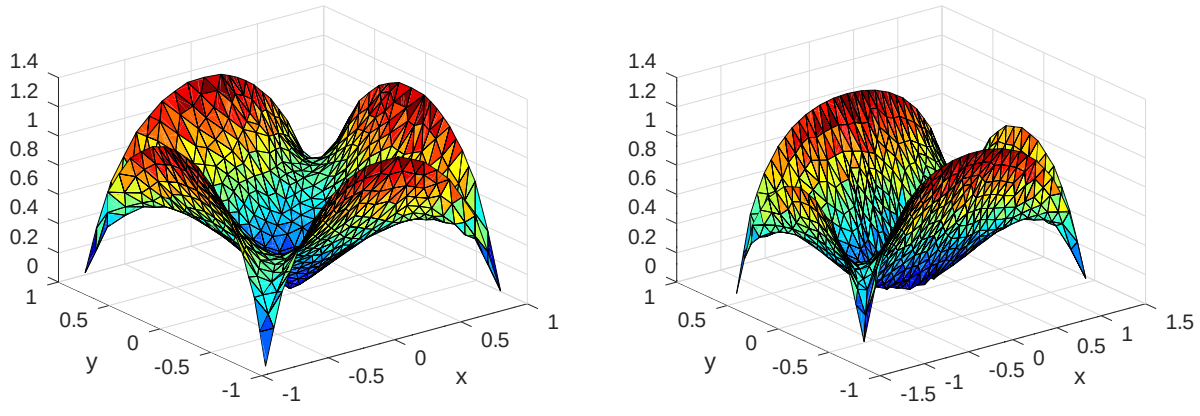
To examine the stiffness of a square cross section with a circular cross section examine a square with the same area as a circle with radius $R = 1$. Thus the length of a side is $\sqrt{\pi} \approx 1.77$. The code below solves the boundary value problem (121) and then computes the torsional rigidity by integrating

$$J = -\iint\limits_{\Omega} x\,\frac{\partial\,\chi}{\partial x} + y\,\frac{\partial\,\chi}{\partial y}\, dA\,.$$

The numerical result of $J \approx 1.39$ has to be compared to the result of $J = \frac{\pi}{2} \approx 1.57$ for the disk with Radius 1. Thus the square cross section leads to less torsional rigidity. Then examine the von Mises stress by plotting

$$f(x,y) = \sqrt{(\frac{\partial\,\chi}{\partial x})^2 + (\frac{\partial\,\chi}{\partial y})^2} = \|\nabla\chi\|\,.$$

Find the result in Figure 141(a). The maximal value of ≈ 1.20 is larger than the maximal value 1 for the disk. Thus for the same twisting angle the square is exposed to a larger von Mises stress.



(a) on a square

(b) on a rectangle

Figure 141: The von Mises stress caused by torsion of a bar with square or rectangular cross section

TorsionSquare.m

```
N = 10; l = sqrt(pi)/2;  al = 1; %%al = sqrt(2);  %% use this for the rectangle
Mesh = CreateMeshTriangle('Torsion',
[-al*l -1/al*l -1; al*l -1/al*l -1; al*l 1/al*l -1; -al*l 1/al*l -1],pi/2/N^2);
```

```
Mesh = MeshUpgrade(Mesh);

chi = BVP2Dsym(Mesh,1,0,2,0,0,0);

[chiGP,gradChi] = FEMEvaluateGP(Mesh,chi);
xGP = Mesh.GP(:,1); yGP = Mesh.GP(:,2);
f = xGP.*gradChi(:,1) + yGP.*gradChi(:,2);
J = FEMIntegrate(Mesh,-f)

[chi_x,chi_y] = FEMEvaluateGradient(Mesh,chi);
Stress = sqrt(chi_x.^2 + chi_y.^2);
figure(1); FEMtrisurf(Mesh,Stress); xlabel('x'); ylabel('y');

MaxStress = max(Stress)
-->
J =  1.3873
```

9.15.4 On a rectangle

The above can be repeated for a rectangle with the same are but a ratio of 2 for the length of the sides. The value of $J \approx 1.13$ indicates that the rectangle is even softer and the maximal von Mises stress of ≈ 1.16 is slightly smaller than for the square cross section.

9.16 Dynamic heat conduction problems

The dynamic heat equation with a thermal conductivity $a(x,y)$ is of the form given in equation (6). For the simplified case with no external heating, no convection and the boundary either insulated or at a given temperature arrive at the initial boundary value problem

$$\begin{aligned} \frac{\partial}{\partial t}\, u - \nabla \cdot (a\, \nabla u) &= 0 \qquad \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\ u &= g \qquad \text{for} \quad (x,y,t) \in \Gamma_1 \times (0,T] \\ \vec{n} \cdot (a\, \nabla u) &= 0 \qquad \text{for} \quad (x,y,t) \in \Gamma_2 \times (0,T] \\ u &= u_0 \qquad \text{on} \quad \Omega \quad \text{at} \quad t = 0 \end{aligned} \quad . \tag{122}$$

In Figure 142 the upper half of the domain is shown, at the lower edge the symmetry constraint $\frac{\partial}{\partial n}\, u = 0$ is used. Assume insulation on all of the boundary, except the left edge Γ_1 at $x = 0$, where the temperature equals 1 . As initial temperature we use $u_0(x,y) = 0$ and observe how the domain is warming up as time advances.

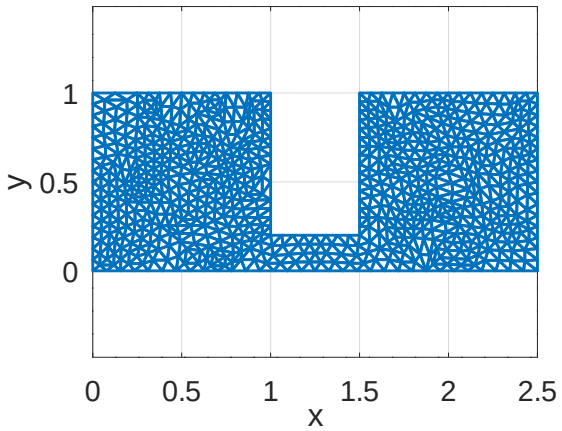
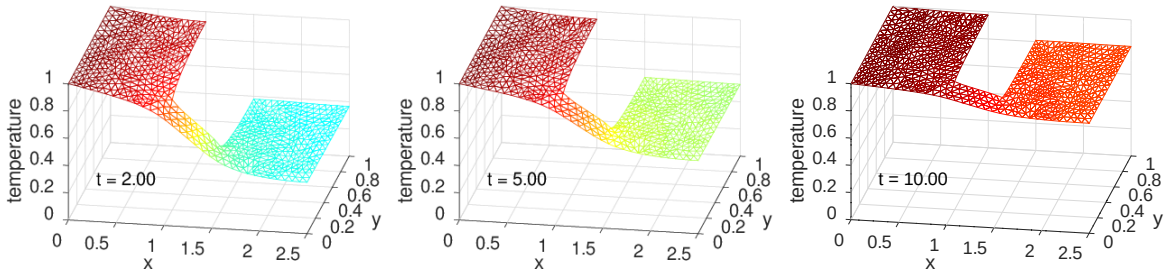


Figure 142: The mesh for a dynamic heat problem

9.16.1 With a narrow section in the domain

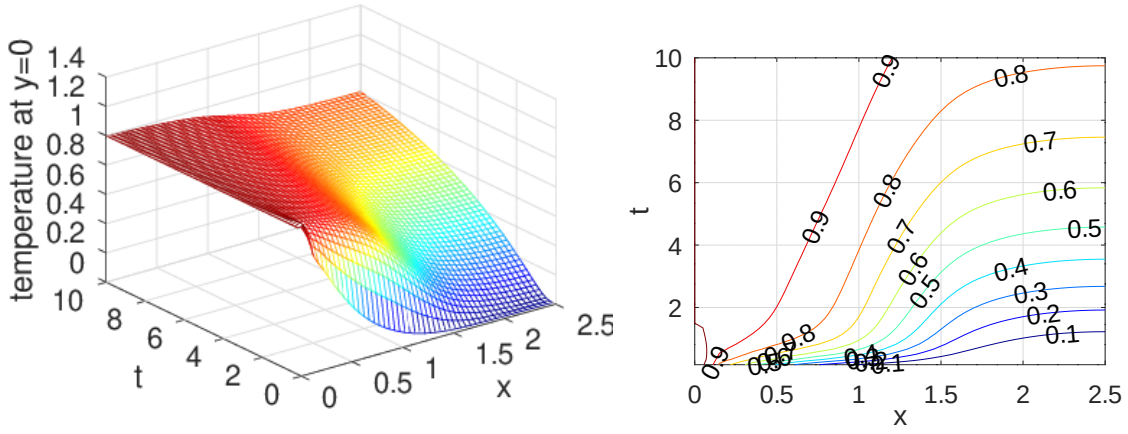
The first case to be examined uses a narrow section between to bigger sections. The dimension of the narrow section can be changed by modifying the parameters $h = 0.2$ and $l = 0.5$.

- In Figure 143 observed the delayed heating of the section on the right.
- In Figure 144 the temperature along the edge $y = 0$ for $0 \le x \le 2.5$ and $0 \le t \le 10$ is shown, as surface and contour lines.
- In Figure 145 the temerature at the corner $(x, y) = (2.5\,,\,0)$ is shown as function of time.



(a) at $t = 2$ (b) at $t = 5$ (c) at $t = 10$

Figure 143: The evolution of the temperature surface at different times



(a) surface along the edge $y = 0$ (b) contours along the edge $y = 0$

Figure 144: The temperature surface at different times along $y = 0$

HeatDynamic.m

```
%% parameters
h = 0.2;  l = 0.5; Nt = 60;  %% number of time steps
FEMmesh = CreateMeshTriangle('Test',...
        [0 0,-2; 2+l 0 -2; 2+l 1,-2; 1+l 1 -2; 1+l h -2; 1 h -2; 1 1 -2; 0 1 -1],0.01);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

figure(1); FEMtrimesh(FEMmesh);
           axis equal; xlabel('x'); ylabel('y')

[u t] = IBVP2D(FEMmesh,1,1,0, 0, 0, 0,1, 0,  0,  0, 0, 10, [Nt,10]);
```

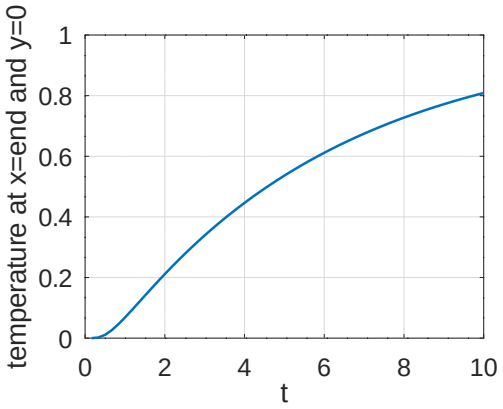


Figure 145: The temperature as function of time at the endpoint $(2.5\,,\,0)$

```
figure(2); FEMtrimesh(FEMmesh,u(:,end))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1]);
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(end))); zlabel('temperature')
figure(3); FEMtrimesh(FEMmesh,u(:,Nt/2+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(Nt/2+1))); zlabel('temperature')
figure(4); FEMtrimesh(FEMmesh,u(:,Nt/3+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(Nt/5+1))); zlabel('temperature')

x = linspace(0,2+l,51);  u_int = zeros(size(t,2)-1,size(x,2));
for jj = 2:size(t,2)
  u_int(jj-1,:) = FEMgriddata(FEMmesh,u(:,jj),x,zeros(size(x)));
endfor

figure(10); mesh(x,t(2:end),u_int)
            xlabel('x'); ylabel('t'); zlabel('temperature at y=0')
figure(11); [c,h] = contour(x,t(2:end),u_int,[0:0.1:1]);
            clabel(c,h);
            xlabel('x'); ylabel('t');
figure(12); plot(t(2:end),u_int(:,end))
            xlabel('t'); ylabel('temperature at x=end and y=0')
```

9.16.2 With a section of lower thermal conductivity

On the modified domain visible in Figure 146 in the middle section the conductivity is considerably smaller than in the two side section, i.e.

$$a(x,y) = \begin{cases} 1 & \text{for } 0 \le x \le 1 \text{ and } x \ge 1.5 \\ \frac{1}{6} & \text{for } 1 < x < 1.5 \end{cases}$$

- In Figure 147 observed the delayed heating of the section on the right.
- In Figure 148 the temperature along the edge $y = 0$ for $0 \le x \le 2.5$ and $0 \le t \le 10$ is shown, as surface and contour lines.
- In Figure 149 the temperature at the corner $(x, y) = (2.5\,,\,0)$ is shown as function of time.

Observe the similar, but not identical, behavior of the two cases examined.

HeatDynamicCoefficient.m

```
%% parameters
h = 1.2;  l = 0.5; Nt = 60;  %% number of time steps
FEMmesh = CreateMeshTriangle('Test',...
    [0 0,-2; 2+l 0 -2; 2+l 1, -2; 1+l 1 -2; 1+l h -2; 1 h -2; 1 1 -2; 0 1 -1],0.01);
```

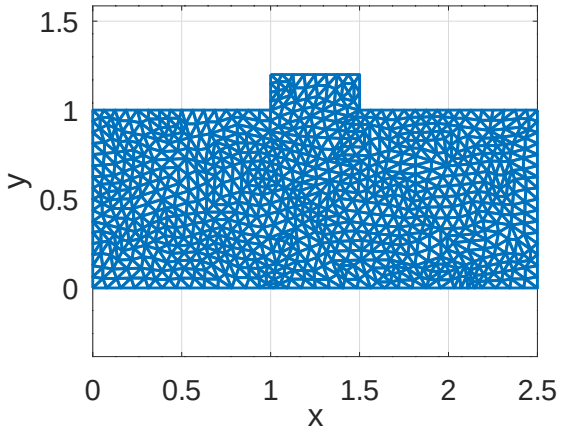
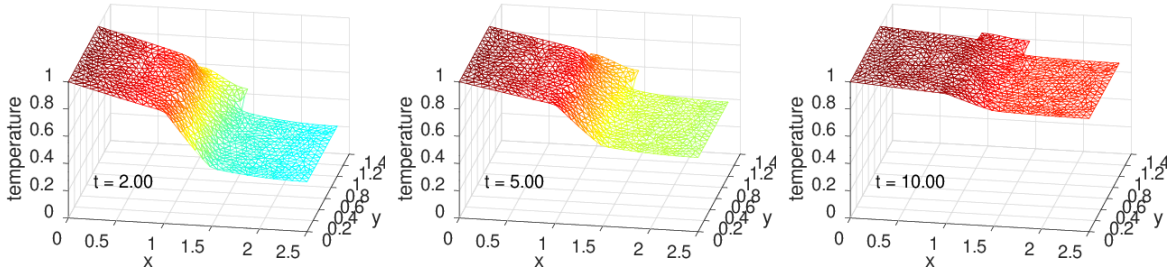
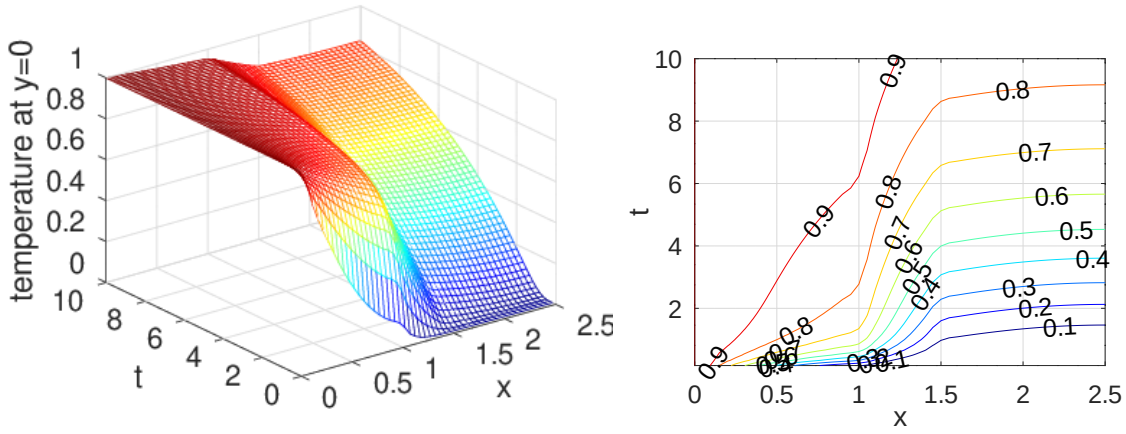


Figure 146: The mesh for a dynamic heat problem



(a) at $t = 2$ (b) at $t = 5$ (c) at $t = 10$

Figure 147: The evolution of the temperature surface at different times



(a) surface along the edge $y = 0$ (b) contours along the edge $y = 0$

Figure 148: The temperature surface at different times along $y = 0$

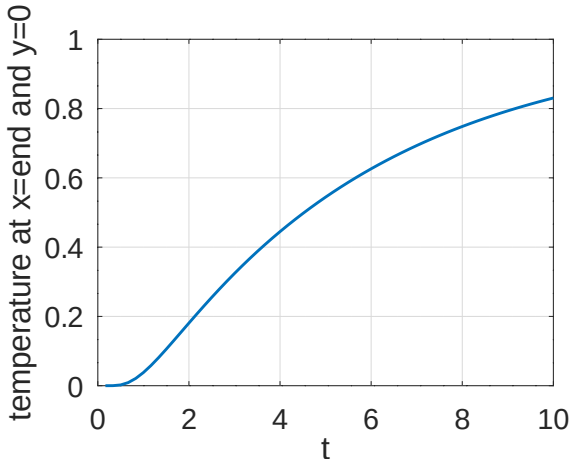


Figure 149: The temperature as function of time at the endpoint $(2.5\,,0)$

```
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
%%FEMmesh = MeshUpgrade(FEMmesh,'cubic');

figure(1); FEMtrimesh(FEMmesh);
           axis equal; xlabel('x'); ylabel('y')

function res = a(xy,dummy)
  l = 0.5;
  res = ones(size(xy,1),1);
  res(find(abs(xy(:,1)-1-l/2)<l/2)) *= 1/6;
endfunction

[u t] = IBVP2D(FEMmesh,1,'a',0, 0, 0, 0,1, 0,  0,  0, 0, 10, [Nt,10]);

figure(2); FEMtrimesh(FEMmesh,u(:,end))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1]);
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(end))); zlabel('temperature')
figure(3); FEMtrimesh(FEMmesh,u(:,Nt/2+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(Nt/2+1))); zlabel('temperature')
figure(4); FEMtrimesh(FEMmesh,u(:,Nt/3+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(0.2,0.2,0.2,sprintf('t = %4.2f',t(Nt/5+1))); zlabel('temperature')

x = linspace(0,2+l,51);  u_int = zeros(size(t,2)-1,size(x,2));
for jj = 2:size(t,2)
  u_int(jj-1,:) = FEMgriddata(FEMmesh,u(:,jj),x,zeros(size(x)));
endfor

figure(10); mesh(x,t(2:end),u_int)
            xlabel('x'); ylabel('t'); zlabel('temperature at y=0')

figure(11); [c,h] = contour(x,t(2:end),u_int,[0:0.1:1]);
            clabel(c,h);
            xlabel('x'); ylabel('t');

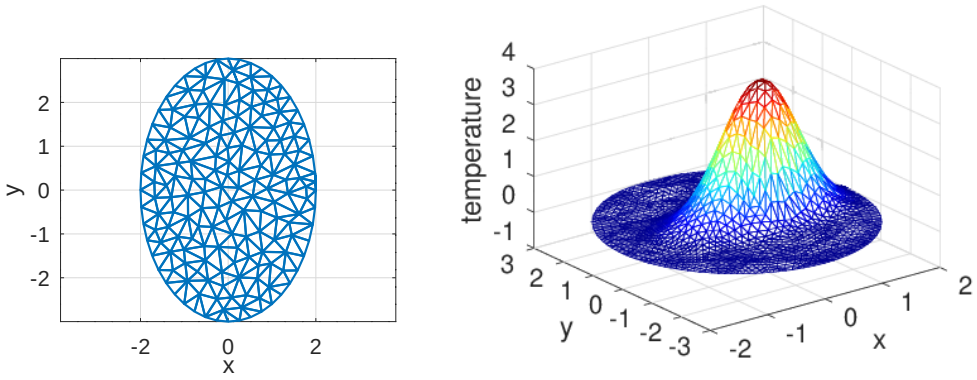
figure(12); plot(t(2:end),u_int(:,end))
            xlabel('t'); ylabel('temperature at x=end and y=0')
```

9.16.3 Cooling of a cylinder

Examine a cylinder with elliptical cross section and an initial temperature distribution $u_0(x, y)$, independent on z. The boundary temperature is fixed at 0 . The domain and the initial temperature profile are visible in Figure 150. The selected, nonsymmetric initial temperature is

$$u_0(x, y) = \exp(-(x - 0.5)^2 - 2\,y^2) \cdot (4 - x^2 - y^2)\ .$$

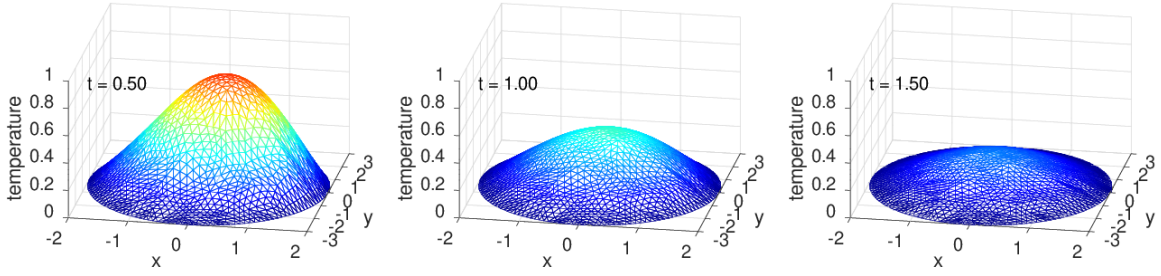
The initial boundary value problem is solved for times $0 \le t \le 2$. A few snapshots are visible in Figure 151. By



(a) the elliptical domain (b) the initial temperature

Figure 150: The domain and the initial temperature

looking at different time slices an animation can be generated.



(a) at $t = 0.5$ (b) at $t = 1.0$ (c) at $t = 1.5$

Figure 151: The temperature at different times

CylinderCooling.m

```
R = 2;  N = 50; alpha = linspace(0,2*pi*N/(N-1),N)';
Tend = 2; Nt = 60; %% number of shown time steps
FEMmesh = CreateMeshTriangle('circle',[R*cos(alpha),1.5*R*sin(alpha),...
                              -ones(size(alpha))],0.1);

figure(1); FEMtrimesh(FEMmesh)
           xlabel('x') ; ylabel('y'); axis equal
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = u_init(xy)
  x = xy(:,1); y = xy(:,2);
  res = exp(-(x-0.5).^2-2*y.^2) .* (2^2 -x.^2-y.^2);
endfunction
```

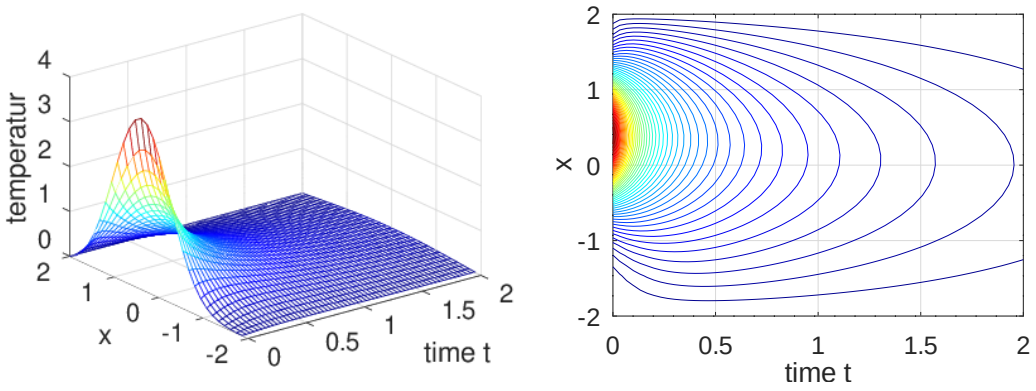
```
figure(2); FEMtrimesh(FEMmesh,u_init(FEMmesh.nodes))
           xlabel('x'); ylabel('y'); zlabel('temperature');

[u,t] = IBVP2Dsym(FEMmesh,1,1,0, 0, 0,  0,  0, 'u_init',0, Tend, [Nt,10]);

figure(3); FEMtrimesh(FEMmesh,u(:,Nt/4+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(-1.8,-2,0.9,sprintf('t = %4.2f',t(Nt/4+1))); zlabel('temperature')
figure(4); FEMtrimesh(FEMmesh,u(:,Nt/2+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
           text(-1.8,-2,0.9,sprintf('t = %4.2f',t(Nt/2+1))); zlabel('temperature')
figure(5); FEMtrimesh(FEMmesh,u(:,3*Nt/4+1))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1]);
           text(-1.8,-2,0.9,sprintf('t = %4.2f',t(3*Nt/4+1))); zlabel('temperature')
```

```
figure(11) %% show the animation
steps = 2;
for jj = 0:30
  FEMtrimesh(FEMmesh,u(:,jj*steps+1))
  text(-1.8,-2,0.9,sprintf('t = %4.2f',t(jj*steps+1))); zlabel('temperature')
  xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1])
  pause(0.2)
endfor
```

Obviously the temperature is decaying as time advances. To examine this behavior determine the tempetarures along the center line at $y = 0$, as function of time. In Figure 152.



(a) surface (b) contour lines

Figure 152: The temperature at different times along $y = 0$

```
x = linspace(-R,R,31);  u_center = zeros(length(x),length(t));
for jj = 1:length(t)
  u_center(:,jj) = FEMgriddata(FEMmesh,u(:,jj),x,zeros(size(x)));
endfor
figure(21); mesh(t,x,u_center)
            xlabel('time t'); ylabel('x'); zlabel('temperatur')
figure(22); contour(t,x,u_center,51)
            xlabel('time t'); ylabel('x');
```

The decay of the temperature at the center point $(0, 0)$ is visible in Figure 153, with linear and logarithmic scale. The exponential decay is clearly displayed in the logarithmic scale. This is consistent with the theoretical result

$$u(t, x, y) = \sum_{n=1}^{\infty} c_n \, e^{-\lambda_n t} \, u_n(x, y) \tag{123}$$

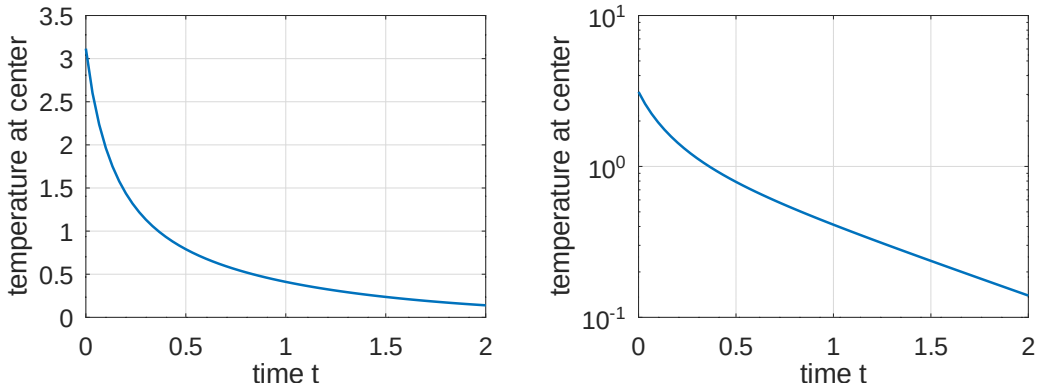
where $\lambda_1 < \lambda_2 \leq \lambda_3 \leq \lambda_4 ...$ and $u_n(x,y)$ are the eigenvalues and eigenfunctions of the boundary value problem

$$\begin{aligned} -\nabla \cdot \nabla u_n &= \lambda_n\, u \qquad \text{for} \quad (x,y) \in \Omega \\ u &= 0 \qquad \text{for} \quad (x,y) \in \Gamma \end{aligned} \quad .$$

For large times t in equation (123) the first eigenvalue will dominate, i.e.

$$u(t,x,y) \approx c_1\, e^{-\lambda_1\, t}\, u_1(x,y) \; .$$

Using the *Octave* command `polyfit()` with data from the right section in the logarithmic plot in Figure 153



(a) linear scale (b) logarithmic scale

Figure 153: The temperature decay at the center $(0,0)$

estimate the decay by the exponential $\exp(-1.06\,t)$. Using `BVP2Deig()` the exponent is estimated by $\lambda_1 \approx 1.04$, i.e. rather close to the above result by `polyfit()`, which indicates

$$\log(u(0,0,t)) \approx 0.1556 - 1.0638\,t \quad \text{or} \quad u(0,0,t) \approx 1.1684\, e^{-1.0638\,t} \quad \text{for } t \text{ large.}$$

```
figure(23); plot(t,u_center(16,:))
            xlabel('time t'); ylabel('temperature at center')
figure(24); semilogy(t,u_center(16,:))
            xlabel('time t'); ylabel('temperature at center')

p = polyfit(t(40:end),log(u_center(16,40:end)),1)
EigVal = BVP2Deig(FEMmesh,1,0,1,0,3)'
-->
p      =   -1.0638    0.1556
EigVal =    1.0425    2.1314   3.1506
```

Observe that $\lambda_2 \neq \lambda_3$, since the domain is not circular. If the above computations are rerun on a circle of radius $R = 2$ obtain $\lambda_1 \approx 1.45$ and $\lambda_2 = \lambda_3 \approx 3.68$. The first eigenvalue $\lambda_1 \approx 1.45$ is larger, thus the cylinder will cool down faster and the second and third eigenvalues coincide, caused by the circular symmetry of the domain.

9.16.4 Heat waves

In Figure 154 a domain $\Omega \subset \mathbb{R}^2$ is visible. The heat equation (a special case of the IBVP (6)) to be solved is

$$\begin{aligned} \tfrac{\partial}{\partial t}\, u(x,y,t) - \Delta\, u(x,y,t) &= f(x,y,t) \qquad \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\ \tfrac{\partial}{\partial n}\, u(x,y,t) &= 0 \qquad \text{for} \quad (x,y,t) \in \Gamma \times (0,T] \\ u(x,y,0) &= 0 \qquad \text{on} \quad \Omega \end{aligned} \quad .$$

The function $f(x,y,t)$ equals $\cos(0.5\,\pi\,t)$ for $x \leq -0.9$ and zero otherwise. Thus there is a periodic excitation with period 5 at the very left end of the appendix for $-1 \leq x \leq -0.9$.

The solution is generated by the command `IBVP2D()` and then evaluated along the slice at height $y = 1$ for different values of the time t, using `FEMgriddata()`. Find the result in Figure 155.

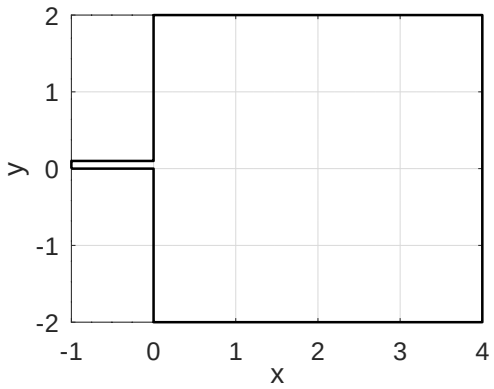
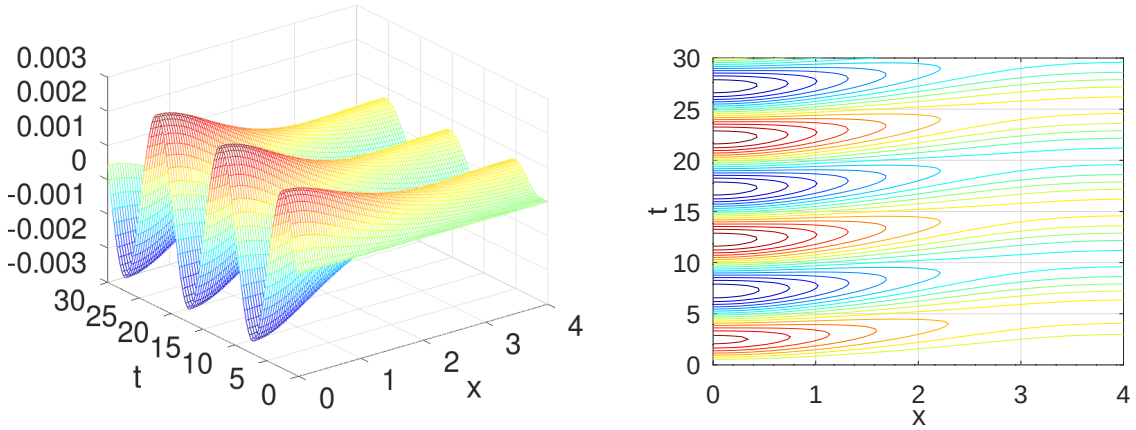


Figure 154: The domain for a heat wave propagation

- In Figure 155(a) the periodic behavior of the temperature is clearly visible.
- In Figure 155(b) observe the phase shift as one moves away from the heat source.

Observe that the behavior of the solution is very different from a wave equation in Section 9.17, even if the setup is comparable.



(a) the surface (b) the contour lines

Figure 155: The propagation of a heat wave

HeatWave.m

```
l = 1; h = 0.1; L = 4; d = 2; H = 2;
FEMmesh = CreateMeshTriangle('test',...
           [-l,0,-2;0 0 -2;0,-d,-2;L,-d,-2; L,H,-2;0,H,-2;0,h,-2;-l,h,-2],0.01);
figure(1); FEMtrimesh(FEMmesh)
           xlabel('x'); ylabel('y'); axis equal
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = f(xy,t)
  res = cos(0.2*pi*t)*ones(size(xy,1),1);
  res(xy(:,1)>-0.9) = 0;
endfunction

function res = u0(xy)   res = zeros(size(xy,1),1); endfunction

m = 1; a = 1; b0 = 0; bx = by = 0; f = 0; gn1 = gn2 = 0;
```

```
tic();
[u,t] = IBVP2D(FEMmesh,m,a,b0,bx,by,'f',0,gn1,gn2,'u0',0,30,[2*60,10]);
%%[u,t] = IBVP2Dsym(FEMmesh,m,a,b0,'f',0,gn1,gn2,'u0',0,30,[2*60,10]);
SolverTime = toc()

figure(2); FEMtrimesh(FEMmesh,u(:,end))
           xlabel('x'); ylabel('y'); xlim([0,L]);

umax = 0.3*max([-min(u(:)),max(u(:))]);
figure(3)
if 0  %% animation
  for jj = 1:length(t)
    FEMtrimesh(FEMmesh,u(:,jj))
    xlabel('x'); ylabel('y')
    zlim(umax*[-1 1]); caxis(0.3*umax*[-1 1]);
    text(0.8*L,0.8*H,umax,sprintf('t = %4.2f',t(jj)))
    xlim([0,L])
    pause(0.1);
  endfor
else
  FEMtrimesh(FEMmesh,u(:,end))
  xlabel('x'); ylabel('y')
  zlim(umax*[-1 1]); caxis(0.3*umax*[-1 1]);
  text(0.8*L,0.8*H,umax,sprintf('t = %4.2f',t(end)))
endif

x = linspace(0,L,101); u_line = zeros(size(t,1),size(x,2));
for jj = 1:length(t)
  u_line(jj,:) = FEMgriddata(FEMmesh,u(:,jj),x,ones(size(x)));
endfor

figure(4); mesh(x,t,u_line)
           xlabel('x'); ylabel('t');
figure(5); contour(x,t,u_line,0.003*[-1:0.1:+1])
           xlabel('x'); ylabel('t');
```

9.16.5 Static heat equation in a ball in $\mathbb{R}^3$, solved as a 1D problem

The dynamic heat equation in spherical coordinates (r, θ, φ) is given by

$$\frac{\rho\, c}{k}\,\frac{\partial}{\partial t}\,u = +\Delta\, u + f = \frac{1}{r^2}\,\frac{\partial}{\partial r}\,(r^2\,\frac{\partial\, u}{\partial r}) + \frac{1}{r^2\,\sin\theta}\,\frac{\partial}{\partial\theta}\,(\sin\theta\,\frac{\partial\, u}{\partial\theta}) + \frac{1}{r^2\sin^2\theta}\,\frac{\partial^2\, u}{\partial\varphi^2} + f\ .$$

Examining a static problem, depending on the radius r only leads to the ordinary differential equation

$$-\frac{\partial}{\partial r}\,(r^2\frac{\partial\, u}{\partial r}) = r^2 f(t,r)\ .$$

At first sight[43] FEMoctave is not set up to solve problems depending on one variable only, but one may construct solutions whose values do not depend on the second variable, i.e. transform a 1D problem artificially to a 2D problem. It has to be pointed out that this is not computationally efficient. Find the efficient 1D approach at the end of the section.

The BVP $-\frac{\partial}{\partial r}\,(r^2\frac{\partial\, u}{\partial r}) = 0$ with $u(1) = 0$ and $u(R) = 1$ is solved by[44]

$$u(r) = \frac{R}{R-1}\,\frac{r-1}{r}\ .$$

With FEMoctave the solution can be solved by the code below.

[43] The command `IBVP1D()` will solve this problem.

[44] The ODE $\frac{\partial}{\partial r}\,(r^2\,\frac{\partial\, u}{\partial r}) = 0$ implies $r^2\,\frac{\partial\, u}{\partial r} = c_1$, i.e. $\frac{\partial}{\partial r}\,u = \frac{c_1}{r^2}$. An integration leads to the general solution of the ODE $u(r) = c_2 - \frac{c_1}{r}$. Then use the two boundary conditions to determine $c_1 = c_2 = \frac{R}{R-1}$ and $u(r) = \frac{R}{R-1}\,\frac{r-1}{r}$.

- Define the parameters of the system.
- Generate the mesh for $1 \le r \le R = 3$ and the dummy variable $-\frac{W}{2} \le y \le +\frac{W}{2}$. At the upper and lower edge at $y = \pm\frac{W}{2}$ apply the Neumann boundary conditions $\frac{\partial}{\partial n} u = 0$. This assures that the solution will depend on r only. At $r = 1$ and $r = R$ specify the Dirichlet conditions.
- Define the functions for the parameter $a(r) = r^2$ and the Dirichlet conditions $u(1) = 0$ and $u(R) = 1$.
- Solve the BVP with the help of `BVP2Dsym()`.
- Display the solutions. To obtain the graphs with one independent variable r use `FEMgriddata()` along the axis $y = 0$ and $1 \le r \le R = 3$.
- In Figure 156 find the graphs of the solution u, either as function of r and y, or as function of r only.

HeatBall.m

```
%% solve a static heat problem in a ball, using the radius r only as variable
R = 3; W = 0.1; N = 10;
FEMmesh = CreateMeshRect(linspace(1,R,N+1),[-W/2,+W/2],-2,-2,-1,-1);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
%%FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = a(r) ;  res = r(:,1).^2;      endfunction
function res = gD(r);  res = sign(r(:,1)-1); endfunction

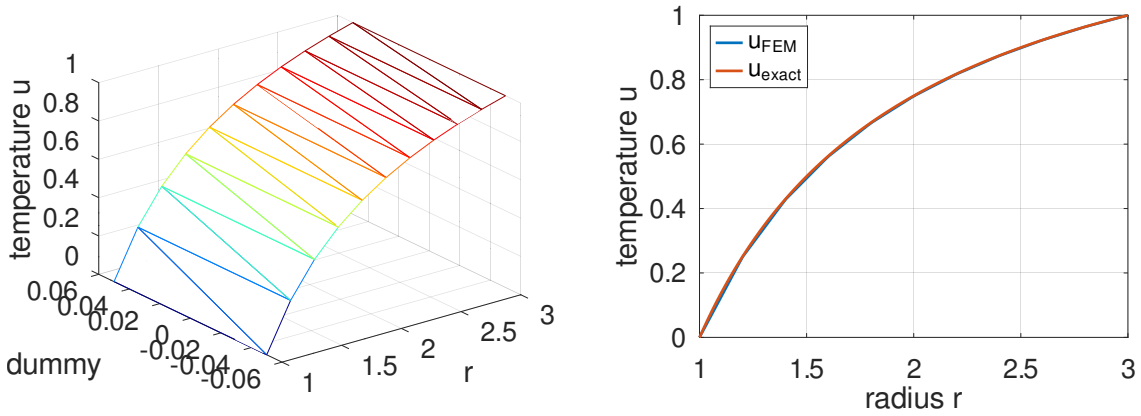
u3D = BVP2Dsym(FEMmesh,'a',0,0,'gD',0,0);

figure(1); FEMtrimesh(FEMmesh,u3D)
           xlabel('r'); ylabel('dummy'); zlabel('temperature u')

r = linspace(1,R); u = FEMgriddata(FEMmesh,u3D,r,0*r);
u_exact = R/(R-1)*(r-1)./r;

figure(2); plot(r,u,r,u_exact)
            xlabel('radius r'); ylabel('temperature u')
            legend('u_{FEM}','u_{exact}','location','northwest')

figure(3); plot(r,u-u_exact)
            xlabel('radius r'); ylabel('u_{FEM}-u_{exact}')
```

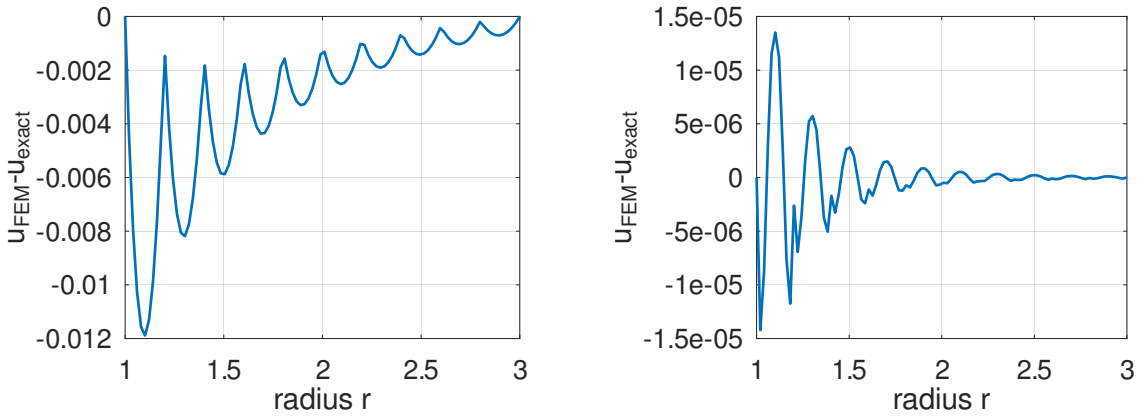


(a) as function of r and the dummy variable

(b) as function of the radius r only

Figure 156: The steady state solution of a heat problem in a ball

For the results in Figure 156 linear elements are used. A simple call of `MeshUpgrade()` allows to use cubic elements. In Figure 157 find the differences to the exact solution for linear and cubic elements. Observe that the error for cubic elements is considerably smaller.



(a) with linear elements (b) with cubic elements

Figure 157: The errors of the steady state solution of a heat problem in a ball

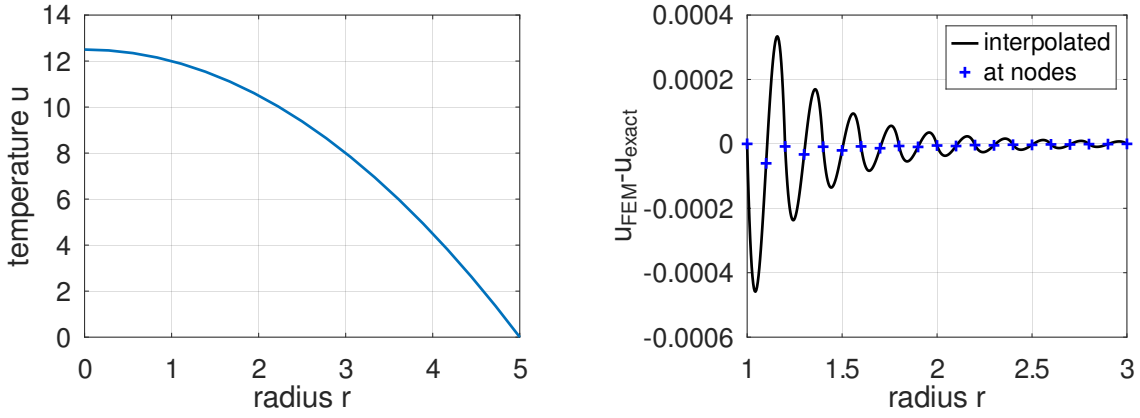
The above is a problem with one space variable only. Thus it is advisable to use `BVP1D()`. Find the results of the code below in Figure 158. Observe the effect of superconvergence

HeatBallRadial.m

```
R1 = 1; R2 = 3; BCleft = 0; BCright = 1;
interval = linspace(R1,R2,11);
[r,u] = BVP1D(interval,@(r)-r.^2,0,0,1,0,BCleft,BCright);
figure(4); plot(r,u,r,R2/(R2-R1)*(r-R1)./r)
           xlabel('radius r'); ylabel( 'temperature u')
           legend('u_{FEM}','u_{exact}','location','northwest')

figure(5); plot(r,u-R2/(R2-R1)*(r-R1)./r,'+-')
           xlabel('radius r'); ylabel( 'temperature u')
           legend('difference','location','southeast')

r_fine = linspace(R1,R2,501)'; u_fine = pwquadinterp(r,u,r_fine);
u_exact = R2/(R2-R1)*(r_fine-R1)./r_fine;
figure(6); plot(r_fine,u_fine-u_exact,'k',r,u-R2/(R2-R1)*(r-R1)./r,'b+')
           xlabel('radius r'); ylabel('u_{FEM}-u_{exact}')
           legend('interpolated','at nodes')
```



(a) the solution (b) difference to true solution

Figure 158: The steady state solution of a heat problem in a ball with second order elements, solved as true 1D problem

9.16.6 Dynamic heat equation in a cylinder, solved as a 1D problem

Similar to Section 9.16.5 the dynamic heat equation in cylindical coordinates (r, θ, z) is given by

$$\frac{\rho\, c}{k}\,\frac{\partial}{\partial t}\,u = +\Delta\, u + f = \frac{1}{r}\,\frac{\partial}{\partial r}\,(r\,\frac{\partial\, u}{\partial r}) + \frac{1}{r^2}\,\frac{\partial^2\, u}{\partial \theta^2} + \frac{\partial^2\, u}{\partial z^2} + f\ .$$

Assuming that the solution depends on time t and radius r only and some rescaling leads to the dynamic equation

$$r\,\frac{\partial}{\partial t}\,u(r,t) = \frac{\partial}{\partial r}\,(r\,\frac{\partial\, u(r,t)}{\partial r}) + r\, f(r,t)\ .$$

Examine a cylinder of radius R and apply time dependent heating in the inner section $r < \frac{R}{2}$ of the cylinder, e.g.

$$f(r,t) = \begin{cases} \sin(t) & \text{for} \quad 0 \le r < \frac{R}{2} \\ 0 & \text{for} \quad \frac{R}{2} \le r \le R \end{cases}\ .$$

HeatCylinder.m

```
R = 3; BCleft = [0,0]; BCright = [0,0];
f = @(r,t)sin(1*t)*(r<R/2);
u0 = 0;  t0 = 0; t_end = 6*pi;  steps = [121,10];  interval = linspace(0,R,21);
r = @(r)r; solver = 'RK';
[r,u,t] = IBVP1D(interval,r,r,0,0,r,f,BCleft,BCright,u0,t0,t_end,steps,'solver',solver);
figure(1); mesh(t,r,u)
           xlabel('time t'); ylabel('radius r'); zlabel('temperature u')
figure(2); contour(t,r,u,[-0.5:0.1:+0.5])
           xlabel('time t'); ylabel('radius r');
figure(3); plot(t,u(1,:),t,u(end,:))
           xlabel('time t'); ylabel('temperature u'); xlim([0,max(t)])
           legend('at center r=0','at outer edge r=R','location','south')
```

As results obtain the temperature u as function of time t and radius r, visible in Figures 159 and 160. The periodic oscillations of U are clearly visible. In Figures 160 and 161 observe the smaller amplitudes of u with respect to time at the outer edge $r = R$ and the phase shift.

With the above code `HeatCylinder.m` it is easy to examine the effects of a different frequency or amplitude in $k\,\sin(\omega\, t)$.

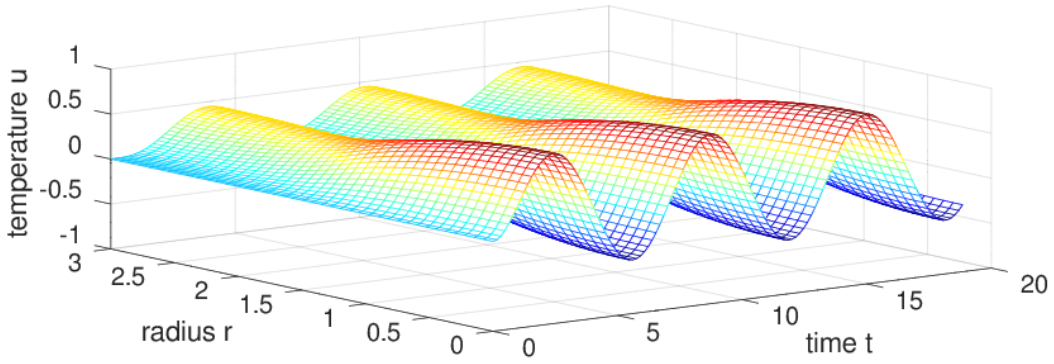


Figure 159: The temperatur $u(r,t)$ inside the cylinder

9.17 Wave propagation, Kirchhoff diffraction

In Figure 162 half of a domain $\Omega \subset \mathbb{R}^2$ is visible, the lower half is generated by a reflection at the lower edge. For the computation this is taken into account by the symmetry boundary condition $\frac{\partial\, u}{\partial n} = 0$. The wave equation (a special

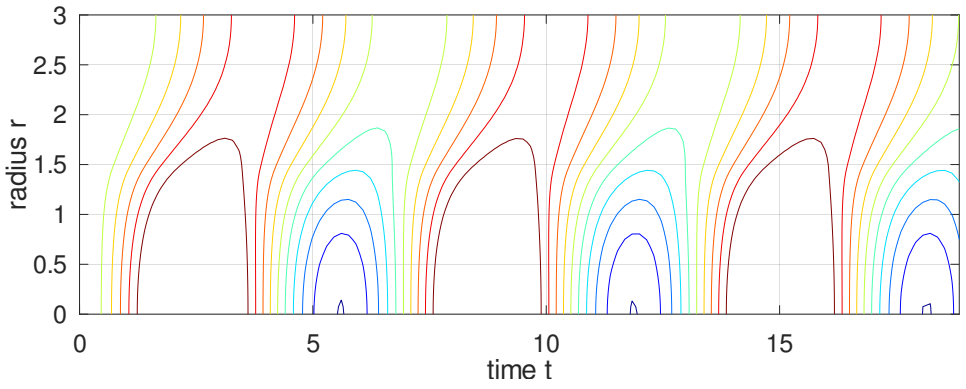


Figure 160: The contours of the temperatur $u(r,t)$ inside the cylinder

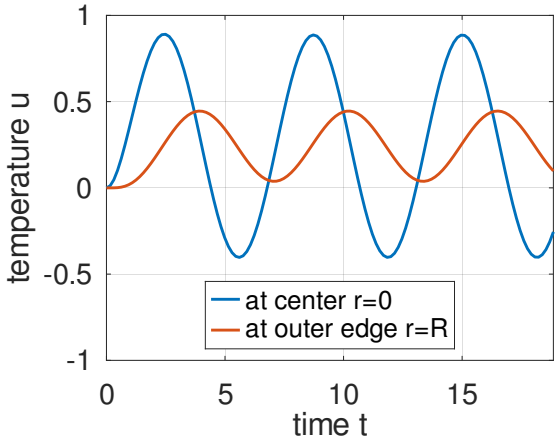


Figure 161: The temperatur $u(r,t)$ at the inner ($r=0$) and outer ($r=R$) edge

case of the IBVP (9)) to be solved is

$$\begin{array}{rcll} \frac{\partial^2}{\partial t^2}\, u(x,y,t) - \Delta\, u(x,y,t) & = & f(x,y,t) & \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\ \frac{\partial}{\partial n}\, u(x,y,t) & = & 0 & \text{for} \quad (x,y,t) \in \Gamma \times (0,T] \\ u(x,y,0) = \frac{\partial}{\partial t}\, u(x,y,0) & = & 0 & \text{on} \quad \Omega \end{array} \quad . \tag{124}$$

The function $f(x,y,t)$ equals $\sin(3\,\pi\,t)$ for $x \le -0.9$ and zero otherwise. Thus there a periodic excitation at the very left end of the appendix for $-1 \le x \le -0.9$. The wave speed equals 1 and the appendix (more precise: the two appendices) is a source of waves.

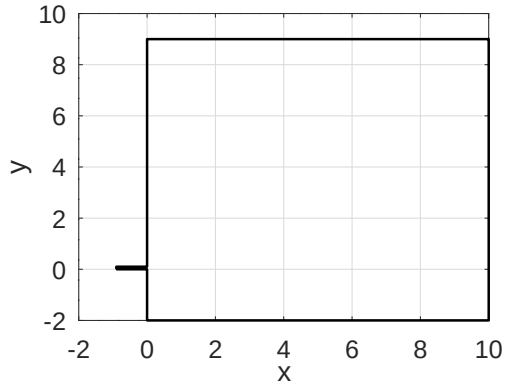


Figure 162: The domain for the wave propagation

Figure 163 shows the solution $u(x,y,11)$ at time $t = 11$.

- The wave speed equals 1, thus at $t = 11$ the first waves are about to arrive at $x = 10$ for $y = 0$ and at $y = +10$ for $x = 0$.
- In the top right section the unperturbed waves generated by the outlet of the appendix at $y = 0$ are visible.
- In the top left corner the upward moving waves interfere with the waves reflected at the upper edge at $y = 9$.
- At the lower edge at $y = -2$ the waves are reflected leading to interference. The result is identical to the situation of a second source at $y = -4$.
- In the lower part of the figure observe the result of the classical double–slit diffraction pattern by Kirchhoff, see e.g. en.wikipedia.org/wiki/Double-slit_experiment.

Observe that the behavior of the solution is very different from a heat equation in 9.16.4, even is the setup is comparable.

In the code below you can play with the different parameters and select whether an animation is shown on the screen or the final snapshot only.

WavePropagation.m

```
l = 1; h = 0.1; L = 10; d = 2; H = 9;
FEMmesh = CreateMeshTriangle('test',...
        [-l,0,-2;0 0 -2;0,-d,-2;L,-d,-2; L,H,-2;0,H,-2;0,h,-2;-l,h,-2],0.01);
figure(1); FEMtrimesh(FEMmesh)
           xlabel('x'); ylabel('y'); axis equal
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = f(xy,t)
  res = sin(3*pi*t)*ones(size(xy,1),1);
  res(xy(:,1)>-0.9) = 0;
endfunction
function res = v0(xy);  res = zeros(size(xy,1),1); endfunction
function res = u0(xy)   res = zeros(size(xy,1),1); endfunction
```

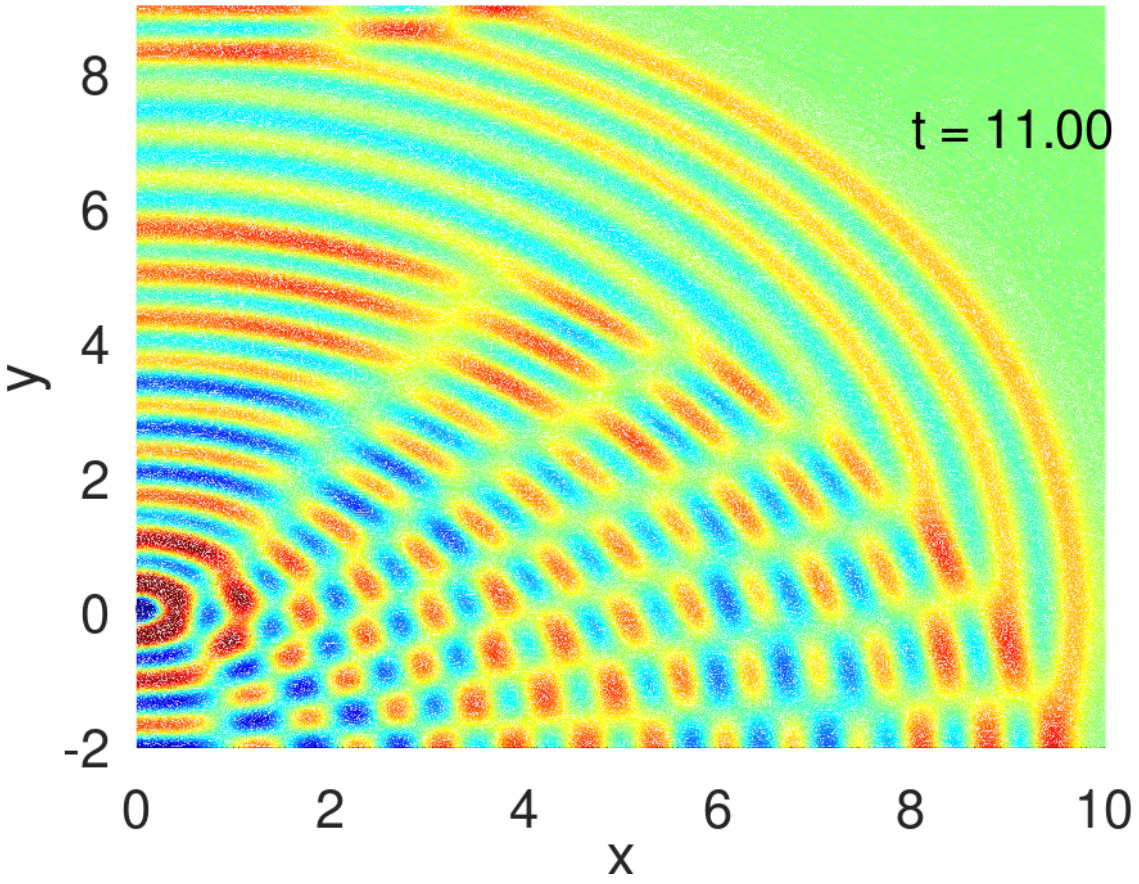


Figure 163: Wave propagation, leading to a Kirchhoff diffraction pattern

```
m = 1; a = 1; b0 = 0; bx = by = 0; f = 0; gn1 = gn2 = 0;
tic();
[u,t] = I2BVP2D(FEMmesh,m,0,a,b0,bx,by,'f',0,gn1,gn2,'u0','v0',0,11,[56,10]);
SolverTime = toc()

umax = 0.3*max([-min(u(:)),max(u(:))]);
figure(2)
if 0  %% animation
  for jj = 1:length(t)
    FEMtrimesh(FEMmesh,u(:,jj))
    xlabel('x'); ylabel('y')
    zlim(umax*[-1 1]); caxis(0.3*umax*[-1 1]);
    text(0.8*L,0.8*H,umax,sprintf('t = %4.2f',t(jj)))
    view(0,90); xlim([0,L]);  ylim([-d,H]);
    pause(0.1);
  endfor
else
  FEMtrimesh(FEMmesh,u(:,end))
  xlabel('x'); ylabel('y')
  zlim(umax*[-1 1]); caxis(0.3*umax*[-1 1]);
  text(0.8*L,0.8*H,umax,sprintf('t = %4.2f',t(end)))
  view(0,90); xlim([0,L]);  ylim([-d,H]);
endif
```

9.18 Sound waves in $\mathbb{R}^2$ and $\mathbb{R}^3$

The standard wave equation $\frac{\partial^2}{\partial t^2}\,u - \Delta u = 0$ can be written in cylindrical

$$\frac{\partial^2}{\partial t^2}\,u(\rho,\phi,z,t) = \frac{1}{\rho}\,\frac{\partial}{\partial \rho}(\rho\,\frac{\partial u}{\partial \rho}) + \frac{1}{\rho^2}\,\frac{\partial^2}{\partial \phi^2}\,u + \frac{\partial^2}{\partial z^2}\,u$$

or spherical coordinates

$$\frac{\partial^2}{\partial t^2}\,u(r,\phi,\theta,t) = \frac{1}{r^2}\,\frac{\partial}{\partial r}\,(r^2\,\frac{\partial\,u}{\partial r}) + \frac{1}{r^2\sin\theta}\,\frac{\partial}{\partial \theta}\,(\sin\theta\,\frac{\partial\,u}{\partial \theta}) + \frac{1}{r^2\,\sin^2\theta}\,\frac{\partial^2\,u}{\partial \phi^2}\ .$$

This allows to reduce some problems to two space dimensions.

9.18.1 A sound wave in $\mathbb{R}^3$ with cylindrical coordinates

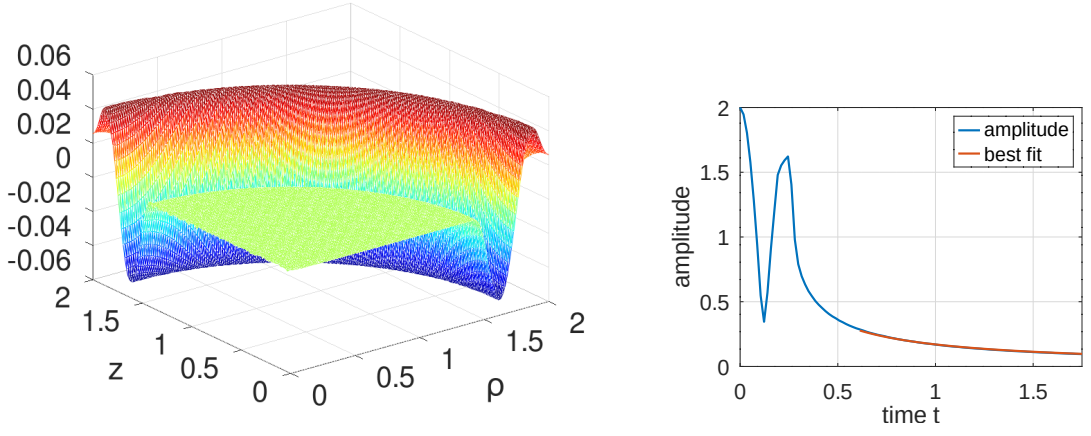
Assuming that the solution $u(\rho,z,t)$ is independent on ϕ and multiplying the wave equation by ρ arrive at

$$\rho\,\frac{\partial^2}{\partial t^2}\,u(\rho,z,t) - \frac{\partial}{\partial \rho}(\rho\,\frac{\partial u}{\partial \rho}) - \frac{\partial}{\partial z}\,(\rho\,\frac{\partial\,u}{\partial z}) = 0 \tag{125}$$

and thus it is in the from of the general hyperbolic equation (9) and can be solved numerically with `I2BVP2D()`. On a domain $0 \le \rho \le R$ and $0 \le \theta \le \pi$ we assume zero initial velocity $\frac{d}{dt}\,u(\rho,z,0) = 0$ and initial displacement

$$u(\rho,z,0) = \begin{cases} 1+\cos(10\,r) & \text{for } 0 \le r \le \frac{\pi}{10} \\ 0 & \text{for } \frac{\pi}{10} \le r \end{cases}\ ,$$

where we use $r = \sqrt{\rho^2+z^2}$. The result of solving this initial boundary value problem will be a spherical wave moving with speed 1 and a decaying amplitude. Find the result at time $t = 1.75$ in Figure 164. Using an energy argument the amplitude of the wave front is expected to decay like $c\,\frac{1}{t}$. Using linear regression for times $t > 0.6$ this is confirmed with the resulting Figure 164.



(a) the spherical wave

(b) decay of the ampltude

Figure 164: A spherical sound wave at time $t = 1.75$, and the decaying amplitude with the best fitting $\frac{c}{t}$

SoundWaveSpherical.m

```
R = 2; H = 2; N = 60;
FEMmesh = CreateMeshRect(linspace(0,R,N),linspace(0,H,N),-2,-2,-2,-2);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

function res = u_0(xy)
  r = sqrt(xy(:,1).^2+xy(:,2).^2);
  res = 1+cos(10*r);   res(r>pi/10) = 0;
```

```
endfunction
function res = rho(xy,dummy); res = xy(:,1);              endfunction;
function res = v_0(xy) ;      res = zeros(size(xy,1),1); endfunction

tic();
[u,t] = I2BVP2D(FEMmesh,'rho',0,'rho',0,0,0,0,0,0,0,'u_0','v_0',0,1.75,[100,10]);
ComputationTime = toc()

figure(1); clf
if 0 %% animation
  for jj = 1:length(t)
    FEMtrimesh(FEMmesh,u(:,jj))
    xlabel('rho'); ylabel('z'); zlim([-0.5 0.5]); caxis(0.1*[-0.5,0.5])
    pause(0.1)
  endfor
else
  FEMtrimesh(FEMmesh,u(:,end))
  xlabel('\rho'); ylabel('z')
endif

max_u = max(u) - min(u); t_start = find(t>0.6,1); t_tail = t(t_start:end)';

[p,~,~,p_var] = LinearRegression(1./t_tail,max_u(t_start:end)');
figure(2); plot(t,max_u,t_tail, p./t_tail)
            xlabel('time t'); ylabel ('amplitude'); legend('amplitude','best fit')
```

9.18.2 A sound wave in $\mathbb{R}^2$

In a rectangle $0 \le x, y \le R$ solve the standard wave equation

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0$$

with Neumann boundary conditions $\frac{\partial u}{\partial n} = 0$, zero initial velocity and initial displacement

$$u(x,y,0) = \begin{cases} 1 + \cos(10\,r) & \text{for } 0 \le r \le \frac{\pi}{10} \\ 0 & \text{for } \frac{\pi}{10} \le r \end{cases} ,$$

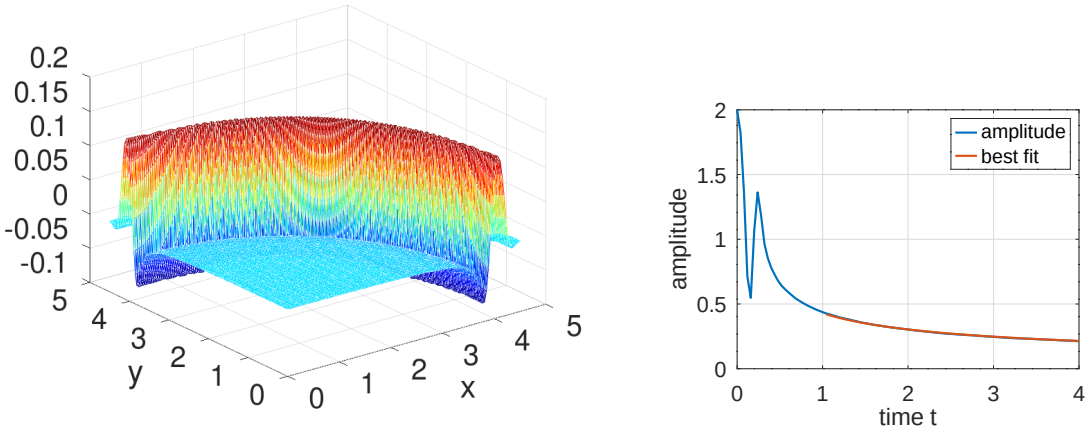
where we use $r = \sqrt{x^2 + y^2}$. The result of solving this initial boundary value problem will be a circular wave moving with speed 1 and a decaying amplitude. Find the result at time $t = 4$ in Figure 165(a). Using an energy argument the amplitude of the wave front is expected to decay like $c\,\frac{1}{\sqrt{t}}$. Using linear regression for times $t > 1$ this is confirmed by the resulting Figure 165(b).

SoundWave.m

```
R = 4.5; H = 4.5; N = 60;
FEMmesh = CreateMeshRect(linspace(0,R,N),linspace(0,H,N),-2,-2,-2,-2);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

function res = u_0(xy)
  r = sqrt(xy(:,1).^2+xy(:,2).^2);
  res = 1+cos(10*r);
  res(r>pi/10) = 0;
endfunction
function res = v_0(xy) ; res = zeros(size(xy,1),1); endfunction
tic();
[u,t] = I2BVP2D(FEMmesh,1,0,1,0,0,0,0,0,0,0,'u_0','v_0',0,4,[100,10]);
ComputationTime = toc()

figure(3); clf
if 0 %% animation
  for jj = 1:length(t)
```



(a) the wave (b) decay of the amplitude

Figure 165: A circular sound wave at time $t = 4$ and the decaying amplitude with the best fitting $\frac{c}{\sqrt{t}}$

```
    FEMtrimesh(FEMmesh,u(:,jj))
    xlabel('x'); ylabel('y');
    zlim(0.1*[-2 2]); caxis(0.5*[-2 2])
    pause(0.1)
  endfor
else
  FEMtrimesh(FEMmesh,u(:,end))
  xlabel('x'); ylabel('y')
endif

max_u = max(u) - min(u); t_start = find(t>1,1); t_tail = t(t_start:end)';
[p,~,~,p_var] = LinearRegression(1./sqrt(t_tail),max_u(t_start:end)');
figure(12); plot(t,max_u,t_tail, p./sqrt(t_tail))
            xlabel('time t'); ylabel ('amplitude'); legend('amplitude','best fit')
```

9.18.3 Sound waves in $\mathbb{R}^3$ and $\mathbb{R}^2$ as 1D problems

- As spherical wave:
  If a solution u of the wave equation $\frac{\partial^2}{\partial t^2}\, u = \Delta u$ depends on the radius $r = \sqrt{x^2 + y^2 + z^2}$ only the IBVP is given by

$$\begin{aligned} r^2\, \tfrac{\partial^2}{\partial t^2}\, u(r,t) &= \tfrac{\partial}{\partial r}\left(r^2\, \tfrac{\partial}{\partial r}\, u(r,t)\right) && \text{for } 0 < r < R \text{ and } t > 0 \\ u(r,0) &= u_0(r) && \text{for } 0 < r < R \\ \tfrac{\partial}{\partial t}\, u(r,0) &= u_1(r) && \text{for } 0 < r < R \end{aligned}$$

  For a zero initial velocity $u_1(r) = 0$ and initial amplitude

$$u(r,0) = u_0(r) = \begin{cases} 1 + \cos(10\, r) & \text{for } 0 \le r \le \frac{\pi}{10} \\ 0 & \text{for } \frac{\pi}{10} \le r \end{cases}$$

  and Neumann boundary conditions at $r = 0$ and $r = R$ the problem can be solved by `I2BVP1D()` with the code `SoundWaveSpherical1D.m` below. Find the result in Figure 166. Observe that for advanced time t the solution is equal to zero at the origin $r = 0$.

SoundWaveSpherical1D.m

```
N = 2*60; R = 3; interval = linspace(0,R,N)';
f_r2 = @(r)r.^2;
u0 = @(r)(1+cos(10*r)).*(r<pi/10);   u1 = 0;
w2 = f_r2; w1 = 0; a = f_r2; b = 0; c = 0; d = 1; f = 0;
```

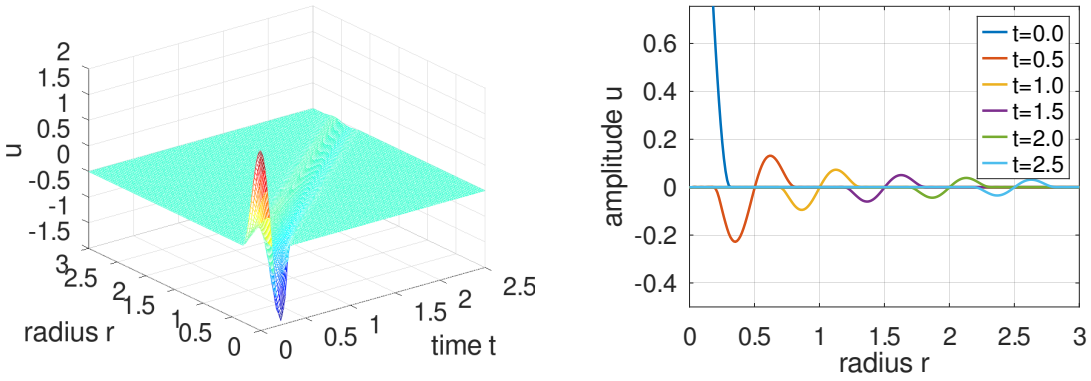


Figure 166: A spherical wave as function of time t and radius $r = \sqrt{x^2 + y^2 + z^2}$

```
BCleft = [0,0]; BCright = [0,0];
t0 = 0; tend = 2.5; steps = [100,10];
[r,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,steps);

figure(1); mesh(t,r,u); xlabel('time t'); ylabel('radius r'); zlabel('u')
           xlim([min(t),max(t)]); ylim([min(r),max(r)])

[t05,t05_ind] = find(abs(t-0.5)<1e-5); [t1,t1_ind] = find(abs(t-1)<1e-5);
[t15,t15_ind] = find(abs(t-1.5)<1e-5); [t2,t2_ind] = find(abs(t-2)<1e-5);
[t25,t25_ind] = find(abs(t-2.5)<1e-5);

figure(2); plot(r,u(:,1),r,u(:,t05_ind),r,u(:,t1_ind),r,u(:,t15_ind),...
                r,u(:,t2_ind),r,u(:,t25_ind));
           ylabel('amplitude u'); ylim([-0.5,0.7]); xlabel('radius r');
           legend('t=0.0','t=0.5','t=1.0','t=1.5','t=2.0','t=2.5',...
                  'location','northeast')
```

- As cylindrical wave:
  If a solution u of the wave equation $\frac{\partial^2}{\partial t^2}\, u = \Delta u$ depends on the radius $r = \sqrt{x^2 + y^2}$ only the IBVP is given by

$$\begin{aligned} r\, \tfrac{\partial^2}{\partial t^2}\, u(r,t) &= \tfrac{\partial}{\partial r}\, \left(r\, \tfrac{\partial}{\partial r}\, u(r,t) \right) && \text{for } 0 < r < R \text{ and } t > 0 \\ u(r,0) &= u_0(r) && \text{for } 0 < r < R \\ \tfrac{\partial}{\partial t}\, u(r,0) &= u_1(r) && \text{for } 0 < r < R \end{aligned}$$

  For a zero initial velocity $u_1(r) = 0$ and the same initial amplitude $u_0(r)$ as above and Neumann boundary conditions at $r = 0$ and $r = R$ the problem can be solved with the code `SoundWaveSpherical1D.m` below, using `I2BVP1D()`. Find the result in Figure 167. Observe that for advanced time t the solution is **not equal** to zero at the origin $r = 0$. The amplitudes at $t > 0$ are larger than for the above spherical case. This can be derived analytically, using a conservation of energy argument.

SoundWaveCylindrical1D.m

```
N = 2*60; R = 3; interval = linspace(0,R,N)';
f_r = @(r)r;
u0 = @(r)(1+cos(10*r)).*(r<pi/10);   u1 = 0;
w2 = f_r; w1 = 0; a = f_r; b = 0; c = 0; d = 1; f = 0;
BCleft = [0,0]; BCright = [0,0];
t0 = 0; tend = 2.5; steps = [100,10];
[r,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,steps);

figure(1); mesh(t,r,u); xlabel('time t'); ylabel('radius r'); zlabel('u')
           xlim([min(t),max(t)]); ylim([min(r),max(r)])
```

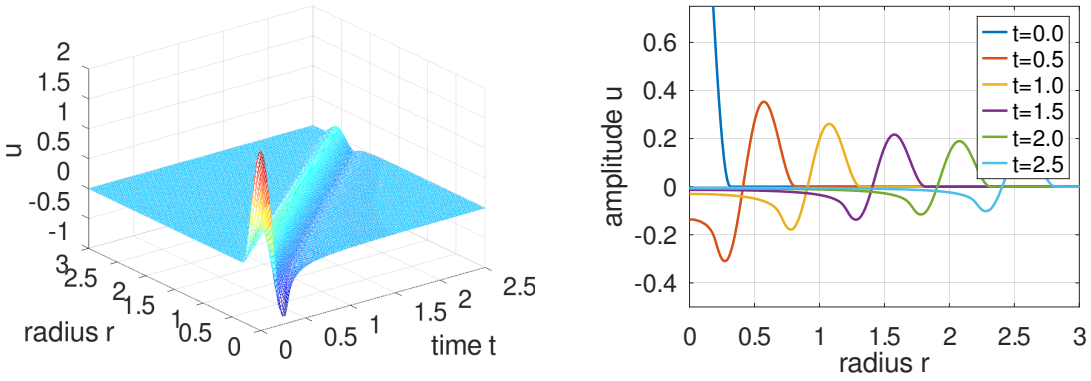


Figure 167: A cylindrical wave as function of time t and radius $r = \sqrt{x^2 + y^2}$

```
[t05,t05_ind] = find(abs(t-0.5)<1e-5); [t1,t1_ind] = find(abs(t-1)<1e-5);
[t15,t15_ind] = find(abs(t-1.5)<1e-5); [t2,t2_ind] = find(abs(t-2)<1e-5);
[t25,t25_ind] = find(abs(t-2.5)<1e-5);

figure(2); plot(r,u(:,1),r,u(:,t05_ind),r,u(:,t1_ind),r,u(:,t15_ind),...
              r,u(:,t2_ind),r,u(:,t25_ind));
           ylabel('amplitude u'); ylim([-0.5,0.75]); xlabel('radius r');
           legend('t=0.0','t=0.5','t=1.0','t=1.5','t=2.0','t=2.5',...
                  'location','northeast')
```

9.19 Reflection and transmission of a wave by a change of impedance

A change of impedance will cause an incoming wave to be partially reflected. Examine the initial boundary value problem

$$\begin{array}{rcll} \frac{\partial^2}{\partial t^2} u(x,t) & = & \frac{\partial}{\partial x}\left(a(x)\,\frac{\partial}{\partial x} u(x,t)\right) & \text{for } 0 < x < 12 \text{ and } t > 0 \\ \frac{\partial}{\partial x} u(0,t) = \frac{\partial}{\partial x} u(12,t) & = & 0 & \text{for } t > 0 \\ u(x,0) & = & u_0(x) & \text{for } 0 < x < 12 \\ \frac{\partial}{\partial t} u(x,0) & = & u_1(x) & \text{for } 0 < x < 12 \end{array}$$

with the discontinuous coefficient $a(x)$

$$a(x) = \begin{cases} 1 & \text{for } 0 \le x \le 4 \\ 2 & \text{for } 4 < x \end{cases}$$

and the initial values

$$u_0(x) = \begin{cases} \sin(\pi\, x) & \text{for } 0 \le x \le 1 \\ 0 & \text{for } 1 < x \end{cases} \quad \text{and} \quad u_1(x) = \begin{cases} -\pi\, \cos(\pi\, x) & \text{for } 0 \le x \le 1 \\ 0 & \text{for } 1 < x \end{cases} .$$

The sin–shaped pulse will travel in the positive x–direction with speed 1 and then at $x = 4$ a part will be reflected and traveling back with speed 1 and another part will continue, but with speed $\sqrt{2}$. In Figure 168 find the plot of the surface and in Figure 169 a plot of the solutions at times $t = 0$, 2, 4, 6 and 8, and the contour plot.

Reflection.m

```
interval = linspace(0,14,301)';
a = @(x) 1+1*(x>4);
b = 0; c = 0; d = 1; f = 0;
w2 = 1; w1 = 0; ; BCleft = [0,0]; BCright = [0,0];
t0 = 0; tend = 9; steps = [90,100];
```

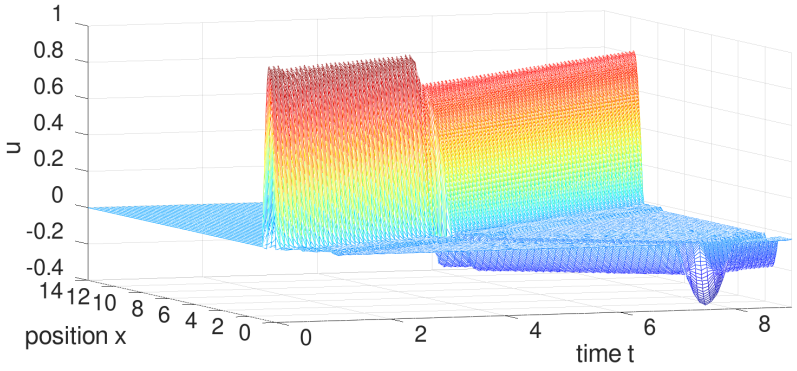
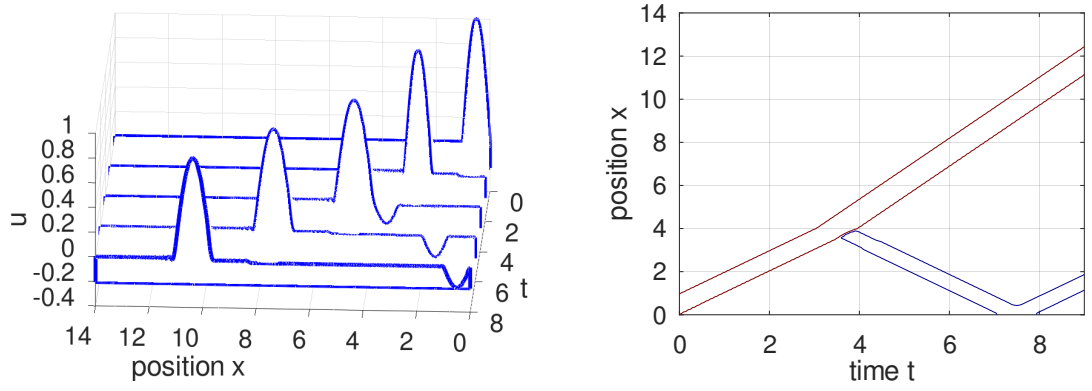


Figure 168: The amplitude $u(x,t)$ for a reflected pulse



(a) slices at $t = 0$, 2, 4, 6 and 8

(b) the contours at $u(x,t) = \pm 0.1$

Figure 169: Waterfall plot and contour plot for the amplitude $u(x,t)$ for a reflected pulse

```
u0 = @(x)sin(pi*x).*(x<=1); u1 = @(x)-pi*cos(pi*x).*(x<=1);

[x,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,steps);

figure(1); mesh(t,x,u); xlabel('time t'); ylabel('position x'); zlabel('u')
           xlim([min(t),max(t)]); ylim([min(x),max(x)]); view([-20,10])
t_ind = [1 21 41 61 81];
figure(2); H = waterfall(x,t(t_ind),u(:,t_ind)'); view([-177,20])
           xlabel('position x'); zlabel('u')
           set(H, 'edgecolor', [0,0.0,1]); set(H, 'linewidth', 3)
figure(3); contour(t,x,u,[-0.1,0.1]); xlabel('time t'); ylabel('position x');
```

9.20 Sine–Gordon equation

The Sine–Gordon equation is given by

$$\frac{\partial^2}{\partial t^2}\, u(t,x) - \frac{\partial^2}{\partial x^2}\, u(t,x) = \sin(u(t,x))\ . \tag{126}$$

9.20.1 Traveling kink solutions of the Sine–Gordon equation

Traveling solutions of the Sine–Gordon equation are determined by

$$\begin{aligned} u(x,t) &= f(x-ct) = f(\xi) \\ c^2\, f''(\xi) - f''(\xi) &= \sin(f(\xi)) \\ (c^2-1)\, f''(\xi) &= \sin(f(\xi)) \end{aligned}$$

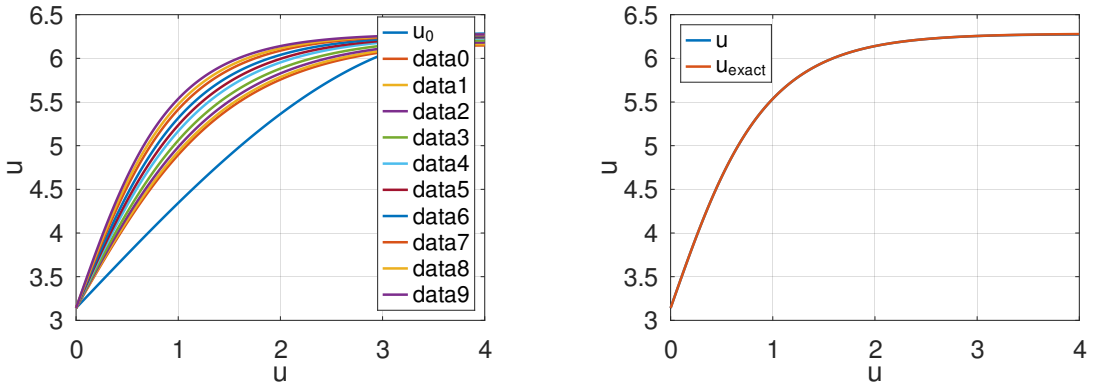
For speeds $0 < c < 1$ exact solutions are given by

$$f(\xi) = 4\, \arctan\left(\exp\left(\pm\frac{\xi-\xi_0}{\sqrt{1-c^2}}\right)\right)\ .$$

Use `BVP1DNL()` to construct solutions of the nonlinear boundary value problem

$$-(1-c^2)\,\frac{\partial^2}{\partial \xi^2}\, f(\xi) = \sin(f(\xi) \quad \text{for } 0<\xi<L \text{ with } f(0)=\pi \text{ and } \tfrac{\partial}{\partial \xi}\, f(L) = 0 \quad .$$

For larger values of $c < 1$ it is difficult to find good initial guesses for the Newton method used by `BVP1DNL()`. The code below increases the parameter c step by step from $c = 0.3$ to $c = 0.8$, leading to Figure 170.



(a) solutions for different parameters c

(b) exact and FEM solution for $c = 0.8$

Figure 170: Traveling kink solutions for the Sine–Gordon equation

SineGordonTravelingParametric.m

```
c = 0.2; L = 4;  Interval = linspace(0,L,1001);
f  = {@(x,u)sin(u),@(x,u)cos(u)};
u0 = @(x)pi+pi*sin(pi/2*x/L);
[x,u,inform] = BVP1DNL(Interval,(c^2-1),0,0,1,f,pi,[0,0],u0);
figure(1); clf; plot(x,u0(x),x,u)
           xlabel('u'); ylabel('u');
           legend('u_0','data0','location','southeast')
           hold on
c_list = [0.3 0.4 0.5 0.6 0.65 0.7 0.75 0.775 0.8];
for ii = 1:length(c_list)
  c = c_list(ii)
  [x,u,inform] = BVP1DNL(Interval,(c^2-1),0,0,1,f,pi,[0,0],u,'tol',1e-10,'display','off');
  figure(1); plot(x,u); xlabel('u'); ylabel('u');
  pause(1)
endfor
u0 = @(x)4*atan(exp(x/sqrt(1-c^2)));
figure(2); plot(x,u,x,u0(x)); xlabel('u'); ylabel('u');
           legend('u','u_{exact}','location','northwest'); hold off
```

9.20.2 Breather solutions of the Sine–Gordon equation

For parameters $0 < \omega < 1$ the Sine–Gordon equation (126) admits periodic solutions, called breather solutions. The exact solutions are given by

$$u(x,t) = 4 \arctan\left(\frac{\sqrt{1-\omega^2}\,\sin(\omega\,t)}{\omega\,\cosh(\sqrt{1-\omega^2}\,x)}\right) .$$

To determine these solution solve the dynamic, nonlinear IBVP for a large value of L.

$$\begin{aligned}
\tfrac{\partial^2}{\partial t^2}\,u(t,x) - \tfrac{\partial^2}{\partial x^2}\,u(t,x) &= \sin(u(t,x)) && \text{for } 0 < x < L \text{ and } 0 < t \\
\tfrac{\partial}{\partial x}u(t,0) = u(t,L) &= 0 && \text{for } 0 < t \\
u(0,x) &= 0 && \text{for } 0 < x < L \\
\tfrac{\partial}{\partial t}u(0,x) &= 4\,\tfrac{\sqrt{1-\omega^2}}{\cosh(\sqrt{1-\omega^2}\,x)} && \text{for } 0 < x < L
\end{aligned}$$

The initial conditions are evaluated using elementary calculus[45]. The code `SineGordonBreather.m` generates Figure 171.

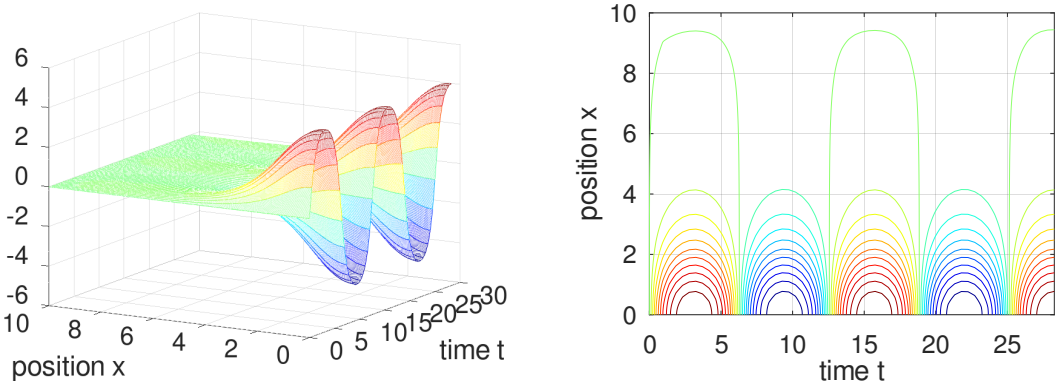


Figure 171: A breather solution of the Sine–Gordon equation

45

$$\frac{\partial}{\partial t}\,u(x,t) = 4\left(1+\left(\frac{\sqrt{1-\omega^2}\,\sin(\omega\,t)}{\omega\,\cosh(\sqrt{1-\omega^2}\,x)}\right)^2\right)^{-1}\left(\frac{\sqrt{1-\omega^2}\,\cos(\omega\,t)}{\cosh(\sqrt{1-\omega^2}\,x)}\right) \quad \text{thus} \quad \frac{\partial}{\partial t}\,u(x,0) = 4\,\frac{\sqrt{1-\omega^2}}{\cosh(\sqrt{1-\omega^2}\,x)}$$

SineGordonBreather.m

```
omega = 0.5; L = 10; Interval = linspace(0,L,501); Tend = 4.5*pi/omega;

f  = @(x,t,u)-sin(u);
v0 = @(x)4*sqrt(1-omega^2)./cosh(sqrt(1-omega^2)*x);
[x,u,t] = I2BVP1DNL(Interval,1,0,1,0,0,1,f,[0,0],0,0,v0,0,Tend,[61,100] ,'Solver','implicit');

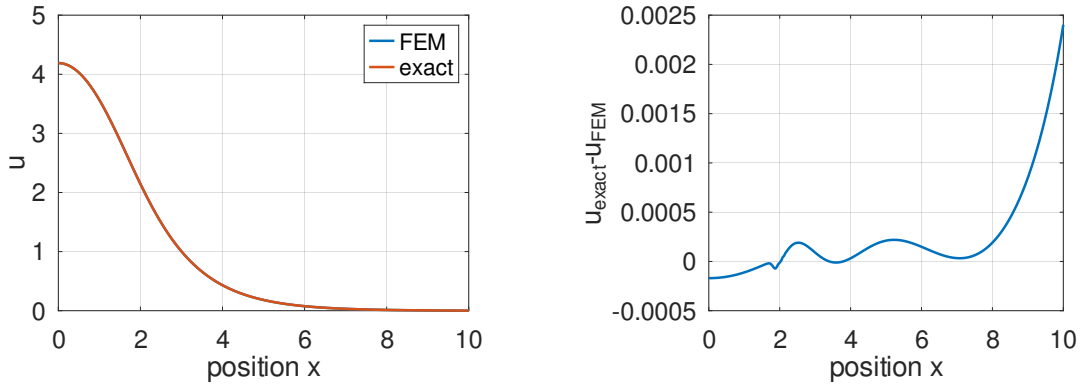
figure(1); mesh(t,x,u); view(-60,15)
           xlabel('time t'); ylabel('position x')
figure(2); contour(t,x,u,21)
           xlabel('time t'); ylabel('position x')

u_max = @(x)4*atan(sqrt(1-omega^2)./(omega*cosh(sqrt(1-omega^2)*x)));
figure(3); plot(x,u(:,end),x,u_max(x))
           xlabel('position x'); ylabel('u'); legend('FEM','exact')
figure(4); plot(x,u_max(x)-u(:,end))
           xlabel('position x'); ylabel("u_{exact}-u_{FEM}");
```

At times $t_{max} = \frac{(2\,n+\frac{1}{2})\,\pi}{\omega}$ use $\sin(\omega\,t) = 1$ to find

$$u(t_{max}, x) = 4\,\arctan\left(\frac{\sqrt{1-\omega^2}}{\omega\,\cosh(\sqrt{1-\omega^2}\,x)}\right)\,.$$

This exact solution can be compared to the FEM solution, shown in Figure 172. The main contribution to the difference $u_{exact} - u_{FEM}$ is determined by the finite interval length $L = 10$.



(a) exact and FEM solution (b) the difference

Figure 172: The final state of a breather solution of the Sine–Gordon equation

9.21 Scattering, Helmholtz equation

Examine the solution of the scattering problem due to an infinite array of circular conductors of radius a at a distance d, lit up by a plane wave linearly polarized along the wires and direction perpendicular to the array. Evaluate the transmission coefficient and reflection coefficient. This example was proposed by Giuseppe Gonzalves.

For the wave equation $\frac{\partial^2}{\partial t^2}\,U(x,y,t) = c^2 \Delta\,U(x,y,t)$ examine time harmonic solutions with angular velocity ω, i.e. $U(x,y,t) = u(x,y)\,e^{+i\,\omega\,t}$. This leads to

$$\begin{aligned} \frac{\partial^2}{\partial t^2}\,U(x,y,t) &= c^2\,\Delta U(x,y,t) \\ -\omega^2\,u(x,y)\,e^{+i\,\omega\,t} &= c^2\,\Delta u(x,y)\,e^{+i\,\omega\,t} \\ -\frac{\omega^2}{c^2}\,u(x,y) &= \Delta u(x,y) \end{aligned}$$

With $k = \frac{\omega}{c}$ the boundary value problem to be examined is a Helmholtz[46] equation.

$$\begin{array}{rcll} -\Delta u - k^2\, u & = & 0 & \text{for } -l < x < l \text{ and } -\frac{d}{2} < y < +\frac{d}{2} \\ \frac{\partial}{\partial y} u(x,y) & = & 0 & \text{for } -l < x < l \text{ and } y = \pm\frac{d}{2} \\ -\frac{\partial}{\partial x} u(-l,y) & = & 2\,i\,k - i\,k\,u(-l,y) & \text{for } -\frac{d}{2} < y < +\frac{d}{2} \\ +\frac{\partial}{\partial x} u(+l,y) & = & -i\,k\,u(l,y) & \text{for } -\frac{d}{2} < y < +\frac{d}{2} \end{array} \tag{127}$$

Observe that some of the coefficients on the boundary are complex, thus a BVP with complex coefficients and solutions has to be solved.

To motivate the boundary conditions in (127) examine a one dimensional setup $-l < x < +l$ and assume an incoming signal $u_{in}(x)$ from the left is transmitted and reflected at $x = 0$, leading to $u_{out}(x)$ and $u_{ref}(x)$.

$$u(x) = \begin{cases} u_{in}(x) + u_{ref}(x) & \text{for } x < 0 \\ u_{out}(x) & \text{for } x > 0 \end{cases}$$

Work with functions of the type $e^{i\,k\,(c\,t \pm x)} = e^{i \pm i\,k\,x}\, e^{i\,\omega\,t}$.

- For a nonreflecting boundary condition at $x = +l$ use

$$\begin{array}{rcl} U_{out}(x,t) & = & c_{out}\, e^{i\,k\,(c\,t-x)} \;=\; u_{out}(x)\, e^{+i\,\omega\,t} \\ u_{out}(x) & = & c_{out}\, e^{-i\,k\,x} \\ \dfrac{\partial}{\partial x}\, u_{out}(x) & = & -i\,k\,c_{out}\, e^{-i\,k\,x} \;=\; -i\,k\,u_{out}(x) \end{array}$$

  The coefficient $c_{out} = |c_{out}|\, e^{i\delta}$ leads to the amplitude $|c_{out}|$ and the phase shift δ of the transmitted signal $u_{out}(x)$.

- The incoming signal with amplitude 1 is described by

$$U_{in}(x,t) = e^{i\,k\,(c\,t-x)} = e^{-i\,k\,x}\, e^{i\,\omega\,t} = u_{in}(x)\, e^{i\,\omega\,t}\,.$$

  For the sum of the incoming and and the reflected signal on $-l < x < 0$ assume

$$\begin{array}{rcl} U_{in}(x,t) + U_{ref}(x,t) & = & e^{i\,k\,(c\,t-x)} + c_{ref}\, e^{i\,k\,(c\,t+x)} \\ & = & \left(e^{-i\,k\,x} + c_{ref}\, e^{i\,k\,x}\right) e^{+i\,\omega\,t} \;=\; e^{-i\,k\,x}\, \left(1 + c_{ref}\, e^{i\,k\,2\,x}\right) e^{+i\,\omega\,t} \\ u_{in}(x) + u_{ref}(x) & = & e^{-i\,k\,x} + c_{ref}\, e^{+i\,k\,x} \\ -\dfrac{\partial}{\partial x}\, (u_{in}(-l) + u_{ref}(-l)) & = & +i\,k\, \left(e^{+i\,k\,l} - c_{ref}\, e^{-i\,k\,l}\right) \\ 2\,i\,k - i\,k\,(u_{in}(-l) + u_{ref}(-l)) & = & 2\,i\,k - i\,k\,e^{i\,k\,l}\,(1 + c_{ref}\, e^{-i\,2\,k\,l}) \end{array}$$

  With the values $k = 2\,\pi$ and $l = 3$ use $e^{\pm i\,k\,l} = e^{\pm i\,6\,\pi} = 1$ and obtain

$$\begin{array}{rcl} -\dfrac{\partial}{\partial x}\, (u(-l) + u_{ref}(-l)) & = & +i\,2\,\pi\, (1 - c_{ref}) \\ 2\,i\,2\,\pi - i\,2\,\pi\,(u(-l) + u_{ref}(-l)) & = & 2\,i\,2\,\pi - i\,2\,\pi\,(1 + c_{ref}) \;=\; +i\,2\,\pi\,(1 - c_{ref}) \end{array}$$

  This is the boundary condition at $x = -l = -3$ in (127).

The code in `Scattering.m` solves the BVP (127) and generates Figure 173 with the complex solution $u(x,y)$. Observe that the absolute value of the solution is rather constant for $x > 0$, consistent with $u(x,y) = u_{out}(x,y) = c_{out}\, e^{-i\,k\,x}$.

Scattering.m

```
global k len
k = 2*pi; d = 0.5; len = 3; a = d/10;
xy = [len d/2 -2; len -d/2 -2; -len -d/2 -2; -len d/2 -2];
Np = 36; phi = linspace(0,2*pi,Np+1)(1:Np); phi = phi';
Wire.name = 'Hole';
```

[46] Hermann von Helmholtz (1821–1894), German physicist

```
Wire.border = [a*cos(phi) a*sin(phi) -ones(Np,1)];    Wire.point = [0 0];
Mesh = CreateMeshTriangle('strip',xy,1/(200),Wire);
Mesh = MeshUpgrade(Mesh,'cubic');

function res = gn1(xy,dummy)
   global k len
   res = 2*j*k*(xy(:,1)<=(-len+2*eps));     %% applies on left edge
end
function res = gn2(xy,dummy)
   global k len
   res = -j*k*(abs(xy(:,1))>=(len-2*eps));  %% applies on left and right edge
end

u = BVP2D(Mesh,1,-k^2,0,0,0,0,'gn1','gn2','type','complex');

figure(1); FEMtrisurf(Mesh,real(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('real(u(x))')
figure(2); FEMtrisurf(Mesh,imag(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('imag(u(x))')
figure(3); FEMtrisurf(Mesh,abs(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('abs(u(x))')
```

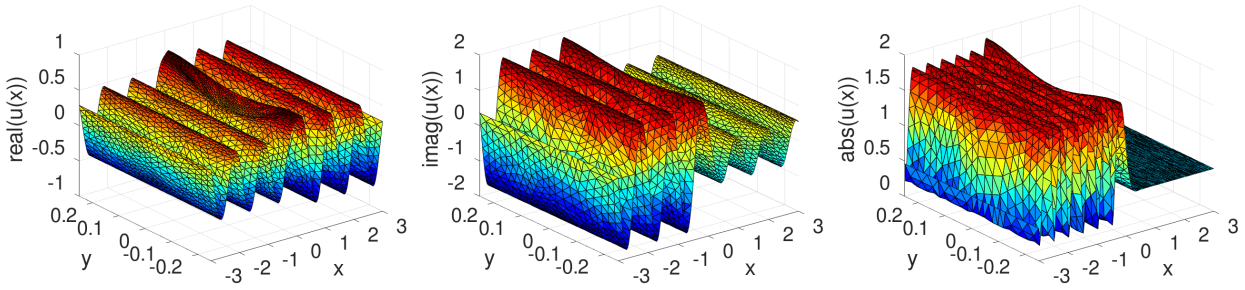


Figure 173: The solution of a Helmholtz equation

To determine the transmission and reflection coefficients use values along the line $y = 0$ and $-l \le x \le +l$, i.e.

$$\tau_{ref} = c_{ref} = u(-l, 0) - 1 \quad \text{and} \quad \tau_{trans} = c_{out} = u(+l, 0) \; .$$

The code below evaluates the solution $u(x, 0)$ for $-l \le x \le +l$ and leads to Figure 174.

Scattering.m

```
x = linspace(-len,+len,5001); y = zeros(size(x));
[u_mid] = FEMgriddata(Mesh,u,x,y);
figure(11); plot(x,real(u_mid)); title('real(u(x))')
figure(12); plot(x,imag(u_mid)); title('imag(u(x))')
figure(13); plot(x,abs(u_mid)); title('abs(u(x))')
figure(14); plot3(x,real(u_mid),imag(u_mid));
            xlabel('x'); ylabel('real(u)'); zlabel('imag(u)'); axis equal
figure(15); plot(real(u_mid),imag(u_mid));
            xlabel('real(u)'); ylabel('imag(u)'); axis equal

tau_transmit = u_mid(end)
tau_reflect  = u_mid(1)-1
-->
tau_transmit =  0.2711 + 0.5396i
tau_reflect  = -0.7122 + 0.3578i
```

In Figure 174 observe:

- For $x > 0$ the values of $u(x, 0) \in \mathbb{C} \sim \mathbb{R}^2$ are on a circle, confirming the assumption $u_{out}(x, 0) = c_{out}\, e^{-i\,k\,x}$.

- For $x < 0$ the values of $u(x,0) \in \mathbb{C}$ are on an ellipse like curve, confirming the assumption

$$u(x,0) = u_{in}(x,0) + u_{ref}(x,0) = e^{-i\,k\,x}\,(1 + c_{ref}\,e^{+i\,k\,2\,x})\ .$$

  Using a vector notation in $\mathbb{C} \sim \mathbb{R}^2$ this can be written in the form

$$\vec{u}(x,0) = \begin{bmatrix} \cos(k\,x) & +\sin(k\,x) \\ -\sin(k\,x) & \cos(k\,x) \end{bmatrix} \begin{pmatrix} 1 + |c_{ref}|\,\cos(2\,k\,x + \delta) \\ 0 + |c_{ref}|\,\sin(2\,k\,x + \delta) \end{pmatrix}\ .$$

  This is a circular movement around the point $(1,0)^T$ with radius $|c_{ref}|$ and angular velocity $2\,k$, all rotated with angular velocity $-k$ about the origin.

- Conservation of energy should imply $|c_{ref}|^2 + |c_{out}|^2 = 1$, which is confirmed by the above values.

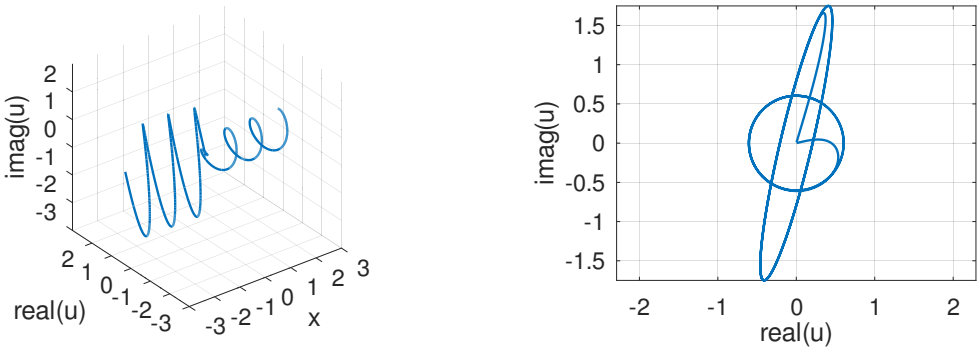


Figure 174: The solution $u(x,0)$ for $-l \le x \le +l$ of a Helmholtz equation

In the above scattering problem the transmission and reflection of the wave was caused by the conductor, modeled by the hole in the domain. Scattering can also be caused by changes in the material, e.g. a circular domain in the middle allowing for twice the speed of the wave. Since $k = \frac{\omega}{c}$ this is implemented by dividing the factor $-k^4$ by 4. Find an implementation in the code `ScatteringNoHole.m`, leading to Figures 175 and 176.

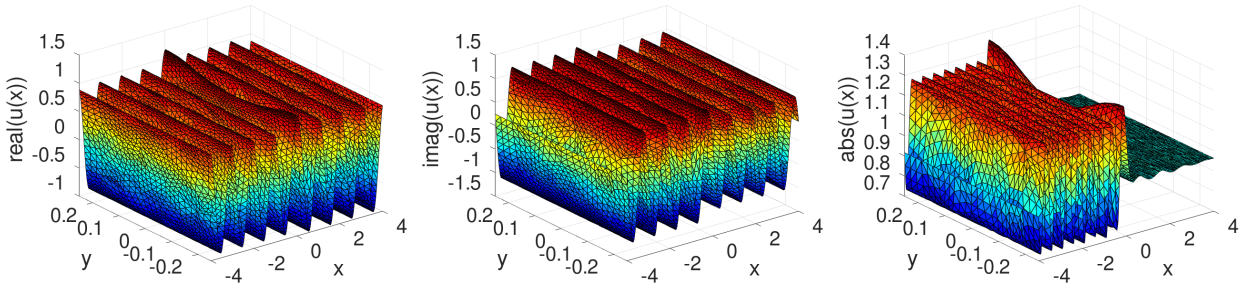


Figure 175: The solution of a Helmholtz equation with variable speed c of the wave

ScatteringNoHole.m

```
global k len a
k = 2*pi; d = 0.5; len = 4; a = d/3;
xy = [len d/2 -2; len -d/2 -2; -len -d/2 -2; -len d/2 -2];
Mesh = CreateMeshTriangle('strip',xy,len/(2*800));
Mesh = MeshUpgrade(Mesh,'cubic');

function res = gn1(xy,dummy)
```

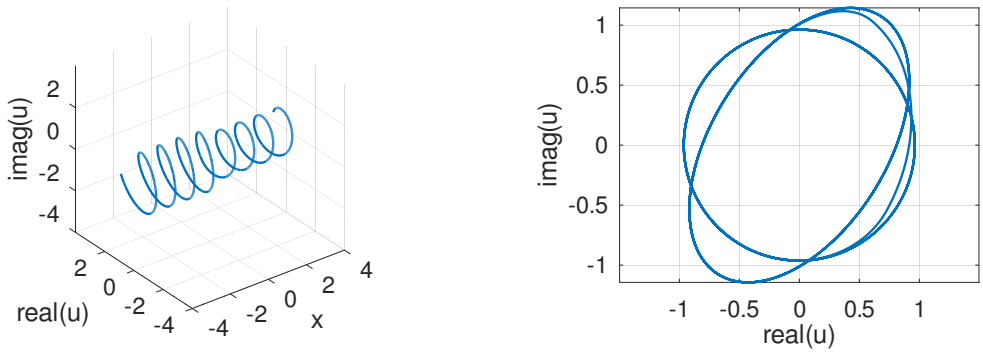


Figure 176: The solution $u(x,0)$ for $-l \leq x \leq +l$ of a Helmholtz equation with variable c

```
   global k len
   res = 2*j*k*(xy(:,1)<=(-len+10*eps));         %% applies on left edge
end

function res = gn2(xy,dummy)
   global k len
   res = (0-1j*k)*(abs(xy(:,1))>=(len-10*eps));  %% applies on left and right edge
end

function res = k2(xy)
  global k a
  alpha = 0.25;
  x = xy(:,1); y = xy(:,2);
  res = -k^2*ones(size(x));
  if 1  %% circle with different speed
     ind = (x.^2+y.^2)<a^2;
   else  %% band with different speed
     ind = x.^2<a^2;
  endif
  res(ind) *= alpha;
endfunction

u = BVP2D(Mesh,1,'k2',0,0,0,0,'gn1','gn2','type','complex');

figure(1); FEMtrisurf(Mesh,real(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('real(u(x))')
figure(2); FEMtrisurf(Mesh,imag(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('imag(u(x))')
figure(3); FEMtrisurf(Mesh,abs(u)); ylim([-d/2,d/2])
           xlabel('x'); ylabel('y'); zlabel('abs(u(x))')
x = linspace(-len,+len,5001); y = zeros(size(x));
[u_mid] = FEMgriddata(Mesh,u,x,y);
figure(11); plot(x,real(u_mid)); xlabel('x');  title('real(u(x))')
figure(12); plot(x,imag(u_mid)); title('imag(u(x))')
figure(13); plot(x,abs(u_mid)); title('abs(u(x))')
figure(14); plot3(x,real(u_mid),imag(u_mid));
            xlabel('x'); ylabel('real(u)'); zlabel('imag(u)'); axis equal
figure(15); plot(real(u_mid),imag(u_mid));
            xlabel('real(u)'); ylabel('imag(u)'); axis equal
tau_transmit = u_mid(end)
tau_reflect  = u_mid(1)-1
-->
tau_transmit =   0.8710 + 0.4097i
tau_reflect  =  -0.1171 + 0.2445i
```

9.22 The Black–Scholes equation of mathematical finance

To determine the value of a stock option the partial differential equation of Black, Scholes and Merton can be used, see [Seyd00], [Seyd11] or [Stew13]. A **call option** on a stock gives you the right (but not the obligation) to buy the stock at the fixed maturity time T for a given strike price K. If the actual price S is above the strike K, you call the option and gain $S - K$. If the actual price S is below the strike K you let the option expire. For this right to buy you have to pay a fair price V, the value of this call option. To determine the fair price of the option the possible evolution of the value of the stock is taken into account, assuming that it is a Brownian motion. For the Black–Scholes equation use the symbols in Table 23.

	Symbol
value of stock	$S = e^z$
logarithm of value of stock	$z = \ln(S)$
time to maturity	$0 \le \tau \le T$
value of option at time τ at stock value $S = e^z$	$V(z, \tau)$
safe interest rate	r_0
expected return of stock	r
volatility	σ
strike	K
forward time	$t = T - \tau > 0$

Table 23: Variables for the Black–Scholes PDE

The Black–Scholes PDE is given by[47]

$$\frac{\partial}{\partial \tau} V(z,\tau) = \frac{\sigma^2}{2} \frac{\partial^2}{\partial z^2} V(z,\tau) + r \frac{\partial}{\partial z} V(z,\tau) - r_0 V(z,\tau) \tag{128}$$

with the boundary conditions for a call option

$$V(S,\tau) \approx 0 \quad \text{for } S \text{ very small and} \quad V(S,\tau) \approx S \quad \text{for } S \text{ very large}$$

leading to

$$\lim_{z \to -\infty} V(z,\tau) \approx 0 \quad \text{and} \quad \lim_{z \to +\infty} \frac{1}{e^z} V(z,\tau) \approx 1 \; .$$

To respect the compatibility condition at maturity use $V(t,b) = S - K = e^b - K$ for $b \gg 1$. The initial conditions at $\tau = 0$ or $t = T$ are

$$V(z,0) \;=\; \max\{0, S-K\} \;=\; \max\{0, e^z - K\} \; .$$

This initial boundary value problem can be solved by `FEMoctave` with the command `IBVP1D()`. The `FEMoctave` code in `BlackScholesCall.m` below generates Figure 177. In this figure read of information for a call option with strike $K = 120$.

- If the value of the stock is $S = 110$ half a year before the maturity time, then the fair value of the call option is $V(110, 0.5) \approx 2.1$.
- If the value of the stock is $S = 130$ half a year before the maturity time, then the fair value of the call option is $V(130, 0.5) \approx 15.4$.
- If the value of the stock is $S = 110$ one year before the maturity time, then the fair value of the call option is $V(110, 1) \approx 5.6$.
- If the value of the stock is $S = 130$ one year before the maturity time, then the fair value of the call option is $V(130, 1) \approx 20.5$.

[47] Ask me about a derivation of the Black–Scholes equation using elementary mathematical tools only.

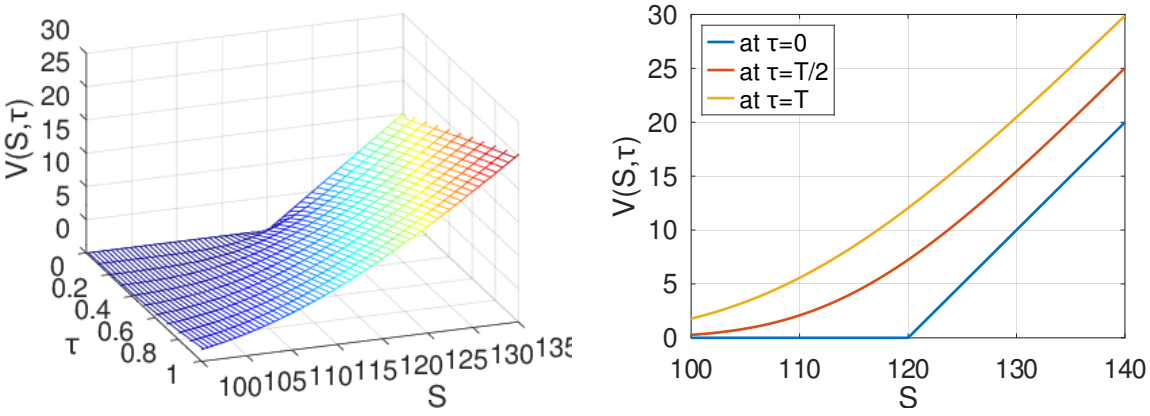


Figure 177: The value V of a European call option as function of time τ and the value of the stock S

BlackScholesCall.m

```
%% script file to solve Black-Scholes for a European call option
K = 120;          % strike
r = 0.076;        % annual gain of stock
sigma = 0.13;     % volatility
r0 = r+sigma^2/2; % safe interest rate
T = 1.0;          % maximal time to maturity
%%%%%%%%%%%%%%%
a = -0.5; b = +0.5; interval = log(K)+linspace(a,b,101)';
BCleft = 0; BCright = exp(max(interval))-K;
u0 = @(z)max(0,exp(z)-K);
[z,V,tau] = IBVP1D(interval,1,sigma^2/2,-r,+r0,0,0,BCleft,BCright,u0,0,T,[10,10]);

figure(1); mesh(tau,exp(z),V); xlabel('\tau'); ylabel('S'); zlabel('V(S,\tau)')
           xlim([0,T]); ylim([90,135]); zlim([0,30]); caxis([0,25]); view([70,30])
S = linspace(90,140,101)';
V0 = interp1(exp(z),V(:,1),S); Vend = interp1(exp(z),V(:,end),S);
t_ind = find(abs(tau-T/2)<100*eps); Vmid = interp1(exp(z),V(:,t_ind),S);
figure(2); plot(S,V0,S,Vmid,S,Vend); xlabel('S'); ylabel('V(S,\tau)')
           legend('at \tau=0','at \tau=T/2','at \tau=T','location','northwest')
```

The above results are for an European call option, but there are similar put options. A put option on a stock gives you the right (but not the obligation) to sell the stock at the fixed maturity time T for a given strike price K. If the actual price S is below the strike K you buy on the market at price S and use the put option to sell at price K. You gain $K - S$. If the actual price S is above the strike K you let the option expire. For this right to sell you have to pay a fair price V, the value of this call option. Determine the value of a put option use the Black–Scholes PDE (128) again, but with the boundary conditions

$$V(S,\tau) \approx K - S \quad \text{for } S \text{ very small and} \quad V(S,\tau) \approx 0 \quad \text{for } S \text{ very large.}$$

To respect the compatibility condition at maturity use $V(t,a) = K - S = K - e^a$ for $e^a \ll 1$. The initial conditions at $\tau = 0$ or $t = T$ are

$$V(z,0) \;=\; \max\{0, K - S\} \;=\; \max\{0, K - e^z\}\,.$$

This initial boundary value problem can be solved by `FEMoctave` with the command `IBVP1D()`. The `FEMoctave` code `BlackScholesPut.m` below generates Figure 178. In this figure read of information for a put option with strike $K = 120$.

- If the value of the stock is $S = 110$ half a year before the maturity time, then the fair value of the put option is $V(110, 0.5) \approx 7.12$.
- If the value of the stock is $S = 130$ half a year before the maturity time, then the fair value of the put option is $V(130, 0.5) \approx 0.48$.

- If the value of the stock is $S = 110$ one year before the maturity time, then the fair value of the put option is $V(110, 1) \approx 5.85$.
- If the value of the stock is $S = 130$ one year before the maturity time, then the fair value of the put option is $V(130, 1) \approx 0.76$.

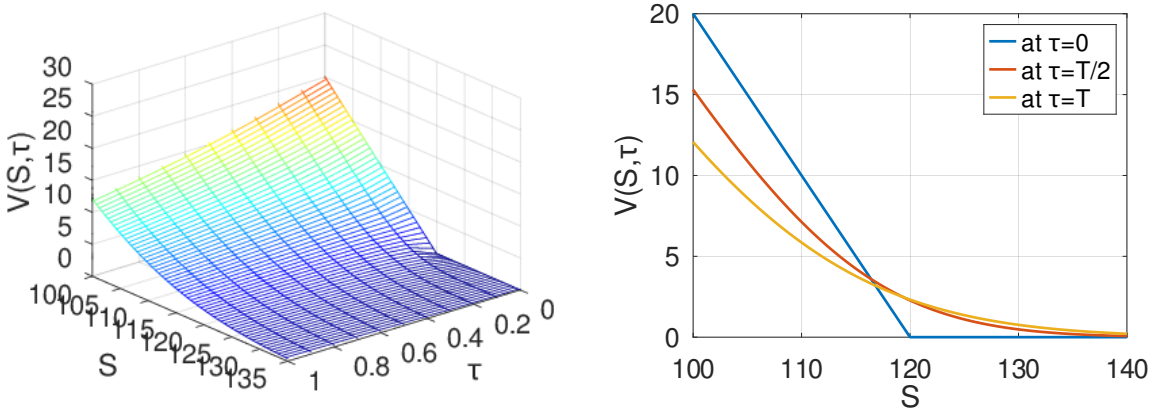


Figure 178: The value V of a European put option as function of time τ and the value of the stock S

BlackScholesPut.m

```
%% script file to solve Black-Scholes for a European put option
K = 120;          % strike
r = 0.076;        % annual gain of stock
sigma = 0.13;     % volatility
r0 = r+sigma^2/2; % safe interest rate
T = 1.0;          % maximal time to maturity
%%%%%%%%%%%%%%%
a = -0.5; b = +0.5; interval = log(K)+linspace(a,b,101)';
BCright = 0; BCleft = K-exp(min(interval));
u0 = @(z)max(0,K-exp(z));
[z,V,tau] = IBVP1D(interval,1,sigma^2/2,-r,+r0,0,0,BCleft,BCright,u0,0,T,[10,10]);

figure(1); mesh(tau,exp(z),V); xlabel('\tau'); ylabel('S'); zlabel('V(S,\tau)')
           xlim([0,T]); ylim([100,135]); zlim([0,30]); caxis([0,25]); view([140,30])
S = linspace(100,140,101)';
V0 = interp1(exp(z),V(:,1),S); Vend = interp1(exp(z),V(:,end),S);
t_ind = find(abs(tau-T/2)<100*eps); Vmid = interp1(exp(z),V(:,t_ind),S);
figure(2); plot(S,V0,S,Vmid,S,Vend); xlabel('S'); ylabel('V(S,\tau)')
           legend('at \tau=0','at \tau=T/2','at \tau=T','location','northeast')
```

The above codes use an initial boundary value problem to determine the fair price of an European call or put option. For this very simple option this is not necessary. It is possible to express the solution in terms of the error function or the cumulative distribution function `normcdf()` for the normal distribution. The *Octave* package `financial` does implement this approach.

```
pkg load financial
S = 130; tau = T/2;
[Call,Put] = blsprice(S,K,r+sigma^2/2,tau,sigma ,0);
disp(sprintf("For S = %g obtain Call = %g or put = %g at time tau = %g",S,Call,Put,tau))
-->
For S = 130 obtain Call = 15.4411 or Put = 0.479557 at time tau = 0.5
```

9.23 Spherical harmonics

In quantum mechanics the spherical harmonic functions play an essential role, see e.g. [GrifSchr04]. They are based on the Laplace operator in spherical coordinates.

$$\Delta\, u = \nabla\nabla\, u = \frac{1}{r^2}\,\frac{\partial}{\partial r}\left(r^2\,\frac{\partial\, u}{\partial r}\right) + \frac{1}{r^2\,\sin(\theta)}\,\frac{\partial}{\partial\theta}\left(\sin(\theta)\,\frac{\partial\, u}{\partial\theta}\right) + \frac{1}{r^2\,\sin^2(\theta)}\,\frac{\partial^2\, u}{\partial\varphi^2}$$

The spherical harmonics are solutions of the eigenvalue problem

$$\begin{aligned} L^2\, u &= \left(\frac{1}{\sin\theta}\,\frac{\partial}{\partial\theta}\,(\sin\theta\,\frac{\partial}{\partial\theta}) + \frac{1}{\sin^2\theta}\,\frac{\partial^2}{\partial\varphi^2}\right)\, u \\ \lambda\,\sin\theta\, u &= \frac{\partial}{\partial\theta}\,(\sin\theta\,\frac{\partial\, u}{\partial\theta}) + \frac{1}{\sin\theta}\,\frac{\partial^2\, u}{\partial\varphi^2} = \frac{\partial}{\partial\theta}\,(\sin\theta\,\frac{\partial\, u}{\partial\theta}) + \frac{\partial}{\partial\varphi}\left(\frac{1}{\sin\theta}\,\frac{\partial\, u}{\partial\varphi}\right) \end{aligned}$$

on the domain $0 < \varphi < 2\,\pi$ and $0 < \theta < \pi$. Observe that the differential equation with the independent variables φ and θ has the form

$$\begin{pmatrix} \frac{\partial}{\partial\varphi} \\ \frac{\partial}{\partial\theta} \end{pmatrix} \left(\begin{bmatrix} \frac{1}{\sin(\theta)} & 0 \\ 0 & \sin(\theta) \end{bmatrix} \begin{pmatrix} \frac{\partial\, u}{\partial\varphi} \\ \frac{\partial\, u}{\partial\theta} \end{pmatrix} \right) = \lambda\,\sin(\theta)\, u\ .$$

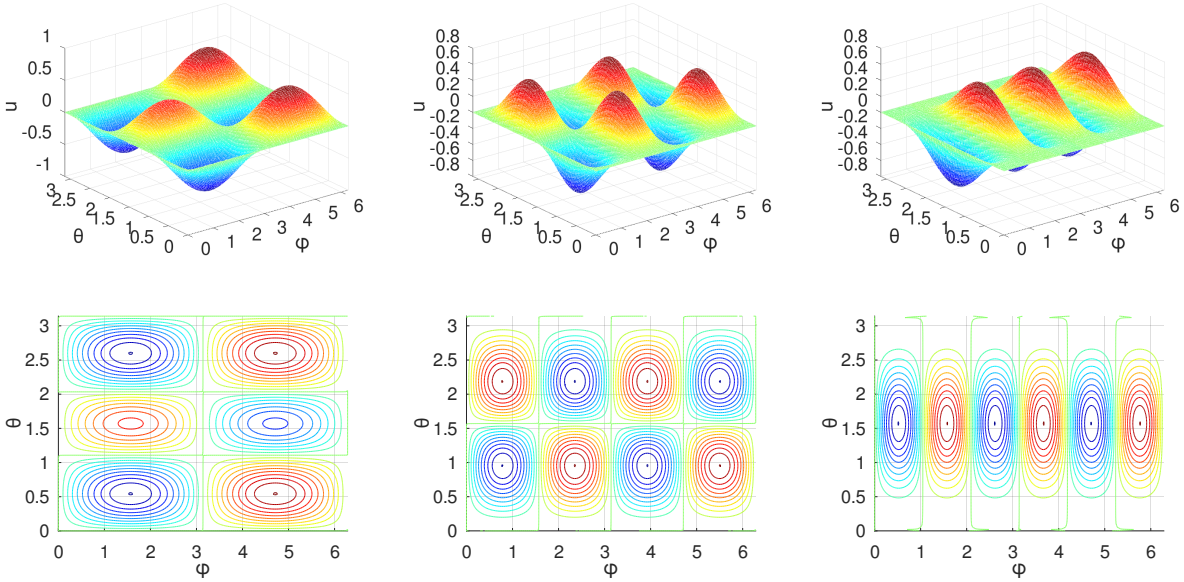
Thus the matrix

$$\mathbf{a} = \begin{bmatrix} \frac{1}{\sin(\theta)} & 0 \\ 0 & \sin(\theta) \end{bmatrix}$$

has to be given as coefficient. The boundary conditions are

$$\frac{\partial}{\partial\,\theta}\,u(\varphi,0) = \frac{\partial}{\partial\,\theta}\,u(\varphi,\pi) = 0 \quad , \quad u(0,\theta,0) = u(2\,\pi,\theta) \quad \text{and} \quad \frac{\partial}{\partial\varphi}\,u(0,\theta) = \frac{\partial}{\partial\varphi}\,u(2\,\pi,\theta)\ .$$

The implementation in `FEMoctave` can not deal with periodic boundary conditions, thus replaced by the Dirichlet conditions $u(0,\theta) = u(2\,\pi,\theta) = 0$ and the additional constraint[48] $u(\pi,\theta) \approx 0$. As consequence the smallest eigenvalue $\lambda = 0$ with the constant eigenfunction is missed. In Figure 179 find the graphs for a few spherical harmonics. The figure is generated by the code `SphericalHarmonics.m`.



(d) $l = 3$ and $m = 1$ (e) $l = 3$ and $m = 2$ (f) $l = 3$ and $m = 3$

Figure 179: The spherical harmonics for the eigenvalue $\lambda = 12 = l\cdot(l+1)$, i.e. $l = 3$ and $m = 1$, 2 and 3

- Create the mesh, define the coefficient functions and determine the eigenvalues and eigenfunctions.

SphericalHarmonics.m

[48] This leads to the correct solutions, since we know that the φ dependence is of the form $\sin(m\,\varphi + \delta)$.

```
Nphi = 50; Ntheta = 50;
phi = linspace(0,2*pi,Nphi)'; theta = linspace(0,pi,Ntheta)';
Mesh = CreateMeshRect(phi,theta,-2,-2,-1,-1);
Mesh = MeshUpgrade(Mesh,'cubic');

function res = a(PhiTheta)
  Theta = PhiTheta(:,2);
  res = [1./sin(Theta), sin(Theta), zeros(size(Theta))];
endfunction

function res = w(PhiTheta)
  res = sin(PhiTheta(:,2));
endfunction
Nval = 30;
[Eval,Evec] = BVP2Deig(Mesh,'a',0,'w',0,Nval);
```

- Select the eigenfunctions with $u(\pi, 1) \approx 0$.

SphericalHarmonics.m

```
flag = zeros(Nval,1);%% select the function different from zero at midpoint
for ii = 1:Nval
  flag(ii) = abs(FEMgriddata(Mesh,Evec(:,ii),pi,1))<1e-2;
endfor
Indices = find(flag>0);
Eval = Eval(Indices);  Evec = Evec(:,Indices); Nval = sum(flag);
```

- Display the graph and contour plot of the selected eigenfunction.

SphericalHarmonics.m

```
n = 6; u = Evec(:,n);
figure(1); FEMtrimesh(Mesh,u)
           xlabel('\phi') ; ylabel('\theta'); zlabel('u'); xlim([0,2*pi]); ylim([0,pi])
figure(2); clf; FEMtricontour(Mesh,u)
           xlabel('\phi') ; ylabel('\theta'); xlim([0,2*pi]); ylim([0,pi])
```

- Generate and display the numerical results. The value of m to detect eigenfunctions with the factor $\sin(m\,\varphi)$ is evaluated using correlation coefficients.

SphericalHarmonics.m

```
phi = linspace(0,2*pi,100);
for nn = 1:Nval
  u = FEMgriddata(Mesh,Evec(:,nn),phi,ones(size(phi)));
  for m = 1:5
    Corr = corrcoef(u,sin(m*phi));
    if abs(Corr(1,2))>0.95
    disp(['solution ',num2str(nn),' with m = ',num2str(m),...
          ' and eigenvalue = ',num2str(Eval(nn))])
    endif
  endfor
endfor
-->
solution 1 with m = 1 and eigenvalue = 2
solution 2 with m = 1 and eigenvalue = 6
solution 3 with m = 2 and eigenvalue = 6
solution 4 with m = 1 and eigenvalue = 12
solution 5 with m = 2 and eigenvalue = 12
solution 6 with m = 3 and eigenvalue = 12
solution 7 with m = 1 and eigenvalue = 20
solution 8 with m = 2 and eigenvalue = 20
solution 9 with m = 3 and eigenvalue = 20
solution 10 with m = 4 and eigenvalue = 20
solution 11 with m = 1 and eigenvalue = 30
```

```
solution 12 with m = 2 and eigenvalue = 30
solution 13 with m = 3 and eigenvalue = 30
solution 14 with m = 4 and eigenvalue = 30
solution 15 with m = 5 and eigenvalue = 30
```

The result confirms the analytical eigenvalues of $\lambda = l \cdot (l+1)$ for $1 \leq m \leq l$, see [GrifSchr04].

9.24 Schrödinger's equation of quantum mechanics

The dynamic Schrödinger equation is given by[49]

$$i\,\hbar\,\frac{\partial}{\partial t}u(x,t) = \left[-\frac{\hbar^2}{2\,m}\,\frac{\partial^2}{\partial x^2} + V(x)\right]\,u(x,t) \tag{129}$$

By choosing appropriate coordinates for length, mass and time the differential equation takes the simplified form

$$i\,\frac{\partial}{\partial t}u(x,t) = -\frac{\partial^2}{\partial x^2}\,u(x,t) + V(x)\,u(x,t)\;.$$

The function $u(x,t)$ is complex valued and the square of its norm $|u(x,t)|^2$ leads to the probability density for the location of the particle. With `FEMoctave` a few situations can be examined, since complex values of the function $u(x,t)$ are possible[50] for `IBVP1D()` with *Octave*.

9.24.1 A free particle moving and tunneling through a potential wall

For a potential of height V_0 and width W of the form

$$V(x) = \begin{cases} V_0 & \text{for } 0 < x < W \\ 0 & \text{for } x < 0 \text{ and } x > W \end{cases}$$

use an initial value

$$u_0(x) = \frac{1}{\pi^{1/4}\,\sqrt{\sigma}}\,\exp(-\frac{1}{2\,\sigma^2}\,(x-x_0)^2 + i\,\frac{v_0}{2}\,x)\,.$$

This is a Gauss shaped function centered at $x = x_0$ with width σ and initial velocity v_0. It satisfies the normlization condition

$$\int_{\mathbb{R}} |u_0(x)|^2\,dx = 1\;,$$

With the code `Tunneling.m` shown below this particle can be observed as time advances.

Tunneling.m

```
pkg load femoctave
Lm = -20; Lp = +20; BCleft = 0; BCright = 0; w = 1; a = 1; b = 0; d = 1; f = 0;
V0 = 0*21; Width = 1;
c = @(x) V0*(x<Width).*(x>0);
interval = linspace(Lm,Lp,1001)'; t0 = 0; t_end = 1.5;  steps = [150,100];
sigma = 1; Position = -5 ;  speed = 10;
u0 = @(x)exp(-0.5*sigma^(-2)*(x-Position).^2).^2; cu = 1/sqrt(quad(u0,Lm,Lp));
%% nonzero initial velocity
u0 = @(x)cu*exp(-0.5*sigma^(-2)*(x-Position).^2).*exp(i*speed/2*x);
uMax = max(u0(interval));

solver = 'RK';
[x,u,t] =
IBVP1D(interval,-w*i,a,b,c,d,f,BCleft,BCright,u0,t0,t_end,steps,'solver',solver);
figure(11); mesh(t,x,abs(u).^2); title('|u|^2')
            xlabel('time t'); ylabel('position x'); xlim([0,t_end]);
figure(12); contour(t,x,abs(u).^2,uMax*[0.1,0.2,0.5]); title('contours of |u|^2')
            xlabel('time t'); ylabel('position x')
[m,ind01]= min(abs(t-0.1)); [m,ind10] = min(abs(t-1));
```

[49] Source: Wikipedia en.wikipedia.org/wiki/Schroedinger_equation or the excellent book [GrifSchr04].
[50] No thorough testing done yet though.

```
[m,ind05] = min(abs(t-0.5)); [m,ind15] = min(abs(t-1.5));
figure(13); plot(x,abs(u(:,ind01)).^2,x,abs(u(:,ind05)).^2,...
                 x,abs(u(:,ind10)).^2,x,abs(u(:,end)).^2,'r')
            xlabel('position x'); ylabel('|u|^2')
            legend(['at t = ',num2str(t(ind01))],['at t =',num2str(t(ind05))],...
                   ['at t = ',num2str(t(ind10))],['at t = ',num2str(t(ind15))])
```

9–1 Example : Movement of a free particle
To examine the evolution of a free particle use the parameters

```
V0 = 0 ; Width = 1; sigma = 1; Position = -5 ;  speed = 10;
```

In Figure 180 observe the initial Gauss curve move with speed 10 and increasing its width as time advances. The width of the "particle" is increasing. The expression $\int_{-20}^{+20} |u(x,t)|^2 \, dx = 1$ is conserved as time advances. This has to be the case, since $|u(x,t)|^2$ is a probability density. ♢

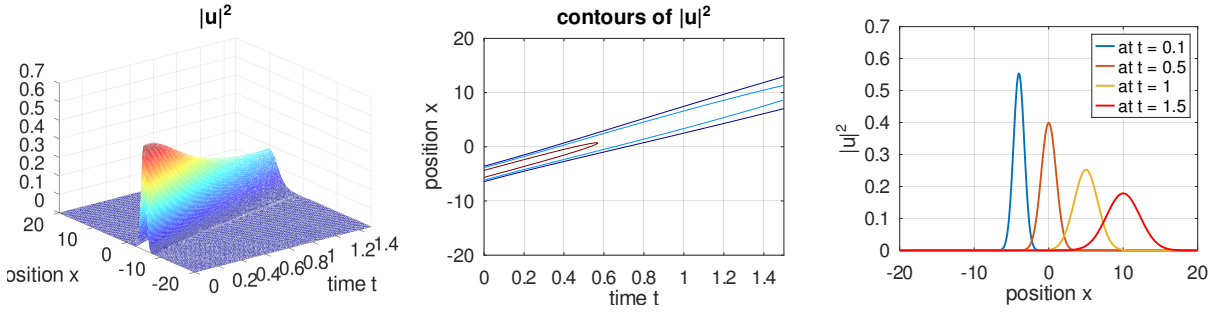


Figure 180: The movement of a free particle with speed 10 and an initial Gauss curve

9–2 Example : Partial reflection of a particle
With an intermediate value of `V0 = 21` a partial reflection is observed in Figure 181. About half of the signal will pass the potential barrier, the other half is reflected. A part of the signal is tunneling through the potential wall of height $V_0 = 21$.

```
V0 = 21; Width = 1;  sigma = 1; Position = -5 ;  speed = 10;
```

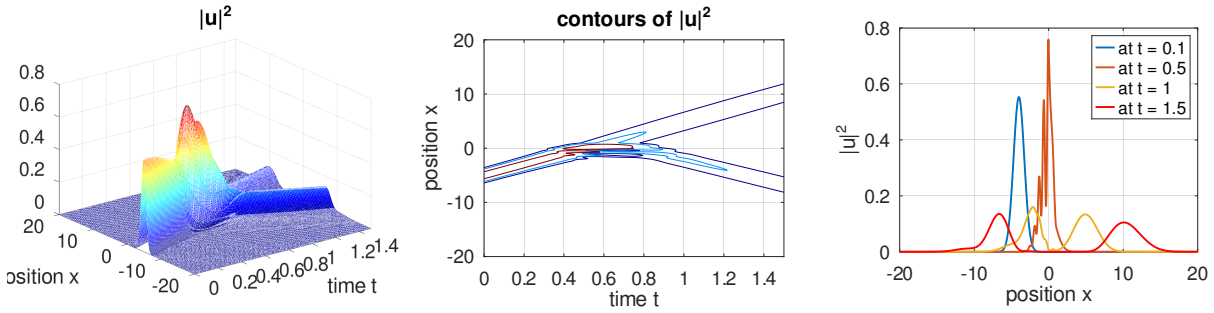


Figure 181: The movement of a particle with speed 10 and an initial Gauss curve with an intermediate size potential

♢

9–3 Example : Almost total reflection of a particle
With a larger value of `V0 = 30` an almost total reflection is observed in Figure 182. Only a very small part of the signal is tunneling through the potential wall of height $V_0 = 30$.

```
V0 = 30; Width = 1; sigma = 1; Position = -5 ;  speed = 10;
```

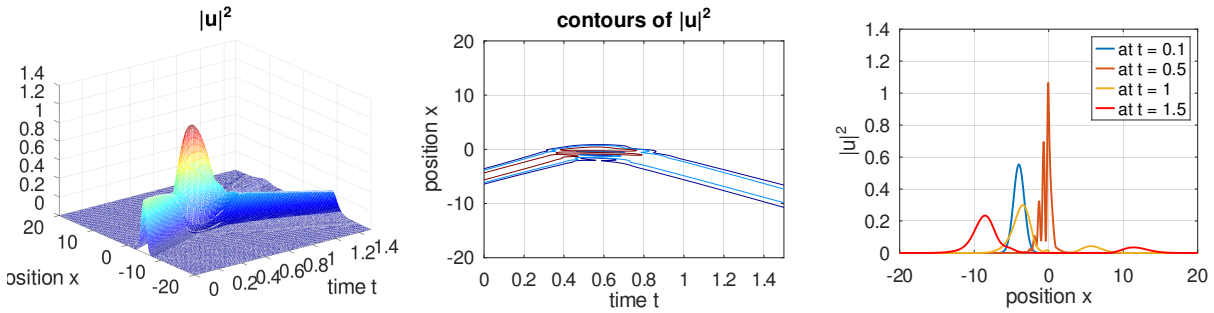


Figure 182: The movement of a particle with speed 10 and an initial Gauss curve with a stronger size potential

♢

9–4 Example : A bouncing quantum ball
The potential

$$V(x) = \begin{cases} 15\,x & \text{for } -L \le x \le +L \\ \infty & \text{for } |x| > L \end{cases}$$

leads to a constant force in the negative x–direction for $-L < x < L$ and a infinitely hard wall at $x = -L$. The initial boundary value problem to be solved is

$$\begin{aligned} i\,\tfrac{\partial}{\partial t}\,u(x,t) &= -\tfrac{\partial^2}{\partial x^2}\,u(x,t) + V(x)\,u(t,x) && \text{for } -L \le x \le +L \text{ and } t > 0 \\ u(\pm L, t) &= 0 && \text{for } t > 0 \\ u(x,0) &= u_0(x) && \text{for } -L \le x \le +L \end{aligned} \tag{130}$$

The code `SchroedingerBouncingBall.m` with $L = 10$ uses the initial values

$$\begin{aligned} u_0(x) &= \exp(-(x-6)^2) \quad \text{for zero initial momentum, or} \\ u_0(x) &= \exp(-(x-6)^2)\,\exp(-i\,2\,x) \quad \text{for nonzero initial momentum} \end{aligned}$$

The results shown in Figure 183 shows the particle starting out at $x \approx 6$, then accelerated downwards by the potential $V(x) = 15\,x$. At $x = -L = -10$ the infinite potential leads to a reflection of the bouncing particle.

SchroedingerBouncingBall.m

```
L = 10; BCleft = 0; BCright = 0; w = 1; a = 1; b = 0;  d = 1; f = 0;
c = @(x) 15*x;
interval = linspace(-L,+L,201)'; t0 = 0; t_end = 3.5;  steps = [400,50];
coeff = 1; Position = -0.6*L ;
u0 = @(x)exp(-coeff*(x+Position).^2).^2; cu = 1/sqrt(quad(u0,-L,L));
u0 = @(x)1*cu*exp(-coeff*(x+Position).^2).*exp(-i*2*x);%% nonzero initial velocity
u0 = @(x)1*cu*exp(-coeff*(x+Position).^2);             %% zero initial velocity

[x,u,t] = IBVP1D(interval,-w*i,a,b,c,d,f,BCleft,BCright,u0,t0,t_end,steps,...
                 'solver','RK');

figure(11); mesh(t,x,real(u)); xlabel('time t'); ylabel('position x');
            xlim([0,t_end]);  title('real(u)')
figure(12); mesh(t,x,imag(u)); xlabel('time t'); ylabel('position x');
            xlim([0,t_end]);   title('imag(u)')
```

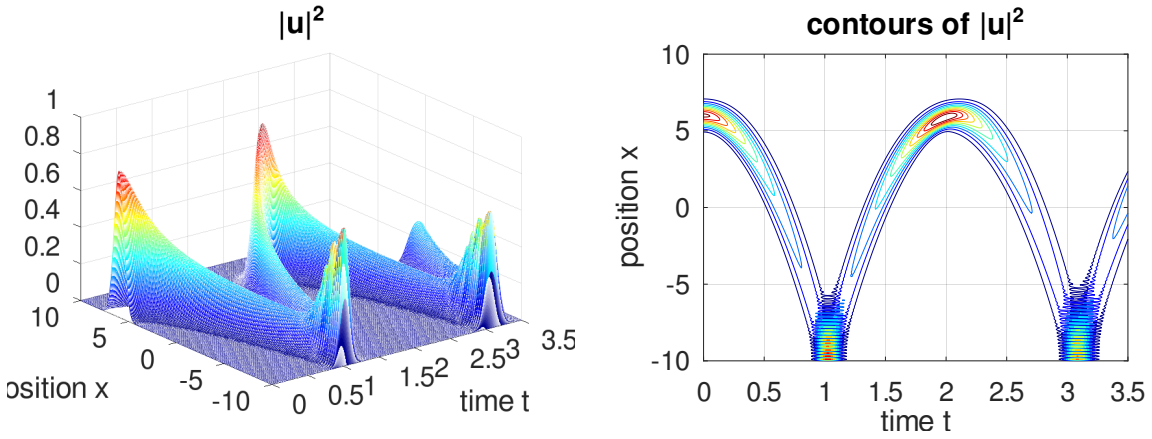


Figure 183: The time evolution of the Schrödinger equation for a bouncing ball

```
figure(13); mesh(t,x,abs(u).^2); xlabel('time t'); ylabel('position x');
            xlim([0,t_end]);  title('|u|^2')
figure(14); contour(t,x,abs(u).^2); xlabel('time t'); ylabel('position x')
            title('contours of |u|^2')
figure(15); plot(x,real(u(:,end)),'b',x,imag(u(:,end)),'g',...
                 x,abs(u(:,end)).^2,'r')
            xlabel('position x'); title(['at t = ',num2str(t_end)])
            legend('real(u)','imag(u)','|u|^2')
```

♢

9.24.2 Schrödinger's harmonic oscillator

For the dynamic Schrödinger equation (129)

$$i\,\hbar\,\frac{\partial}{\partial t}\Psi(x,t) = \left[-\frac{\hbar^2}{2\,m}\,\frac{\partial^2}{\partial x^2} + V(x)\right]\,\Psi(x,t)\;.$$

search for solutions of the form $\Psi(x,t) = f(t)\,\psi(x)$ and use the tool of separation of variables.

$$\begin{aligned} i\,\hbar\,\frac{d}{dt}\,f(t)\,\psi(x) &= \left[-\frac{\hbar^2}{2\,m}\,\frac{\partial^2}{\partial x^2} + V(x)\right]\,f(t)\,\psi(x) \\ E \;=\; i\,\hbar\,\frac{\frac{d}{dt}\,f(t)}{f(t)} &= \frac{1}{\psi(x)}\,\left[-\frac{\hbar^2}{2\,m}\,\frac{\partial^2}{\partial x^2} + V(x)\right]\,\psi(x) \\ \frac{d}{dt}\,f(t) &= \frac{E}{i\,\hbar}\,f(t) \quad\Longrightarrow\quad f(t) \;=\; C\,\exp(-i\frac{E}{\hbar}\,t)\;. \end{aligned}$$

With the potential $V(x) = \frac{1}{2}\,m\,\omega^2\,x^2$ for a harmonic oscillator Schrödinger's eigenvalue equation is given by

$$-\frac{\hbar^2}{2\,m}\,\frac{\partial^2\,\psi}{\partial x^2} + \frac{1}{2}\,m\,\omega^2\,x^2\,\psi = E\,\psi\;. \tag{131}$$

A rescaling with $z = \sqrt{\frac{\hbar}{m\,\omega}}\,x$ and $u(x) := \psi(z) = \psi(\sqrt{\frac{\hbar}{m\,\omega}}\,x)$ leads to

$$\begin{aligned} -\frac{\partial^2\,u(x)}{\partial x^2} + x^2\,u(x) &= -\frac{\partial^2\,\psi(z)}{\partial z^2}\,\frac{\hbar}{m\,\omega} + \frac{m\,\omega}{\hbar}\,z^2\,\psi(z) \\ &= \frac{2}{\omega\,\hbar}\,\left(-\frac{\hbar^2}{2\,m}\,\frac{\partial^2\,\psi(z)}{\partial z^2} + \frac{m\,\omega^2}{2}\,z^2\,\psi(z)\right) \;=\; \frac{2}{\omega\,\hbar}\,E\,\psi(z) \;=\; \lambda\,u(x)\;. \end{aligned}$$

Thus the eigenvalue problem to be solved is

$$-\frac{\partial^2}{\partial x^2}\,u(x) + x^2\,u(x) = \lambda\,u(x) \quad \text{with} \quad u(\pm\infty) = 0\;.$$

Then the eigenvalues of the Schrödinger equation (131) are given by $E = \frac{\omega\,\hbar}{2}\,\lambda$.

The code below uses the command `BVP1Deig()` to approximate the first six eigenvalues. The numerical result confirms that the eigenvalues of the Schrödinger equation (131) are given by

$$E_n = \frac{\omega\,\hbar}{2}\,\lambda_n = \frac{\omega\,\hbar}{2}\,(2n+1) \quad \text{for} \quad n = 0, 1, 2, 3, 4, \ldots$$

Figure 184 shows the shape of the corresponding eigenfunctions $u_n(x)$. The exact formulas for the eigenfunctions use the physicist's Hermite polynomials and exponential functions[51].

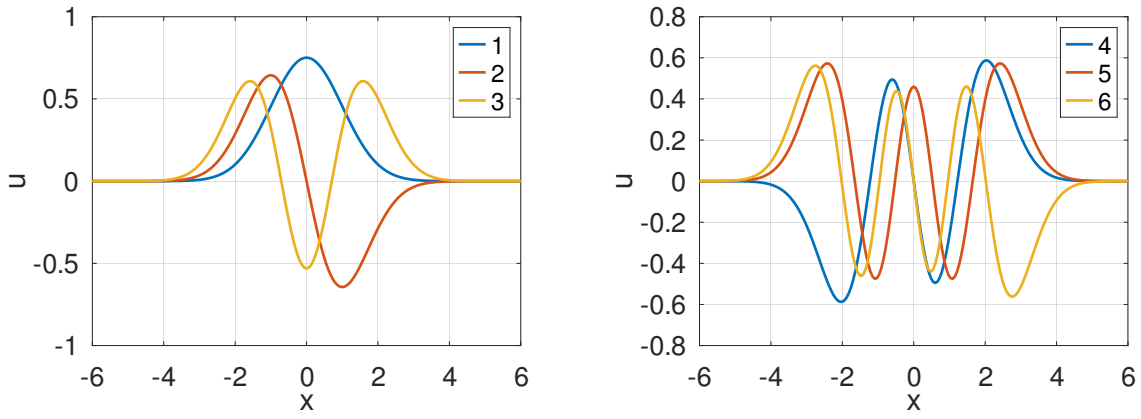
The probability density function for the location of the particle is given by the square of the eigenfunction. Find the graphs for the PDF (probability density function) in Figure 185.

SchroedingerHarmonic.m

```
x_max = 6; interval = linspace(-x_max,x_max,100)';
BCleft = 0; BCright = 0;
[x, eVal, eVec] = BVP1Deig(interval,1,0,@(x)x.^2,1,BCleft,BCright,6);

Eigenvalues = eVal'
figure(1); plot(x,eVec(:,1:3)); xlabel('x'); ylabel('u'); xlim([-x_max,+x_max])
           legend('1','2','3')
figure(2); plot(x,eVec(:,4:6)); xlabel('x'); ylabel('u'); xlim([-x_max,+x_max])
           legend('4','5','6')

figure(11); plot(x,eVec(:,1:3).^2); xlabel('x'); ylabel('u'); xlim([-x_max,+x_max])
           legend('1','2','3'); xlim(3.5*[-1,1])
figure(12); plot(x,eVec(:,4:6).^2); xlabel('x'); ylabel('u'); xlim([-x_max,+x_max])
          legend('4','5','6'); xlim(3.5*[-1,1])
-->
Eigenvalues = 1.0000  3.0000  5.0000 7.0000 9.0001 11.0001
```



(a) the first three eigenfunctions

(b) the next three eigenfunctions

Figure 184: The first six eigenfunctions of the harmonic Schrödinger operator

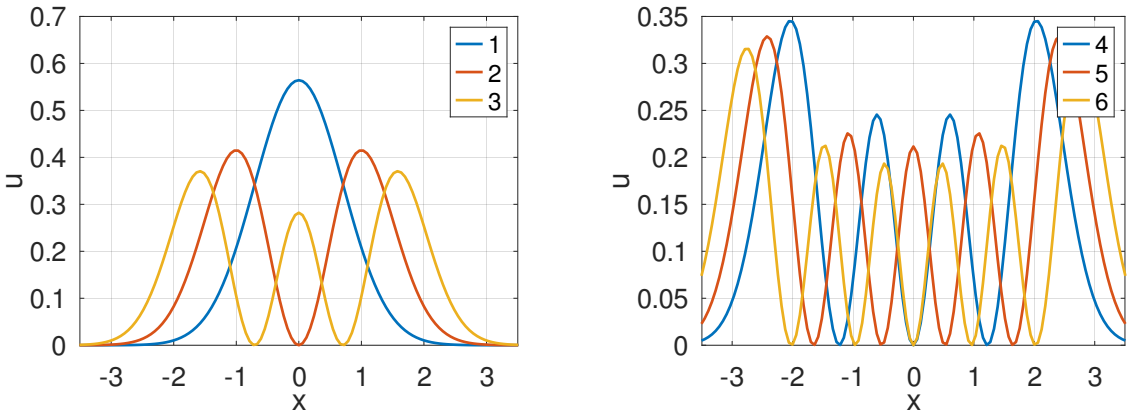
9.24.3 Energies of the hydrogen atom

For the hydrogen atom the Schrödinger equation is

$$i\,\hbar\,\frac{\partial}{\partial t}\,\Psi(r,t) = -\frac{\hbar^2}{2\,m}\,\frac{1}{r^2}\,\frac{\partial}{\partial r}\left(r^2\,\frac{\partial}{\partial r}\,\Psi(r,t)\right) - \frac{e^2}{r}\,\Psi(r,t)$$

51

$$\psi_n(x) = \frac{1}{\sqrt{2^n\,n!}}\left(\frac{m\,\omega}{\pi\,\hbar}\right)^{1/4} e^{-\frac{m\,\omega\,x^2}{2\,\hbar}}\,H_n\!\left(\sqrt{\frac{m\,\omega}{\hbar}}\,x\right) \quad \text{where} \quad H_n(z) = (-1)^n\,e^{(z^2)}\,\frac{d^n}{dx^n}\left(e^{(-z^2)}\right)$$



(a) the first three modes (b) the next three modes

Figure 185: The PDF for the first six modes of the harmonic Schrödinger operator

with $e^2 = \frac{q^2}{4\,\pi\,\varepsilon_0}$. This leads to the eigenvalue problem

$$-\frac{\partial}{\partial r}\left(r^2\,\frac{\partial}{\partial r}\,\psi(r)\right) - \frac{2\,m\,q^2}{4\,\pi\,\varepsilon_0\,\hbar^2}\,r\,\psi(r) = r^2\,\frac{2\,m}{\hbar^2}\,E\,\psi(r) = \lambda\,r^2\,\psi$$

The energy functional F leading to this equation is given by

$$F = \frac{1}{2}\int_0^\infty r^2\,\left(\frac{\partial\,\psi(r)}{\partial r}\right)^2 - \frac{2\,m\,q^2}{4\,\pi\,\varepsilon_0\,\hbar^2}\,r\,\left(\psi(r)\right)^2 - r^2\,\lambda\,\left(\psi(r)\right)^2\,dr$$

and the Euler–Lagrange equation is

$$\begin{aligned}
\frac{d}{dr}\frac{\partial\,F}{\partial\psi'} &= \frac{\partial\,F}{\partial\psi}\\
\frac{d}{dr}\left(r^2\,\frac{\partial\,\psi}{\partial r}\right) &= -\frac{2\,m\,q^2}{4\,\pi\,\varepsilon_0\,\hbar^2}\,r\,\psi(r) - \lambda\,r^2\,\psi\\
-\frac{d}{dr}\left(r^2\,\frac{\partial\,\psi}{\partial r}\right) - \frac{2\,m\,q^2}{4\,\pi\,\varepsilon_0\,\hbar^2}\,r\,\psi(r) &= +\lambda\,r^2\,\psi(r)
\end{aligned}$$

Use the Bohr radius $a_0 = \frac{4\,\pi\,\varepsilon_0\,\hbar^2}{m\,q^2}$ and substituting $v(\rho) = v(2\,r/a_0) = r\,\psi(r)$ leads to

$$\begin{aligned}
-\frac{\partial^2}{\partial\rho^2}\,v(2\,r/a_0) - \frac{1}{2\,r/a_0}\,v(2\,r/a_0) &= \frac{(4\,\pi\,\varepsilon_0)^2\,\hbar^2}{2\,m\,q^4}\,E\,v(2\,r/a_0) = a_0^2\,\frac{m}{2\,\hbar^2}\,E\,v(2\,r/a_0)\\
-\frac{\partial^2}{\partial\rho^2}\,v(\rho) - \frac{1}{\rho}\,v(\rho) &= a_0^2\,\frac{m}{2\,\hbar^2}\,E\,v(\rho) = \lambda\,v(\rho)\\
E &= \frac{2\,\hbar^2}{m\,a_0^2}\,\lambda = \frac{2\,\hbar^2\,m^2\,q^4}{m\,(4\,\pi\,\varepsilon_0)^2\,\hbar^4}\,\lambda = \frac{2\,m\,q^4}{(4\,\pi\,\varepsilon_0)^2\,\hbar^2}\,\lambda
\end{aligned}$$

The first eigenfunction is a minimizer of the functional

$$F(v) = \int_0^\infty \frac{1}{2}\,\left(\frac{\partial\,v}{\partial\rho}\right)^2 - \frac{1}{2\,\rho}\,v^2\,d\rho \quad \text{subject to the constraint} \quad \int_0^\infty v^2(\rho)\,d\rho = 1$$

The boundary conditions are $v(0) = 0$ (use $v(\rho) = r\,\psi(r)$) and $v(\infty) = 0$. This eigenvalue problem is solved numerically with the help of `FEMoctave`.

HydrogenEigen.m

```
n = 6; %% number of eigenvalues
L = 200; interval = linspace(0,L,1000);
```

```
shift = 2;
a = 1; b = 0; c = @(x)-1./x+shift; w = 1;
[x,EVal,EVec] = BVP1Deig(interval,a,b,c,w,0,0,n);

EVal = EVal-shift;
OneOverEvalues = 1./EVal

[MaxVal,MaxInd] = max(abs(EVec));
Sign = zeros(1,n);
for ii=1:n
  Sign(ii) = sign(EVec(MaxInd(ii),ii));
endfor

r = x/2; v1 = abs(EVec(:,1)./x);
figure(1); plot(r,EVec*diag(Sign))
           xlabel('r/a_0'); ylabel('v(r)'); xlim([0,30])
figure(2); plot(r,log(v1))
           xlabel('r/a_0'); ylabel('ln(v_1(r)/r)'); xlim([0,30])
-->
OneOverEvalues =    -4.0000
                   -16.0000
                   -36.0000
                   -64.0000
                  -100.0002
                  -144.2120
```

The numerical values confirm the analytical result

$$-E_n = \frac{2\,m\,q^4}{(4\,\pi\,\varepsilon_0)^2\,\hbar^2}\,\frac{1}{(2\,n)^2} = \frac{m\,q^4}{2\,(4\,\pi\,\varepsilon_0)^2\,\hbar^2}\,\frac{1}{n^2} = \mathrm{Ry}\,\frac{1}{n^2}$$

with the Rydberg constant Ry. Find the graphs of the first six eigenfunctions on the left in Figure 186. The straight line with slope -1 in the graph on the right confirms

$$\Psi_{100}(\frac{r}{r_0}) = \frac{v_1(2\,\frac{r}{a_0})}{r/a_0} = c_1\,e^{-r/a_0} \qquad \Longrightarrow \qquad \ln(\Psi_{100}(\frac{r}{a_0})) = c_2 - \frac{r}{a_0}\,.$$

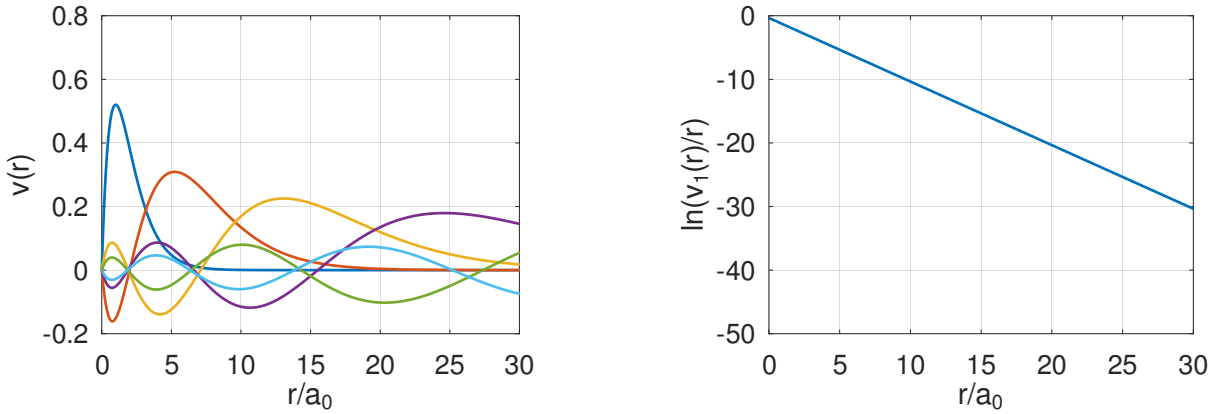


Figure 186: Graphs of a few eigenfunctions of the hydrogen atom

The same problem can be examined for eigenfunctions depending on $\rho = \sqrt{x^2 + y^2}$ and z, i.e. use cylindrical coordinates. The eigenvalue problem for the hydrogen atom in cylindrical coordinates is given by

$$\lambda\,\Psi \quad = \quad -\frac{\hbar^2}{2\,m}\,\nabla^2\,\Psi + V\,\Psi$$

$$= -\frac{\hbar^2}{2\,m}\left(\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\frac{\partial\,\Psi}{\partial\rho})+\frac{1}{\rho^2}\frac{\partial^2\,\Psi}{\partial\varphi^2}+\frac{\partial^2\,\Psi}{\partial z^2}\right)-\frac{q^2}{4\,\pi\,\varepsilon_0}\frac{1}{r}\,\Psi$$

Assuming that the solution does not depend on the azimuthal angle $0 \le \varphi \le 2\,\pi$ this leads to

$$\begin{aligned}
\lambda\,\Psi &= -\frac{\hbar^2}{2\,m}\left(\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\frac{\partial\,\Psi}{\partial\rho})+\frac{\partial}{\partial\,z}\frac{\partial\,\Psi}{\partial\,z}\right)-\frac{q^2}{4\,\pi\,\varepsilon_0}\frac{1}{\sqrt{\rho^2+z^2}}\,\Psi\\
\lambda\,\rho\,\frac{2\,m}{\hbar^2}\,\Psi &= -\left(\frac{\partial}{\partial\rho}(\rho\frac{\partial\,\Psi}{\partial\rho})+\frac{\partial}{\partial z}(\rho\frac{\partial\,\Psi}{\partial\,z})\right)-\frac{q^2}{4\,\pi\,\varepsilon_0}\frac{2\,m}{\hbar^2}\frac{\rho}{\sqrt{\rho^2+z^2}}\,\Psi\\
&= -\left(\frac{\partial}{\partial\rho}(\rho\frac{\partial\,\Psi}{\partial\rho})+\frac{\partial}{\partial z}(\rho\frac{\partial\,\Psi}{\partial\,z})\right)-2\,a_0\frac{\rho}{\sqrt{\rho^2+z^2}}\,\Psi
\end{aligned}$$

On a semicircular domain $\rho > 0$ and $\sqrt{\rho^2+z^2} \le R$ use the boundary conditions $\frac{\partial\,\Psi(0,z)}{\partial\rho} = 0$ and $\Psi(\rho,z) = 0$ for $\sqrt{\rho^2+z^2} = R$. This boundary value problem can be solved numerically with the help of `FEMoctave`, leading to the results in Figures 187 and 188. The computational domain has to be rather large ($R = 35$) to assure that the numerical solutions are close to the analytical solutions. For large radii the solutions display very little variation and thus a coarse mesh can be used. With the command `MeshDeform()` a mesh with variable size triangles is generated.

HydrogenCylindrical.m

```
global R
R = 35; N = 50; alpha = linspace(-pi/2,pi/2,N)';
Rshow = 15;
MeshContour =  [cos(alpha),sin(alpha),-ones(size(alpha))];
MeshContour(N,3) = -2;
FEMmesh = CreateMeshTriangle('circle',MeshContour,0.25e-3);  %% good value with R = 35
function xy_new = Deform(xy)
  global R; CC = 5;
  r = sqrt(xy(:,1).^2 + xy(:,2).^2);
  xy_new = xy.*exp(CC*r)/exp(CC)*R;
endfunction
FEMmesh = MeshDeform(FEMmesh,'Deform');  FEMmesh = MeshUpgrade(FEMmesh,'cubic');

function res = rho(rho_z)
  res = rho_z(:,1);
endfunction

function res = b0(rho_z)
  res = -2*rho_z(:,1)./sqrt(rho_z(:,1).^2 + rho_z(:,2).^2);
endfunction

Nval = 6;
[Eval,Evec] = BVP2Deig(FEMmesh,'rho','b0','rho',0,Nval,'mode',-1);
Eigenvalues = Eval
Ratios = Eigenvalues(1)./Eigenvalues

%% force the maximal absolute value positive
[Max,Ind] = max(abs(Evec)); Sign = zeros(Nval,1);
for ii=1:Nval
  Sign(ii) = sign(Evec(Ind(ii),ii));
endfor
Evec = Evec*diag(Sign);

n = 4;  u = Evec(:,n);
figure(1); FEMtrimesh(FEMmesh,u)
           xlabel('x  [a_0]') ; ylabel('z  [a_0]'); zlabel('u')
           xlim([0,Rshow]); ylim([-Rshow,+Rshow])
figure(2); clf; FEMtricontour(FEMmesh,u)
           xlabel('x  [a_0]') ; ylabel('z  [a_0]'); axis equal
           xlim(1.5*[0,Rshow]); ylim(0.7*[-Rshow,+Rshow])
r = sqrt(FEMmesh.nodes(:,1).^2+FEMmesh.nodes(:,2).^2);
figure(3); FEMtrimesh(FEMmesh,r.*u.^2)
```

```
            xlabel('x  [a_0]') ; ylabel('z  [a_0]'); zlabel('PDF')
            xlim([0,Rshow]); ylim([-Rshow,+Rshow])
x = linspace(0,R,200); u_x = FEMgriddata(FEMmesh,u,zeros(size(x)),x);

figure(4); plot(x,u_x);  xlabel('\rho  [a_0]'); ylabel('u')
```

9.24.4 Ground state energy of a helium atom

The Hamiltonian for a helium atom with a nucleaus and two electrons is given by

$$H = \left\{ -\frac{\hbar^2}{2\,m}\,\nabla_1^2 - \frac{1}{4\,\pi\,\varepsilon_0}\,\frac{2\,q^2}{r_1} \right\} + \left\{ -\frac{\hbar^2}{2\,m}\,\nabla_2^2 - \frac{1}{4\,\pi\,\varepsilon_0}\,\frac{2\,q^2}{r_2} \right\} + \frac{1}{4\,\pi\,\varepsilon_0}\,\frac{q^2}{\|\vec{r}_1 - \vec{r}_2\|}\ .$$

The state function $\psi(\vec{r}_1, \vec{r}_2)$ describes the configuration of the two electrons. The ground state is characterized as minimizer of the energy functional

$$E = \langle \psi, H\,\psi \rangle_{\mathbb{R}^6} \quad \text{subject to the constraint} \quad \|\psi\|^2_{\mathbb{R}^6} = 1\ .$$

Assume that the state function depends on the two radii r_1 and r_2 and use the Bohr radius a_0 for the modified function $u(\rho_1, \rho_2)$ determined by

$$\psi(r_1, r_2) = \frac{u(2\,r_1/a_0, 2\,r_2/a_0)}{r_1\,r_2}\ .$$

One can show that[52] the total energy is given by

$$\begin{aligned} E(u) &= \langle \psi, H\psi \rangle_{\mathbb{R}^6} \;=\; E_{kin} + E_{pot} + E_{coup} \\ &= \frac{\hbar^2}{2\,m}\,(4\,\pi)^2 \int_{\rho_2=0}^{\infty}\int_{\rho_1=0}^{\infty} \left(\frac{\partial\,u(\ldots)}{\partial\rho_1}\right)^2 + \left(\frac{\partial\,u(\ldots)}{\partial\rho_2}\right)^2\,d\rho_1\,d\rho_2 + \\ &\quad + \frac{\hbar^2}{2\,m}\,(4\,\pi)^2 \int_0^{\infty}\int_0^{\infty} \frac{\min\{\rho_1,\rho_2\} - 2\,(\rho_1+\rho_2)}{\rho_1\,\rho_2}\,u^2(\rho_1,\rho_2)\,d\rho_1\,d\rho_2 \\ &= \frac{\hbar^2(4\,\pi)^2}{2\,m}\left(\int_0^{\infty}\int_0^{\infty} \|\nabla u(\rho_1,\rho_2)\|^2 + \frac{\min\{\rho_1,\rho_2\} - 2\,(\rho_1+\rho_2)}{\rho_1\,\rho_2}\,u^2(\rho_1,\rho_2)\,d\rho_1\,d\rho_2 \right) \end{aligned} \tag{132}$$

The minimizer of this functional with the constraint $\|\psi\|_{\mathbb{R}^6}$ is determined by the first eigenvalue of the boundary value problem

$$-\Delta u(\rho_1,\rho_2) + \frac{\min\{\rho_1,\rho_2\} - 2\,(\rho_1+\rho_2)}{\rho_1\,\rho_2}\,u(\rho_1,\rho_2) = \lambda\,u(\rho_1,\rho_2) \quad \text{with Dirichlet BC}$$

on the domain $0 < \rho_1, \rho_2 < \infty$, to be replaced by $0 < \rho_1, \rho_2 < R$ for a large value of R. The coefficient for the potential function of the eigenvalue problem is

$$\frac{\min\{\rho_1,\rho_2\} - 2\,(\rho_1+\rho_2)}{\rho_1\,\rho_2} = \begin{cases} -\frac{\rho_1+2\,\rho_2}{\rho_1\,\rho_2} = -\frac{2}{\rho_1} - \frac{1}{\rho_2} & \text{if } \rho_1 \le \rho_2 \\ -\frac{2\,\rho_1+\rho_2}{\rho_1\,\rho_2} = -\frac{1}{\rho_1} - \frac{2}{\rho_2} & \text{if } \rho_1 \ge \rho_2 \end{cases}$$

The `FEMoctave` shown below leads to an approximate value of $\lambda_1 \approx -1.4395$ for the first eigenvalue and the corresponding ground state energy

$$E_1 = \frac{2\,\hbar^2}{m\,a_0^2}\,\lambda_1 = 4\,E_0\,\lambda_1 \approx -4 \cdot 13.605 \cdot 1.4395\ \text{eV} \approx -78.342\ \text{eV}\ .$$

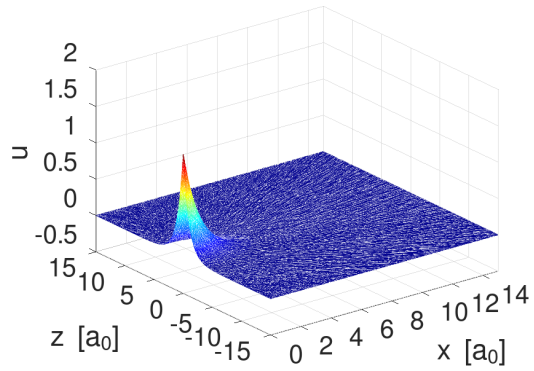
This is rather close to the experimental value of approximately -78.975 eV.

HeliumGroundState.m

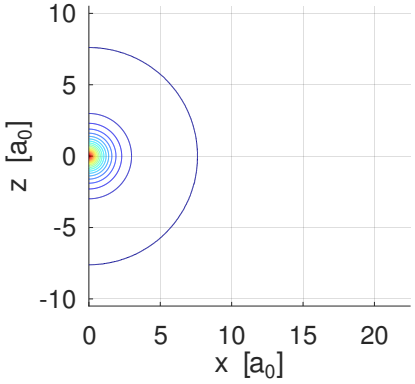
```
R = 100; N = 40;
grid = exp(4*linspace(0,1,N))-1; grid = R*grid/max(grid);

FEMmesh = CreateMeshRect(grid,grid,-1,-1,-1,-1);
if 0  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
```

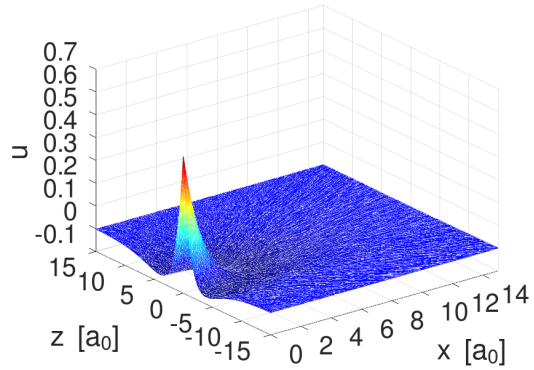
[52] More details available on request, or use standard literature, e.g. [GrifSchr04].



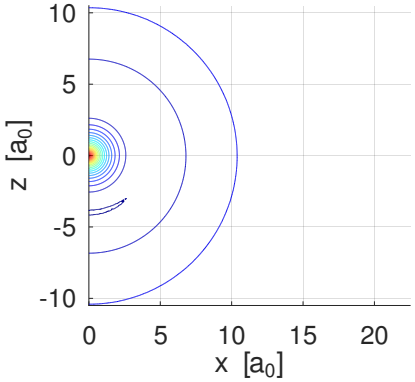
(a) $\psi_{100} = e^{-r}$



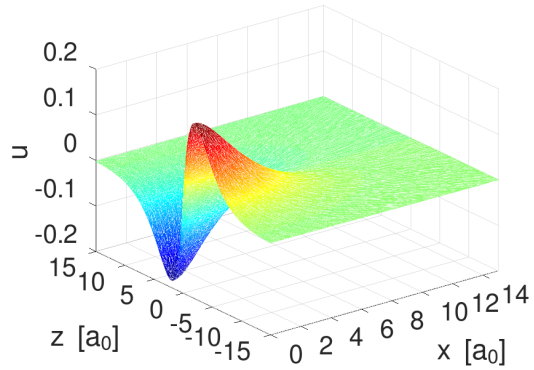
(b) contour of the first eigenfunction



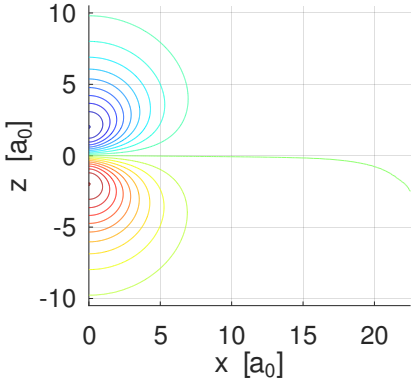
(c) $\psi_{200} = e^{-r/2}\,(2 - r)$



(d) contour of the second eigenfunction

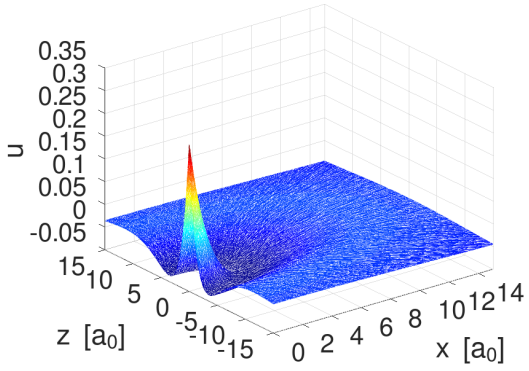


(e) $\psi_{210} = e^{-r/2}\,r\,\cos(\theta)$

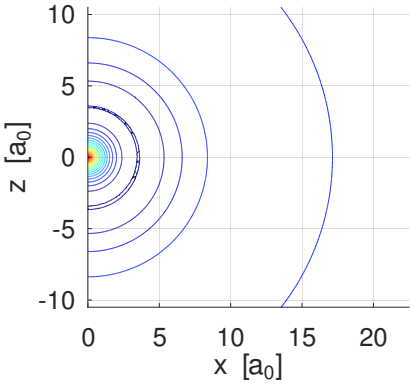


(f) contour of the third eigenfunction

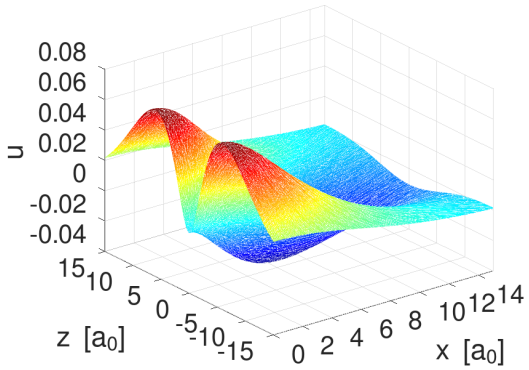
Figure 187: Graphs and contours of the first three eigenfunction of the hydrogen atom



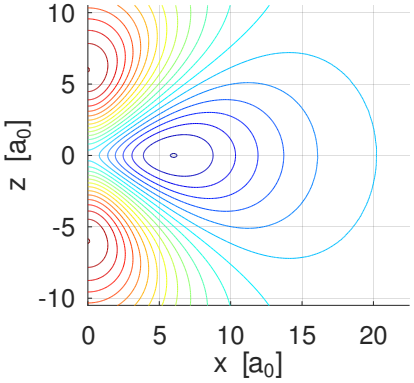
(a) $\psi_{300} = e^{-r/3}\,(1 - \frac{2}{3}\,r + \frac{2}{27}\,r^2)\,(3\,\cos(\theta) - 1)$



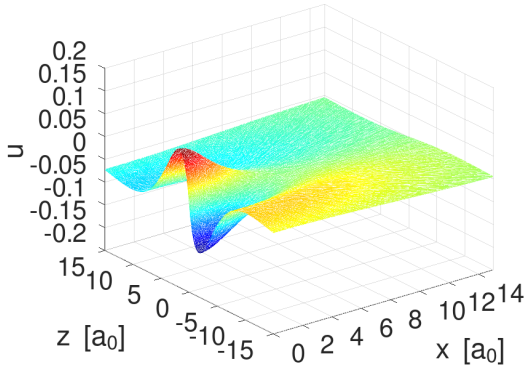
(b) contour of the fourth eigenfunction



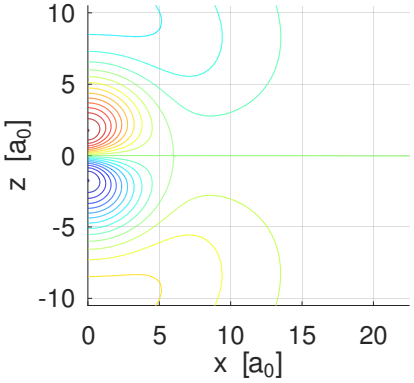
(c) $\psi_{320} = e^{-r/3}\,r^2\,(3\,\cos(\theta) - 1)$



(d) contour of the fifth eigenfunction

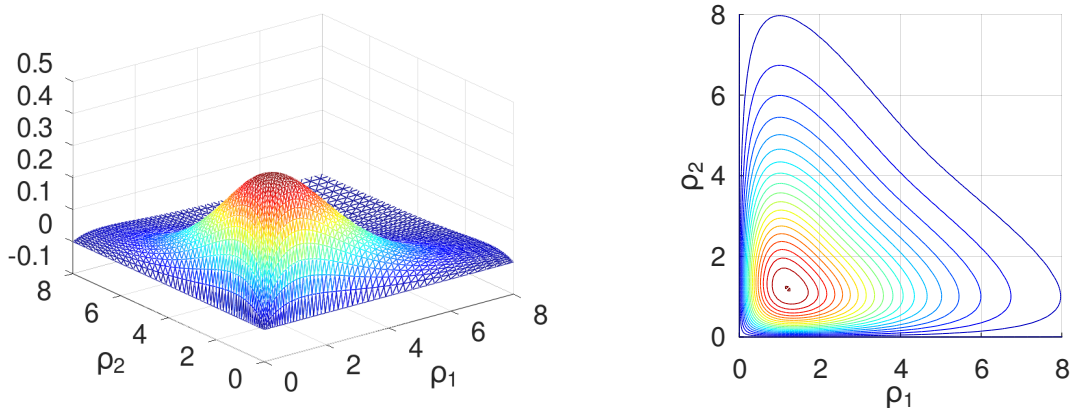


(e) $\psi_{310} = e^{-r/3}\,(6\,r - r^2)\,\cos(\theta)$



(f) contour of the sixth eigenfunction

Figure 188: Graphs and contours of fourth, fifth and sixth eigenfunction of the hydrogen atom



(a) the surface of $u(\rho_1, \rho_2)$ (b) the contour plot

Figure 189: Surface and contour plot for the helium ground state.
Displayed is the function $u(\rho_1, \rho_2) = u(2\,r_1/a_0, 2\,r_2/a_0) = r_1\,r_2\,\psi(r_1, r_2)$

```
else  FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endif

a = 1; b = @(xy) (1*min(xy,[],2)-2*sum(xy,2))./(xy(:,1).*xy(:,2));
%% eigenvalues closest to -2, i.e. most negative
[Eval,Evec] = BVP2Deig(FEMmesh,a,b(FEMmesh.GP),1,0,2,'mode',-2);
Eval

figure(1); FEMtrimesh(FEMmesh,Evec(:,1));       xlabel('\rho_1'); ylabel('\rho_2')
           xlim([0,8]); ylim([0,8])
figure(2); FEMtricontour(FEMmesh,Evec(:,1),21); xlabel('\rho_1'); %%ylabel('\rho_2')
           xlim([0,8]); ylim([0,8]); axis equal
```

Since the solution is symmetric about the axis $\rho_1 = \rho_2$ one can use a smaller computational domain $0 < \rho_2 < \rho_1 < R$, a triangle with corners at $(0,0)$, $(R,0)$ and (R,R). Along the axis $\rho_1 = \rho_2$ use the Neumann boundary condition $\frac{\partial}{\partial n}\,u(\rho,\rho) = 0$. On this domain the BVP is slightly simpler.

$$-\Delta u(\rho_1, \rho_2) - \frac{2\,\rho_1 + \rho_2}{\rho_1\,\rho_2}\,u(\rho_1, \rho_2) = \lambda\,u(\rho_1, \rho_2)\;.$$

The final result does not change, i.e. the same eigenvalue and the images show the lower right half of Figure 189.

HeliumGroundStateTriangle.m

```
N = 40;                          %% select size of grid
function xy_new = Deform(xy)  %% smaller elements close to the origin
  xy_new = exp(4*xy)-1;
  xy_new = xy_new/max(xy_new(:))*100;  %% R = 100;
endfunction
FEMmesh = CreateMeshTriangle('Tria',[0,0,-1;1,0,-1;1,1,-2],0.4*(1/N)^2);
FEMmesh = MeshDeform(FEMmesh,'Deform');
if 0  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
else  FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endif

a = 1; b = @(xy) -(2*xy(:,1)+xy(:,2))./(xy(:,1).*xy(:,2));
[Eval,Evec] = BVP2Deig(FEMmesh,a,b(FEMmesh.GP),1,0,1,'mode',-2);
Eval

[Max,MaxInd] = max(abs(Evec)); Evec = sign(Evec(MaxInd))*Evec; %% assure positive maximal value
figure(101); FEMtrimesh(FEMmesh, Evec);       xlabel('\rho_1'); ylabel('\rho_2')
```

```
              xlim([0,8]); ylim([0,8])
  figure(102); FEMtricontour(FEMmesh, Evec,21); xlabel('\rho_1'); ylabel('\rho_2')
              xlim([0,8]); ylim([0,8]); axis equal
```

9.25 The EIT forward problem

For a conductivity σ on a bounded domain $\Omega \subset \mathbb{R}^2$ consider the PDE

$$\nabla \cdot (\sigma \nabla u) = 0 \quad \text{in} \quad \Omega \subset \mathbb{R}^2 \tag{133}$$

- Apply a voltage u on the boundary and measure the resulting current density J

  $$J(z) = \sigma(z) \frac{\partial u(z)}{\partial n} \quad \text{for} \quad z \in \partial\Omega$$

  to obtain the **Dirichlet to Neumann** map

  $$\Lambda_\sigma \; : \; u \; \rightarrow \; \sigma \frac{\partial u}{\partial n} \quad \text{on} \quad \partial\Omega \tag{134}$$

  also called voltage to current density map.

- Apply a current density J on the boundary and measure the resulting voltage u. For a static situation the total current into Ω has to be zero, i.e.

  $$\oint_{\partial\Omega} J(s)\, ds = \oint_{\partial\Omega} \sigma \frac{\partial u}{\partial n}\, ds = 0$$

  to obtain the **Neumann to Dirichlet** map

  $$\mathcal{R}_\sigma \; : \; \sigma \frac{\partial u}{\partial n} \; \rightarrow \; u \quad \text{on} \quad \partial\Omega \tag{135}$$

  also called current density to voltage map.

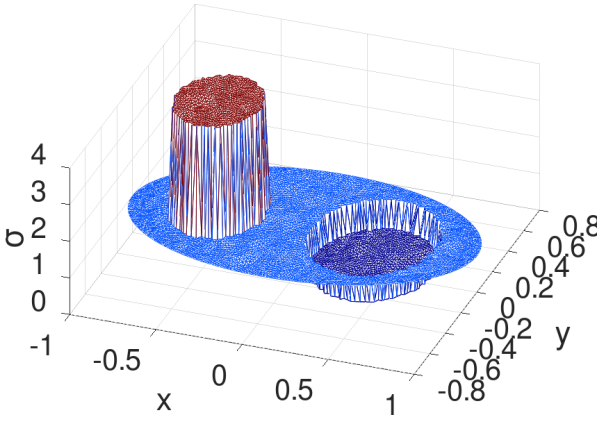


Figure 190: The conductivity with the conducting “heart” on the left and the insulating “lung” on the right

From either one of these maps it is possible to determine the conductivity σ in the domain. This method is called Electrical Impedance Tomography, or short EIT. For a good, readable introduction consider the book [MuelSilt12] or the article [MuelSilt20]. The Neumann to Dirichlet map $\mathcal{R}_\sigma$ is more reliable to measure, based on less susceptibility to noise. Using FEM examine the forward problem, i.e. apply a known current pattern and determine the resulting voltage u on the boundary. In real live this is performed by measurements. Examine the domain (a very theoretical chest cross section) in Figure 190 with the graph of the conductivity σ shown. On the left observe a simple heart with high conductivity, caused by the blood. On the right observe a section with very low conductivity, caused by the air filled lung. Then two current patterns are examined:

1. A current input at the lower edge of the cross section in Figure 191 and a matching current sink at an angle of approximately $120°$. Thus the current is expected to go through the heart, mainly.

2. A similar current input at the lower edge and a matching current sink at an angle of approximately $60°$. Thus the current is expected to go through the lung, mainly.

The boundary Γ of the domain Ω is given by

$$\begin{pmatrix} R_x \cos\alpha \\ R_y \sin\alpha \end{pmatrix} \quad \text{for } 0 \le \alpha \le 2\,\pi \text{ with } R_x = 1 \text{ and } R_y = 0.5\,,$$

with a conductivity of $\sigma = 1$. A simple calculation on the ellipse leads to an arc length of

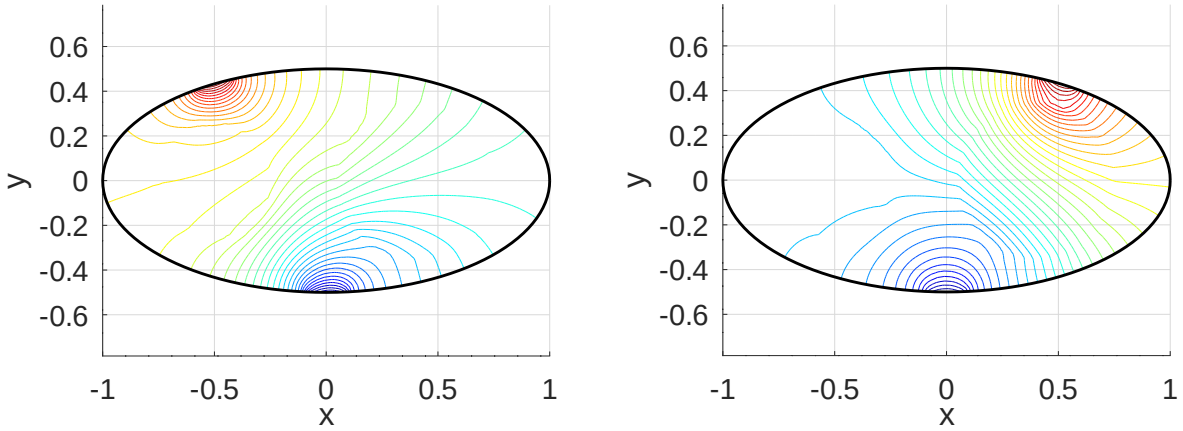
$$ds = \sqrt{R_x^2 \sin^2\alpha + R_y^2 \cos^2\alpha}\; d\alpha\,.$$

The "heart" is given by

$$(x+0.5)^2 + y^2 \le 0.25^2 \quad \text{with conductivity} \quad \sigma = 4$$

and the "lung" is given by

$$(x-0.4)^2 + y^2 \le 0.35^2 \quad \text{with conductivity} \quad \sigma = \frac{1}{4}\,.$$



(a) current sink at angle 120°

(b) current sink at angle 60°

Figure 191: Contours of the voltages

FEMoctave is used twice to determine the voltage u in the domain, leading to the level curves in Figure 191. Observe that the two setups are rather similar, but not exactly symmetrical.

Since the normal derivative $\frac{\partial u}{\partial n}$ on all of the boundary is specified, the BVP does not have a unique solution. An arbitrary constant can be added and consequently the standard FEMoctave code will fail. If the additional condition

$$u_{mean} = \frac{1}{\text{area}(\Omega)} \iint\limits_{\Omega} u\; dA = 0$$

is required, the problem has a unique solution again, and there is hope to obtain a good aproximation by FEM. To get around this problem use the open and free source code of FEMoctave and modify the solver in `BVP2Dsym.m`. Add an additional equation

$$\sum_{i=1}^{n} u_i = 0$$

by one additional line, containing `n=size(A,1); A(n+1,:)=1; b(n+1)=0;` . It is a good idea to rename the function, e.g. to `BVP2DsymMean.m`.

BVP2DsymMean.m

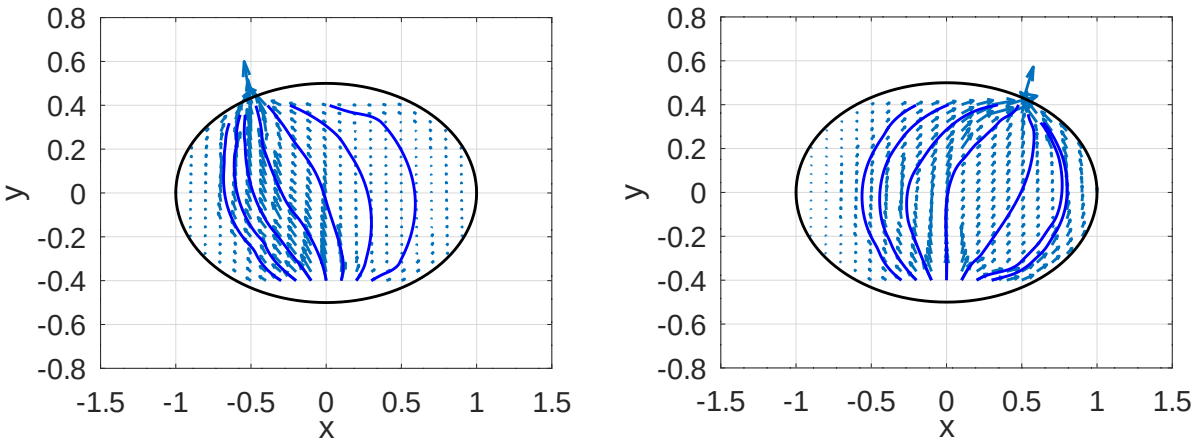
```
function u = BVP2DsymMean(Mesh,a,b0,f,gD,gN1,gN2)
  if nargin ~= 7    print_usage();  endif
  switch Mesh.type
    case 'linear'  %% first order elements
```

```
    [A,b] = FEMEquation (Mesh,a,b0,0,0,f,gD,gN1,gN2);     % compute with compiled code
  case 'quadratic' %% second order elements
    [A,b] = FEMEquationQuad(Mesh,a,b0,0,0,f,gD,gN1,gN2);  % compute with compiled code
  case 'cubic' %% third order elements
    [A,b] = FEMEquationCubic(Mesh,a,b0,0,0,f,gD,gN1,gN2); % compute with compiled code
endswitch
%% add the zero mean condition
n = size(A,1); A(n+1,:) = 1; b(n+1) = 0;
u = FEMSolve(Mesh,A,b,gD);  %% solve the linear system
endfunction
```

Using the current density $\vec{J} = -\sigma \nabla u$ the vector fields in Figure 192 can be determined. With `FEMgriddata()` determine ∇u and then multiply by the conductivity σ to obtain the current density $\vec{J}$. Using the same starting points along $y = -0.4$ a few streamlines are shown.

- In Figure 192(a) the current takes the path of least resistance and is attracted by the highly conducting "heart".
- In Figure 192(b) the current tries to avoid the "lung" section with the low conductivity.

If the conductivity would be constant in all of the domain Ω, then the two graphics in Figure 192 would be perfectly symmetric.



(a) current sink at angle 120°

(b) current sink at angle 60°

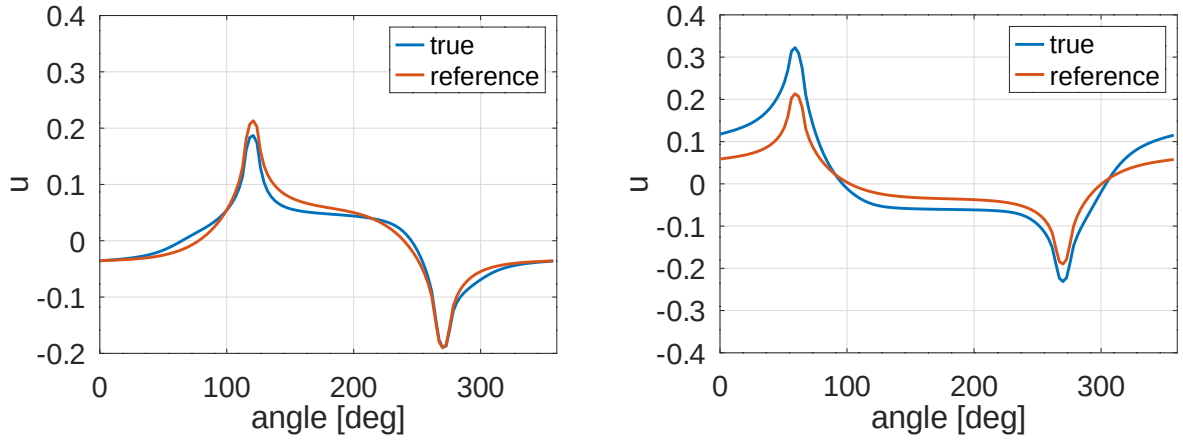
Figure 192: The vector field for the current density $\vec{J}$ and a few streamlines

As a reference the situation of constant $\sigma = 1$ is computed too and the resulting voltages on the boundary are shown in Figure 193. The deviations from the reference on the boundary Γ contain information about the conductivity inside of the domain Ω. The deviations from the reference are shown in Figure 194. Many of those "measurements" allow to determine the Neumann to Dirichlet map, leading to the conductivity σ by an EIT algorithm.

EITforward.m

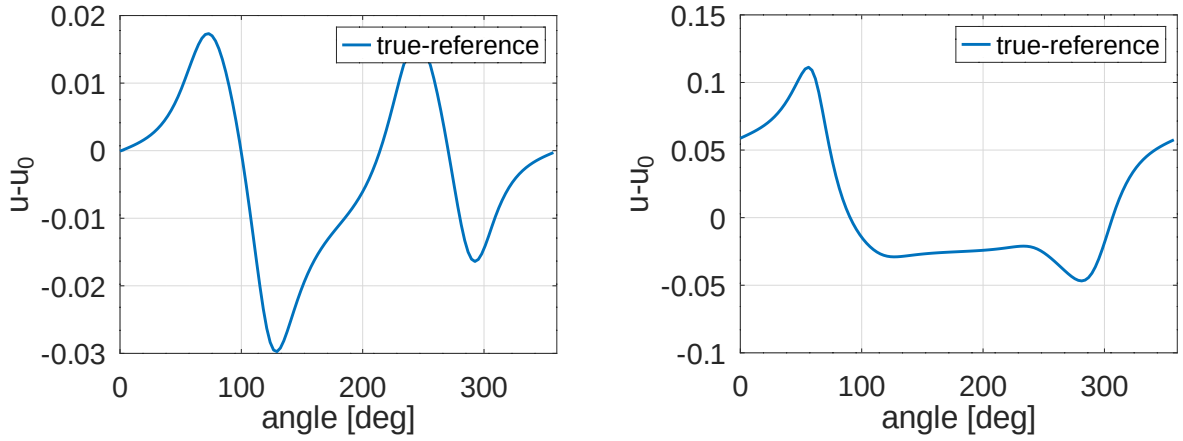
```
global Rx Ry dalpha my_angle
N = 2*64; %% number of angle segments
alpha = linspace(0,2*pi*(N-1)/N,N)';  Rx = 1; Ry = 0.5;
dalpha = 2*(alpha(2)-alpha(1));
x = Rx*cos(alpha); y = Ry*sin(alpha);
BC = -2*ones(size(x));
my_angle = 120  %% select the configuration, use 60 or 120

function res = sigma(xy,dummy)           %% the conductivity
  x = xy(:,1); y = xy(:,2);
  res = ones(size(x));
  res((x+0.5).^2+y.^2<=0.25^2) *=  4 ;   %% heart on the left
  res((x-0.4).^2+y.^2<=0.35^2) *=  1/4;  %% lung on the right
endfunction
```



(a) at angle 120° (b) at angle 60°

Figure 193: Voltage along the boundary



(a) at angle 120° (b) at angle 60°

Figure 194: Differences of the voltage and the reference voltage

```
FEMmesh = CreateMeshTriangle('EIT',[x,y,BC],0.003);
FEMmesh = MeshUpgrade(FEMmesh,'cubic');

figure(1); FEMtrimesh(FEMmesh,sigma(FEMmesh.nodes));  %% show the conductivity
           xlabel('x'); ylabel('y'); zlabel('\sigma'); view(20,50)

function res = flux_n(xy)         %% define the current density on the boundary
  global dalpha my_angle Rx Ry
  alpha = atan2(xy(:,2)/Ry,xy(:,1)/Rx);  %% assure correct angle
  res = zeros(size(alpha));
  res(abs(alpha+pi/2) < dalpha) = -1;
  switch my_angle
    case 60
      res(abs(alpha-pi/3)   < dalpha) = +1;
    case 120
      res(abs(alpha-pi*2/3) < dalpha) = +1;
  endswitch
  res = res./sqrt(Rx^2*sin(alpha).^2 + Ry^2*cos(alpha).^2);%% adjust for the arc length
endfunction

u_0 = BVP2DsymMean(FEMmesh,   1   ,0,0,0,'flux_n',0);  %% the reference result
u   = BVP2DsymMean(FEMmesh,'sigma',0,0,0,'flux_n',0);  %% the actual result

figure(2); FEMtrimesh(FEMmesh,u)                    %% show the solution
           xlabel('x'); ylabel('y');

figure(3); clf; FEMtricontour(FEMmesh,u,41)         %% show the contour levels
           hold on;
           plot([x;x(1)],[y;y(1)],'k');             %% add the boundary
           hold off; xlabel('x'); ylabel('y'); axis equal

u_boundary   = FEMgriddata(FEMmesh,u,  x,y);
u_0_boundary = FEMgriddata(FEMmesh,u_0,x,y);

figure(4); plot(alpha*180/pi,u_boundary,alpha*180/pi,u_0_boundary)
           xlabel('angle [deg]'); ylabel('u'); xlim([0,360])
           legend('true','reference')     %% show the voltages on the boundary
figure(5); plot(alpha*180/pi,u_boundary-u_0_boundary)
           xlabel('angle [deg]'); ylabel('u-u_0'); xlim([0,360])
           legend('true-reference')  %% show the difference
%% create the vector field for the current density
[xx,yy] = meshgrid(linspace(-Rx,Rx,21),linspace(-0.8*Ry,0.8*Ry,21));
[ui,uxi,uyi] = FEMgriddata(FEMmesh,u,xx,yy);
conductivity = reshape(sigma([xx(:),yy(:)]),size(xx));
uxi = conductivity.*uxi;  uyi = conductivity.*uyi;

figure(6); quiver(xx,yy,uxi,uyi,2)    %%% show the vector field
           xlabel('x'); ylabel('y')
           hold on;  plot([x;x(1)],[y;y(1)],'k');  hold off%% add the boundary
%% create and show the streamlines
streamline(xx,yy,uxi,uyi,[-0.3 -0.2,-0.1,0,0.1,0.2 0.3],-0.8*Ry*ones(1,7));
```

Since the condition

$$\oint_{\partial\Omega} J(s)\, ds = \oint_{\partial\Omega} \sigma \frac{\partial u}{\partial n}\, ds = 0$$

is critical it is a good idea to examine the numerical approximation of the flux through the boundary. For this use the nornal vector

$$\vec{n} = \frac{1}{\sqrt{R_x^2 \sin^2\alpha + R_y^2 \cos^2\alpha}} \begin{pmatrix} R_y \cos\alpha \\ R_x \sin\alpha \end{pmatrix}$$

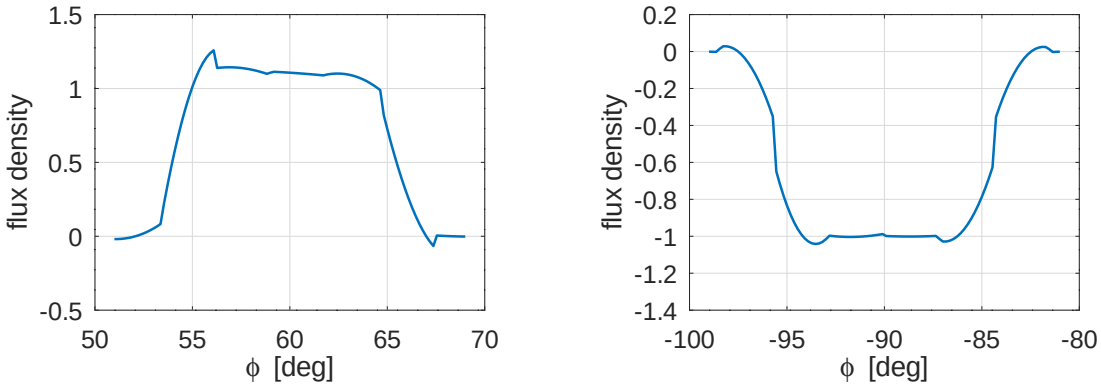
and then integrate over the segements where the flux is not zero

$$\int_{\text{section}} \langle \vec{n}, \nabla u \rangle \, ds \; .$$

To evaluate this numerically use `FEMgriddata()` to determine the values of the gradient $(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y})$ and then `trapz()` to perform a numerical integration. Observe that along the boundary the length segment is given by

$$ds = \sqrt{R_x^2 \sin^2 \alpha + R_y^2 \cos^2 \alpha} \; d\phi \; .$$

The code below leads to an inlet flux of ≈ 0.1975 and to outlet fluxes at either ≈ 0.1980 at $60°$ or ≈ 0.1964 at $120°$.



(a) outlet at angle $+60°$ (b) inlet at angle $-90°$

Figure 195: Flux density at inlet and outlet

AnalyzeBoundary.m

```
%% script to analyze the flux on the boundary
%% assumes that EITforward.m was run before
Angle = -90  % use 60, 120 or -90
Angle = deg2rad(Angle);
Section = pi/20; phi = Angle + linspace(-Section,+Section,100)';
x_b = 0.999*Rx*cos(phi); y_b = 0.999*Ry*sin(phi);

[u_boundary,ux_boundary,uy_boundary]    = FEMgriddata(FEMmesh,u,x_b,y_b);

ds = sqrt(Rx^2*sin(phi).^2 + Ry^2*cos(phi).^2);
n = [Ry*cos(phi)./ds, Rx*sin(phi)./ds];

flux = (ux_boundary.*n(:,1) + uy_boundary.*n(:,2));
figure(1); plot(rad2deg(phi),flux)
           xlabel('\phi [deg]'); ylabel('flux density')
TotalFlux = trapz(phi,flux.*ds)
```

9.26 A catenary

A catenary is a curve describing the shape of a flexible, hanging cable, e.g. [AgarHodiRega19]. A boundary value problem describing a catenary is

$$-u''(x) = -\frac{m\,g}{T}\sqrt{1 + (u'(x))^2} \quad \text{with} \quad u(-1) = u(1) = 0,$$

where m is the mass per unit length and T the horizontal tension. The exact solution is given by

$$u_{exact}(x) = \frac{T}{m\,g}\left(\cosh(\frac{m\,g}{T}\,x) - \cosh(\frac{m\,g}{T})\right) \; .$$

With `BVP1DNL()` this nonlinear boundary value problem can be solved, see the code `Catenary.m` below.

- A grid with 50 elements of order 2 is used.
- To use `BVP1DNL()` the function for the RHS and its derivatives are required, i.e.

$$f(x,u,u') = \frac{m\,g}{T}\sqrt{1+(u')^2} \quad , \quad \frac{\partial}{\partial u} f(x,u,u') = 0 \quad \text{and} \quad \frac{\partial}{\partial u'} f(x,u,u') = \frac{m\,g}{T}\,\frac{u'}{\sqrt{1+(u')^2}} \; .$$

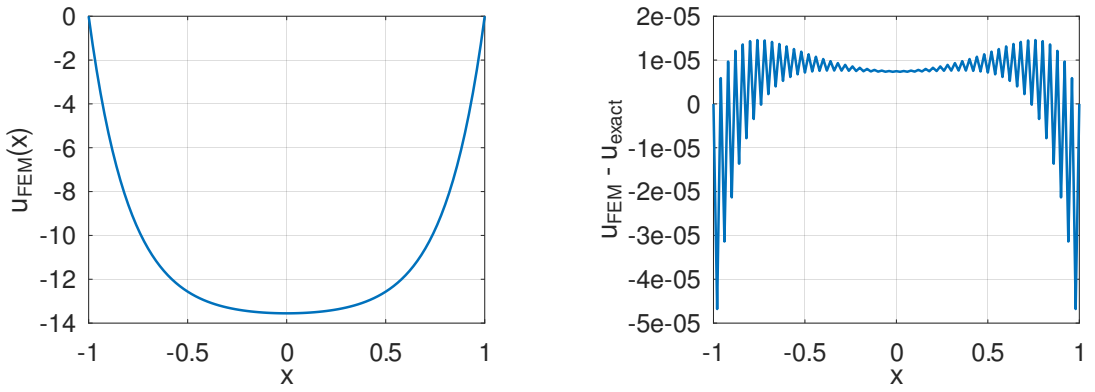
- As initial guess the naive attempt $u(x) = 0$ is used.
- With the option `Tol` at 10^{-12} a very low (absolute or relative) tolerance is asked for. Observe that this is the tolerance for the nonlinear solver in `BVP1DNL()` and not the difference to the exact solution. If a more accurate solution is required a finer grid has to be used. Figure 196 shows that the absolute discretization error is of the order 10^{-5}, while the absolute error for the solution of the discretized problem is of the order 10^{-11}.
- With the option `Display` as `iter` the progress of Newton's algorithm is displayed and the quadratic convergence can be observed.

Catenary.m

```
m = 1; g = 9.81; T = 2; Interval = linspace(-1,1,51)';
f = {@(x,u,du)m*g/T*sqrt(1+du.^2), @(x,u,du)0, @(x,u,du)m*g/T*du./sqrt(1+du.^2)};
[x,u,inform] = BVP1DNL(Interval,1,0,0,-1,f,0,0,0, 'Tol',1e-12, 'Display','iter');
inform
figure(1); plot(x,u); xlabel('x'); ylabel('u(x)');

u_exact = @(x)T/(m*g)*(-cosh(m*g/T) + cosh(m*g/T*x));
figure(2); plot(x,u-u_exact(x)); xlabel('x'); ylabel('u_{FEM} - u_{exact}');
-->
iteration 1, RMS(correction) = 9.222842e+00, RMS(phi) = 9.009386e+00
iteration 2, RMS(correction) = 4.310253e-01, RMS(phi) = 4.310159e-01
iteration 3, RMS(correction) = 1.614119e-05, RMS(phi) = 1.614119e-05
iteration 4, RMS(correction) = 8.842381e-12, RMS(phi) = 8.834653e-12

inform =  scalar structure containing the fields:
          info = 1
          iter = 4
          AbsError = 8.8424e-12
```



(a) the FEM solution (b) difference to the exact solution

Figure 196: A catenary, the FEM solution and the difference to the exact solution

To find the length L of the catenary use the integral

$$L = \int_{-1}^{+1} \sqrt{1+(u'(x))^2}\,dx = \int_{-1}^{+1} \sqrt{1+\sinh^2(\frac{m\,g}{T}\,x)}\,dx = \int_{-1}^{+1} \cosh(\frac{m\,g}{T}\,x)\,dx = \frac{2\,T}{m\,g}\,\sinh(\frac{m\,g}{T}) \; .$$

Using the command `FEM1DIntegrate()` this can be evaluated numerically, using Simpson's rule. The numerical result is rather close to the exact integral.

Catenary.m

```
du = FEM1DEvaluateDu(x,u);
Length = [FEM1DIntegrate(x,sqrt(1+du.^2)),2*T/(m*g)*sinh(m*g/T)]
-->
Length = [27.514  27.514]
```

9.27 Brachistochrone problem

In 1696 Johann Bernoulli asked the following question: *Given two points A and B in a vertical plane. What is the path to be used by a moving particle to get from point A to point B in the shortest time possible?* It is assumed that gravity is the only force available and friction is ignored. Using conservation of energy conclude that the velocity at a depth u is given by

$$\frac{m}{2}\,v^2 = m\,g\,u + \frac{m}{2}\,v_0^2 \qquad \Longrightarrow \qquad v = \sqrt{2\,g + v_0^2}\,.$$

For sake of a simplified notation work with $2\,g = 1$, i.e. $v = \sqrt{u + v_0^2}$. To find the travel time $T(u)$ from $x = a$ to $x = b$ use

$$\begin{aligned} dt &= \frac{ds}{v} = \frac{\sqrt{1 + (u')^2}}{\sqrt{u + v_0^2}}\,dx \\ T(u) &= \int_{x=a}^{b} dt = \int_{x=a}^{b} \frac{\sqrt{1 + (u'(x))^2}}{\sqrt{u(x) + v_0^2}}\,dx = \int_{x=a}^{b} g(u(x), u')x))\,dx \end{aligned}$$

The function $u(x)$ has to minimize the travel time $T(u)$, i.e. this is a calculus of variations problem. The Euler–Lagrange equation leads to the differential equation

$$\begin{aligned} g(u,u') &= (1 + (u')^2)^{1/2}\,(u + v_0^2)^{-1/2} \\ \frac{\partial}{\partial u}\,g(u,u') &= -\frac{1}{2}\,(1 + (u')^2)^{1/2}\,(u + v_0^2)^{-3/2} \\ \frac{\partial}{\partial u'}\,g(u,u') &= (1 + (u')^2)^{-1/2}\,u'\,(u + v_0^2)^{-1/2} \\ -\frac{d}{dx}\,\frac{\partial\,g}{\partial u'} &= -\frac{\partial\,g}{\partial u} \\ -\frac{d}{dx}\,\left((1 + (u')^2)^{-1/2}\,(u + v_0^2)^{-1/2}\,u'\right) &= +\frac{1}{2}\,(1 + (u')^2)^{1/2}\,(u + v_0^2)^{-3/2} \end{aligned}$$

Supplement this differential equation with the boundary conditions $u(a) = 0$ and $u(b) = h$. Then the problem can be examined with the help of `BVP1DNL()`. To use `BVP1DNL()` the function $f(x,u,u')$ and its partial derivatives are required.

$$\begin{aligned} a(x,u,u') &= (1 + (u')^2)^{-1/2}\,(u + v_0^2)^{-1/2} \\ f(x,u,u') &= +\frac{1}{2}\,(1 + (u')^2)^{1/2}\,(u + v_0^2)^{-3/2} \\ \frac{\partial}{\partial u}\,f(x,u,u') &= -\frac{3}{4}\,(1 + (u')^2)^{1/2}\,(u + v_0^2)^{-5/2} \\ \frac{\partial}{\partial u'}\,f(x,u,u') &= +\frac{1}{2}\,(1 + (u')^2)^{-1/2}\,u'\,(u + v_0^2)^{-3/2} \end{aligned}$$

The algorithm implemented by the code `Brachistochrone.m` below uses four different initial speeds v_0.

1. Select the values for the interval $[a,b]$ and $u(b)$. Choose[53] $0 < v_0 = 2$ and define the coefficient $a(x,u,u')$ and the function $f(x,u,u')$ and the partial derivatives $\frac{\partial\,f}{\partial u}$ and $\frac{\partial\,f}{\partial u'}$. Define the initial function $u_0(x) = \frac{u(b)}{L}\,x$ and solve for the solution $u_2(x)$.

2. Select the new $v_0 = 1.0$ and redefine the functions. With the initial function $u_2(x)$ determine the new solution $u_1(x)$.

[53] For an initial velocity $v_0 = 0$ the exact solution has an infinite slope at $x = a$. Within `BVP1DNL()` this leads to divisions by almost zero and no convergence. `BVP1DN()` will usually fail for singular problems.

3. Select the new $v_0 = 0.75$ and redefine the functions. With the initial function $u_1(x)$ determine the new solution $u_{75}(x)$.
4. Select the new $v_0 = 0.50$ and redefine the functions. With the initial function $u_{75}(x)$ determine the new solution $u_{50}(x)$.
5. Generate Figure 197 with the four solutions.

In Figure 197 observe that all solutions are far from the shortest connection, i.e. the straight line. The movements start out going down fast, to pick up speed, with the goal to minimize the travel time. For smaller initial speed v_0 the curve is diving down further.

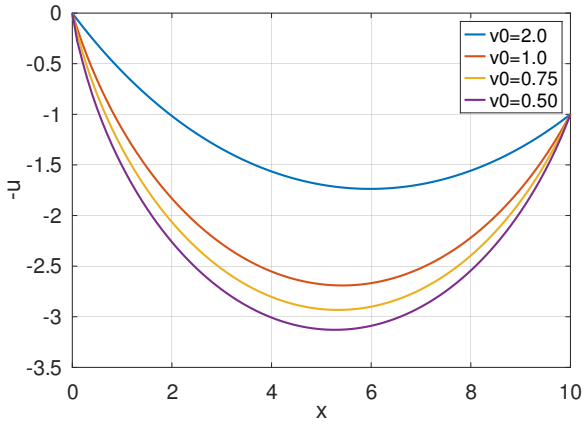


Figure 197: Four solutions of the brachistochrone problem

Brachistochrone.m

```
L = 10; u_left = 0; u_right = 1; Interval = linspace(0,L,51);

v0 = 2.0;
a = @(x,u,du)(1+du.^2).^(-1/2).*(u+v0^2).^(-1/2);
f = {@(x,u,du) 1/2*(1+du.^2).^(+1/2).*(u+v0^2).^(-3/2),
     @(x,u,du)-3/4*(1+du.^2).^(+1/2).*(u+v0^2).^(-5/2),
     @(x,u,du) 1/2*(1+du.^2).^(-1/2).*du.*(u+v0^2).^(-3/2)};

u0 = @(x)u_left+x/L*(u_right-u_left);
[x,u2,inform] = BVP1DNL(Interval,a,0,0,1,f,u_left,u_right,u0,'display','iter','maxiter',50);

v0 = 1.0;
a = @(x,u,du)(1+du.^2).^(-1/2).*(u+v0^2).^(-1/2);
f = {@(x,u,du)1/2*(1+du.^2).^(1/2).*(u+v0^2).^(-3/2),
     @(x,u,du)-3/4*(1+du.^2).^(1/2).*(u+v0^2).^(-5/2),
     @(x,u,du)1/2*(1+du.^2).^(-1/2).*du.*(u+v0^2).^(-3/2)};
[x,u1,inform] = BVP1DNL(Interval,a,0,0,1,f,u_left,u_right,u2,'display','iter','maxiter',50);

v0 = 0.75;
a = @(x,u,du)(1+du.^2).^(-1/2).*(u+v0^2).^(-1/2);
f = {@(x,u,du)1/2*(1+du.^2).^(1/2).*(u+v0^2).^(-3/2),
     @(x,u,du)-3/4*(1+du.^2).^(1/2).*(u+v0^2).^(-5/2),
     @(x,u,du)1/2*(1+du.^2).^(-1/2).*du.*(u+v0^2).^(-3/2)};
[x,u75,inform] = BVP1DNL(Interval,a,0,0,1,f,u_left,u_right,u1,'display','iter','maxiter',50);

v0 = 0.50;
a = @(x,u,du)(1+du.^2).^(-1/2).*(u+v0^2).^(-1/2);
f = {@(x,u,du)1/2*(1+du.^2).^(1/2).*(u+v0^2).^(-3/2),
     @(x,u,du)-3/4*(1+du.^2).^(1/2).*(u+v0^2).^(-5/2),
     @(x,u,du)1/2*(1+du.^2).^(-1/2).*du.*(u+v0^2).^(-3/2)};
[x,u50,inform] = BVP1DNL(Interval,a,0,0,1,f,u_left,u_right,u75,'display','iter','maxiter',50);
```

```
figure(1); plot(x,-u2,x,-u1,x,-u75,x,-u50); xlabel('x'); ylabel('-u')
           legend('v0=2.0','v0=1.0','v0=0.75','v0=0.50')
```

9.28 Stretching of a beam

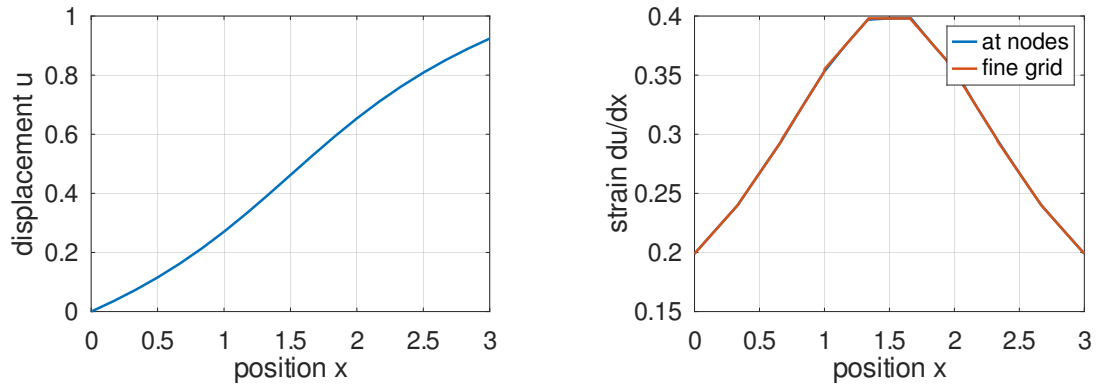
A beam with variable cross section $A(x)$ is stretched by a force F applied to the right endpoint at $x = L$. With Young's modulus of elasticity E the boundary value problem to be solved is

$$\frac{d}{dx}\left(E\,A(x)\,\frac{d\,u(x)}{dx}\right) = F \quad \text{with} \quad u(0) = 0 \quad \text{and} \quad EA(L)\,\frac{d\,u(L)}{dx} = F\,.$$

The solution is the displacement function $u(x)$. The resulting strain is given by the first derivative $\frac{d\,u(x)}{dx}$. For a beam with a thinner midsection examine

$$EA(x) = \frac{1}{2}\,\left(2 - \sin(\frac{x\,\pi}{L})\right)\,.$$

In Figure 198 find the displacement $u(x)$ and the resulting strain $\frac{d\,u(x)}{dx}$, determined by 10 elements of equal length. It is clearly visible that the strain $u'(x)$ is a piece-wise linear function. Thus just evaluating $u'(x)$ at more grid points will not improve the appearance of the solution. Rerunning the code below with more elements (e.g. `N = 50`) will improve the situation.



(a) the displacement $u(x)$ (b) the strain $u'(x)$

Figure 198: Displacement and strain for a beam with thinner midsection

BeamStretch.m

```
L = 3; F = 0.2; N = 10; EA = @(x) (2-sin(x/L*pi))/2;
[x,u] = BVP1D(linspace(0,L,N),EA,0,0,0,0,[F,0]);
figure(1); plot(x,u)
           xlabel('position x'); ylabel('displacement u')
[u2,strain] = pwquadinterp(x,u,x);                %% evaluation at nodes
x_fine = linspace(0,L,501);
[u_fine,strain_fine] = pwquadinterp(x,u,x_fine); %% interpolation to a fine grid
figure(2); plot(x,strain,x_fine,strain_fine)
           xlabel('position x'); ylabel('strain du/dx')
           legend('at nodes', 'fine grid')
```

9.29 How a Fata Morgana is appearing

In calm weather a layer of significantly warmer air may rest over colder, denser air. Thus the density of the air and the index of refraction decrease as one moves up. This leads to a higher speed of light at higher altitude u. As very simple assumption use the linear approximation for the index of refraction $n(u) = 1 - \alpha u$ for some positive constant α. Examine the height $u(x)$ of a ray of light as function of the position $a \le x \le b$. Use

$$\frac{ds}{dt} = v(u) = \frac{1}{n(u)} = \frac{1}{1 - \alpha\,u}$$

for the speed of light. With the arc length $ds = \sqrt{1+(u'(x))^2}\,dx$ determine the time of flight $T(u)$ for a path $u(x)$ of the ray from $x = a$ to $x = b$.

$$T(u) \;=\; \int_a^b \frac{1}{v(x)}\,\frac{ds}{dx}\,dx \;=\; \int_a^b (1-\alpha\,u(x))\,\sqrt{1+(u'(x))^2}\,dx \;=\; \int_a^b g(u(x),u'(x))\,dx$$

Based on Fermat's principle the shape $u(x)$ of the ray is such that the time of flight is minimal. The Euler–Lagrange equation of the calculus of variations leads to an ODE describing this situation. Use the partial derivatives

$$\frac{\partial}{\partial u}\,g(u,u') = -\alpha\,\sqrt{1+(u')^2} \quad\text{and}\quad \frac{\partial}{\partial u'}\,g(u,u') = (1-\alpha\,u)\,\frac{u'}{\sqrt{1+(u')^2}}$$

to conclude

$$\begin{aligned} -\frac{d}{dx}\,\frac{\partial}{\partial u'}\,g(u,u') &= -\frac{\partial}{\partial u}\,g(u,u') \\ -\frac{d}{dx}\left(\frac{1-\alpha\,u}{\sqrt{1+(u')^2}}\,u'\right) &= +\alpha\,\sqrt{1+(u')^2} \end{aligned}$$

To use the algorithm of `BVP1DNL()` work with $a(u,u') = \frac{1-\alpha\,u}{\sqrt{1+(u')^2}}$ and

$$f(u,u') = \alpha\,\sqrt{1+(u')^2} \quad,\quad \frac{\partial}{\partial u}\,f(u,u') = 0 \quad\text{and}\quad \frac{\partial}{\partial u'}\,f(u,u') = \alpha\,\frac{u'}{\sqrt{1+(u')^2}}\;.$$

Then find the ray connecting the points $(0,0$ and $(1,0)$ by the code `FataMorgana.m` below, leading to Figure 199.

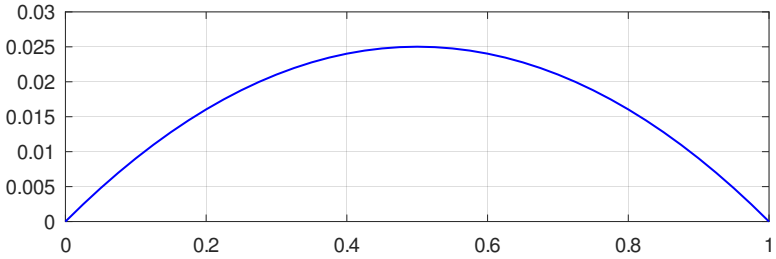


Figure 199: Fata Morgana: the ray of light for a point on the ground

FataMorgana.m

```
Interval = linspace(0,1,21)';
alpha = 0.2;
f = {@(x,u,v)alpha*sqrt(1+v.^2),
     @(x,u,v)0*u;
     @(x,u,v)alpha*v./sqrt(1+v.^2)};
a =  @(x,u,v)(1+alpha*u)./sqrt(1+v.^2);
[x,u] = BVP1DNL(Interval,a,0,0,1,f,0,0,0);
figure(1); plot(x,u,'b');
```

To find out where an object of height 0.05 at $x = 1$ is appearing use the same idea to find the rays form $(0,0)$ to $(1,0)$ and $(1,05)$. Determine the slopes at $x = 0$ and draw the tangent lines of the rays. The resulting Figure 200 shows the object seemingly floating in the air. This is the physics behind a Fata Morgana, see en.wikipedia.org/wiki/Fata_Morgana_(mirage).

FataMorgana.m

```
du = FEM1DEvaluateDu(x,u); du1 = du(1);
[x,u2] = BVP1DNL(Interval,a,0,0,1,f,0,0.05,0);
du = FEM1DEvaluateDu(x,u2); du2 = du(1);
figure(2); plot(x,u,'b',[0,1],[0,du1],'g',x,u2,'b',[0,1],[0,du2],'g')
           legend('true','visual','location','northwest')
```

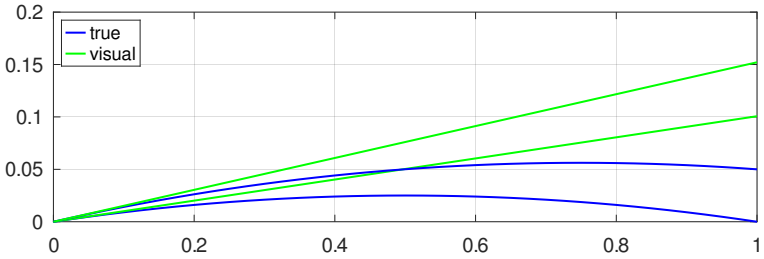


Figure 200: Fata Morgana: how the image of two points appear to the observer

9.30 Keller's nonlinear boundary value problems

In [Kell92, p. 317] the boundary value problem

$$-u''(x) = -e^{u(x)} \quad \text{with} \quad u(-1) = u(1) = 0$$

is examined. The exact solution is given by

$$u(x) = \ln\left(\frac{c^2}{1+\cos(c\,x)}\right) ,$$

where the value of the constant c is determined as solution of the equation $c^2 = 1 + \cos c$. Use Newton's method to find $c \approx 1.1765019$. There are two possible approaches to solve this nonlinear boundary value problem, partial successive substitution or Newton's method. The command `BVP1DNL()` is based on Newton's method.

9.30.1 Partial successive substitution

Start with an initial guess, e.g. $u_0(x) = 0$, leading to $-\exp(u_0) = -1$. Then use the iteration

$$-u''_{n+1}(x) = -e^{u_n(x)} .$$

The coding in FEMoctave is rather straightforward, shown in the code below. Find the solution by (partial) successive substitution in Figure 201(a) and the difference to the exact solution after 5 iterations in Figure 201(b).

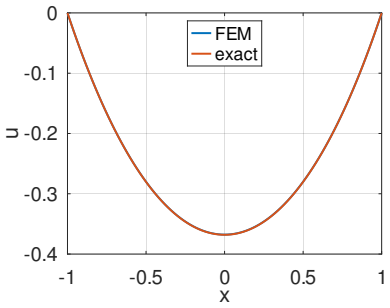
Keller.m

```
%% successive substitution
N = 51; interval = linspace(-1,1,N)';
c = 1.176501939901833; %% or use the solver
opt.TolFun = 1e-15; opt.TolX = 1e-15;  c = fsolve(@(c)1+cos(c)-c^2,1,opt);

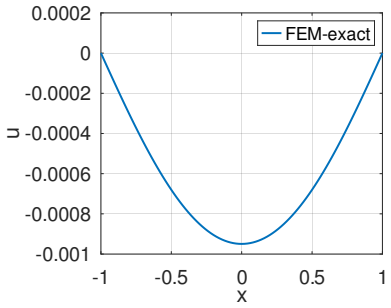
[x,u] = BVP1D(interval,1,0,0,1,-1,0,0);
u_exact = log(c^2./(1+cos(c*x)));
figure(1); plot(x,u); xlabel('x'); ylabel('u_1')
for jj = 1:4;
  [x,u] = BVP1D(interval,1,0,0,1,-exp(u),0,0);
  figure(2); plot(x,u,x,u_exact); xlabel('x'); ylabel('u')
             legend('FEM','exact', 'location','north')
  figure(3); plot(x,u-u_exact); xlabel('x'); ylabel('u')
             legend('FEM-exact')
  pause(0.2)
endfor
```

9.30.2 Newton's method

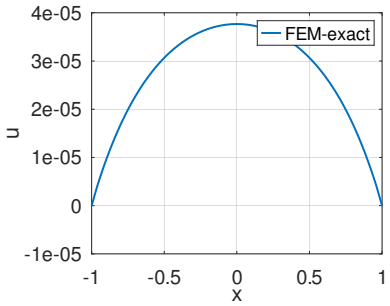
With an approximate initial guess $u_0(x)$ (start with $u_0(x) = 0$) search a sequence of new solutions of the form $u_{n+1}(x) = u_n(x) + \phi(x)$. Use the idea of a linear approximation to solve $0 = f(x_n + \phi) \approx f(x_n) + f'(x_n)\,\phi$ to find $\phi = -\frac{f(x_n)}{f'(x_n)}$, leading to the usual iteration formula $x_{n+1} = x_n + \phi = x_n - \frac{f(x_n)}{f'(x_n)}$. Examine the linear boundary



(a) the solution



(b) error of successive substitution



(c) error of Newton's method

Figure 201: The solution of Keller's boundary value problem and the difference of the approximations by successive substitution and Newton's method

value problem for the unknown function $\phi(x)$. The Taylor approximation $e^{u+\phi} \approx e^u + e^u\phi = e^u(1+\phi)$ leads to the BVP for $\phi(x)$.

$$\begin{aligned} -u_n''(x) - \phi''(x) &= -e^{u_n(x)+\phi(x)} \approx -e^{u_n(x)}\,(1+\phi(x)) \\ -\phi''(x) + e^{u_n(x)}\phi(x) &= u_n''(x) - e^{u_n(x)} \quad \text{with} \quad \phi(-1) = \phi(1) = 0 \end{aligned}$$

Use `[du,ddu] = FEM1DEvaluateDu(xn,u)` to find the values of the first and second derivatives at the nodes. The coefficient $e^{u_n(x)}$ of the term $\phi(x)$ depends on the previous solution $u_n(x)$ and its values have to be known at the Gauss integration points. Find the Gauss points with the help of the function `FEM1DGaussPoints()` and use `uGauss=pwquadinterp(xn,u,xGauss)` to evaluate $u_n(x)$ at the Gauss points. Find the solution by Newton's method in Figure 201(a) and the difference to the exact solution after 5 iterations in Figure 201(c).

Keller.m

```
%% Newton's method
N = 51; interval = linspace(-1,1,N)';
c = 1.176501939901833; %% or use the solver
opt.TolFun = 1e-15; opt.TolX = 1e-15;  c = fsolve(@(c)1+cos(c)-c^2,1,opt);

[x,u] = BVP1D(interval,1,0,0,1,-1,0,0);
u_exact = log(c^2./(1+cos(c*x)));
figure(1); plot(x,u); xlabel('x'); ylabel('u_1')
xGauss = FEM1DGaussPoints(x);
for jj = 1:4
  [du,ddu] = FEM1DEvaluateDu(x,u);
  RHS = + ddu - exp(u);
  uGauss = pwquadinterp(x,u,xGauss); %% evaluate u at Gauss points
  u_coeff = exp(uGauss);
  [x,phi] = BVP1D(interval,1,0,u_coeff,1,RHS,0,0);
  disp(sprintf('max(abs(phi)) = %g, max(abs(RHS)) = %g',max(abs(phi)), max(abs(RHS))))
  u = u + phi;
  figure(2); plot(x,u,x,u_exact); xlabel('x'); ylabel('u')
             legend('FEM','exact','location','north')
```

```
    figure(3); plot(x,u-u_exact); xlabel('x'); ylabel('u')
               legend('FEM-exact')
    pause(0.2)
  endfor
```

9.30.3 Using `BVP1DNL()`

Newton's method can be used with very few lines of code. To use `BVP1DNL()` provide the partial derivative

$$\frac{\partial}{\partial u} f(x,u) = \frac{\partial}{\partial u} \left(-\exp(u)\right) = -\exp(u)$$

and start with the naive initial guess $u_0(x) = 0$. The graphical result and the information provided in the return variable `inform` indicate that the algorithm converged with 4 iterations and the size of the last correction was of the order $1.4 \cdot 10^{-6}$. The RMS of the error is approximately $3 \cdot 10^{-8}$, only computable since we know the exact solution. The graph shows an actual maximal error of $\approx 2 \cdot 10^{-9}$.

Keller.m

```
%% use BVP1DNL
N = 51; interval = linspace(-1,1,N)';
c = 1.176501939901833; %% or use the solver
opt.TolFun = 1e-15; opt.TolX = 1e-15;  c = fsolve(@(c)1+cos(c)-c^2,1,opt);

f = {@(x,u)-exp(u) , @(x,u)-exp(u)};
[x,u,inform] = BVP1DNL(interval,1,0,0,1,f,0,0,0,'Tol',1e-5,'display','iter');
u_exact = log(c^2./(1+cos(c*x)));
figure(2); plot(x,u,x,u_exact); xlabel('x'); ylabel('u')
           legend('FEM','exact','location','north')
figure(3); plot(x,u-u_exact); xlabel('x'); ylabel('u')
           legend('FEM-exact')
inform
RMS_difference = norm(u-u_exact)/sqrt(length(u))
-->
iteration 1, RMS(correction) = 9.323090e-02, RMS(phi) = 9.468330e-02
iteration 2, RMS(correction) = 3.300574e-04, RMS(phi) = 3.300764e-04
iteration 3, RMS(correction) = 4.332861e-09, RMS(phi) = 4.332859e-09

inform = scalar structure containing the fields:
    info     = 1
    iter     = 3
    AbsError = 4.3329e-09

RMS_difference = 1.6647e-09
```

9.30.4 A similar problem with multiple solutions

At first sight[54] the nonlinear boundary value problem

$$-u''(x) = \frac{1}{2} e^{u(x)} \quad \text{with} \quad u(-1) = u(1) = 0$$

found in [Kell92, p. 150] is very similar to the above. But it turns out[55] that this problem has two solutions, one with $u(0) \approx 0.3$ and another solution with $u(0) \approx 3$. By selecting appropriate initial functions $u_0(x) = u_0\,(1 - x^2)$ the algorithm used in `BVP1DNL()` will determine either one of these two solutions.

[54] I later realized that this is a special case of Bratu's equation, examined in Section 9.7 starting on page 256. After rescaling Bratu's equation with $C = 2$ on the interval $[-1, +1]$ find Keller's example. The exact solutions are given by $u(x) = 2\,\ln(\frac{2\,\alpha}{\cosh(\alpha\,x)})$ for $\alpha \approx 0.58939$ and $\alpha \approx 2.126801$. This leads to maximal values of 0.3290 and 2.986.

[55] Examine the initial value problem $u''(x) = -\frac{1}{2}\,\exp(u(x))$ with $u'(0) = 0$ and $u(0) = u_0$. Examine the value $u(1)$ as function of u_0 and use a graph or `fsolve()` to find two zeros at $u_0 \approx 0.32895$ and $u_0 \approx 2.8955$.

Keller2.m

```
y01 = 0.3  %% more accurate y0 = 0.3290;
y02 = 3.0  %% more accurate y0 = 2.8955;
RHS = {@(x,y)0.5*exp(y), @(x,y)0.5*exp(y)};
interval = linspace(-1,1,21);
BCleft = 0; BCright = 0;
[x1,y1] = BVP1DNL(interval,1,0,0,1,RHS,BCleft,BCright,@(x)y01*(1-x.^2),
                  'MaxIter',30,'Display','iter');
[x2,y2] = BVP1DNL(interval,1,0,0,1,RHS,BCleft,BCright,@(x)y02*(1-x.^2),
                  'MaxIter',30,'Display','iter');
figure(1); plot(x1,y1,x2,y2); xlabel('x'); ylabel('y(x)')
```

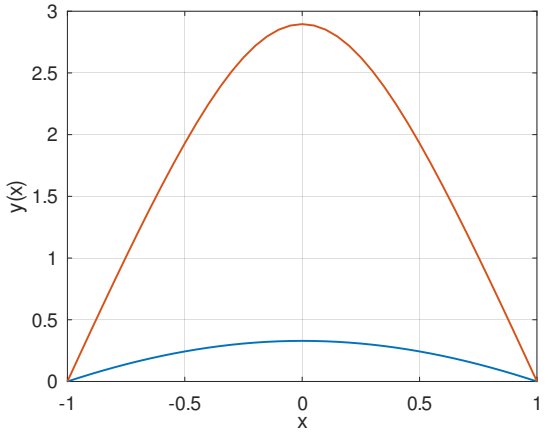


Figure 202: Two solutions of $-u''(x) = \frac{1}{2}\,\exp(u(x))$ with $u(-1) = u(+1) = 0$

9.31 A pendulum problem

With what angular velocity does a pendulum have to start at the lowest point with angle $u(0) = 0$ to reach the maximal angle at time $t = T$? What is the resulting maximal angle? A boundary value problem leading to the answers is

$$\ddot{u}(t) = -\sin(u(t)) \quad \text{with} \quad u(0) = 0 \quad \text{and} \quad \frac{d}{dt}\,u(T) = 0\,. \tag{136}$$

This BVP always has the trivial solution $u(t) = 0$. Since $\sin(u) \le u$ for $u \ge 0$ the acceleration for the pendulum is smaller than the one for the linearized pendulum equation $\ddot{u}(t) = -u(t)$. The solution of the linearized equation is $u(t) = c\,\sin(t)$. As consequence require $T > \frac{\pi}{2}$ to obtain a nontrivial solution for the BVP (136). Conservation of energy[56] requires

$$\text{initial kinetic energy} = \frac{1}{2}\,(\dot{u}(0))^2 = (1 - \cos(u(T)) = \text{final potential energy.}$$

The code `Pendulum.m` shown below solves the BVP (136) for $T = 2$ and the conservation of energy is verified.

[56] Conservation of energy can be used directly to determine the maximal angle as function of the initial velocity. The travel time T is then given by a singular integral.

$$\begin{aligned}
\frac{1}{2}\,v^2 &= (1 - \cos(u_{max})) - (1 - \cos(u)) \;=\; \cos(u) - \cos(u_{max}) \\
\frac{d\,u}{dt} = v(u) &= \sqrt{2\,(\cos(u) - \cos(u_{max}))} \\
T &= \int_0^{u_{max}} \frac{1}{v(u)}\,du \;=\; \int_0^{u_{max}} \frac{1}{\sqrt{2\,(\cos(u) - \cos(u_{max}))}}\,du
\end{aligned}$$

The code in `Pendulum.m` evaluates this integral.

Pendulum.m

```
N = 1001; interval = linspace(0,T,N)';
T = 2.0; BCleft = 0; BCright = [0,0];
f = {@(t,u)sin(u), @(t,u)cos(u)};
[t,u] = BVP1DNL(interval,1,0,0,1,f,BCleft,BCright,@(t)t, 'Display','off','Tol',1e-8);
figure(1); plot(t,u); xlabel('time t'); ylabel('angle u')
v = FEM1DEvaluateDu(t,u);
disp(sprintf('For T = %g: initial angular velocity v(0) = %g, maximal angle u(T) = %g',
                T,v(1),u(end)))
KineticEnergy = v(1)^2/2; Potential = 1-cos(u(end));
disp(sprintf('Kinetic energy at t=0: %g, potential energy at t=T: %g, difference: %g',
             KineticEnergy,Potential,KineticEnergy-Potential))
-->
For T = 2: initial angular velocity v(0) = 1.60481, maximal angle u(T) = 1.86263
Kinetic energy at t=0: 1.2877, potential energy at t=T: 1.2877, difference: 8.5831e-07
```

9.32 A BVP with multiple nonlinear contributions

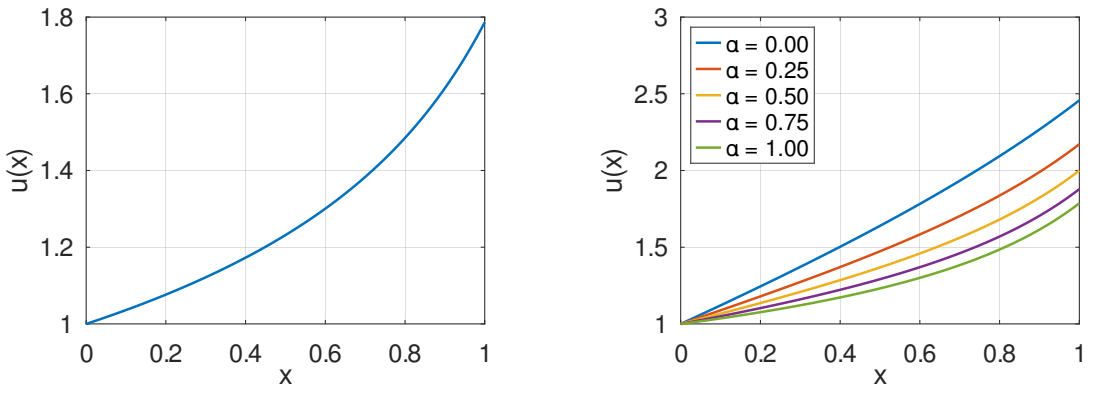
In [AtkiHan09] find Exercise 5.4.1 on page 241, a nonlinear boundary value problem with multiple nonlinear contributions.

$$\begin{aligned} -u''(x) + u'(x)\,u(x) + u^3(x) &= e^x \\ u(0) = 1\,, \quad u'(2) &= 2 \end{aligned}$$

A first attempt at solving this BVP with `BVP1DNL()` directly will fail. The algorithm does not converge, caused by the naive initial guess $u_0(x) = 0$, which should be close to the solution. It turns out that the contribution $u^3(x)$ leads to the blowup of the Newton algorithm. One possible way to solve this BVP is to parameterize the critical contribution. Introduce a parameter $0 \le \alpha \le 1$ and examine

$$\begin{aligned} -u''(x) &= f_\alpha(x, u(x), u'(x)) := -u'(x)\,u(x) - \alpha\,u^3(x) + e^x \\ f_\alpha(x, u, u') &= -u'\,u - \alpha\,u^3 + e^x \\ \frac{\partial}{\partial u} f_\alpha(x, u, u') &= -u' - \alpha\,3\,u^2 \\ \frac{\partial}{\partial u'} f_\alpha(x, u, u') &= -u \end{aligned}$$

Then increase the value of α step by step from 0 to 1. This approach will generate a reliable solution. Find the result from the code `AtkinsonHan.m` below in Figure 203.



(a) The final solution $u(x)$ (b) The solutions for different values of α

Figure 203: A BVP with multiple nonlinear contributions

AtkinsonHan.m

```
N = 501; interval = linspace(0,1,N)'; BCleft = 1; BCright = [2,0];
u = 0;
figure(2); clf; hold on; box on
for alpha = linspace(0,1,5)
  alpha
  f = {@(x,u,du)-u.*du - alpha*u.^3 + exp(x),@(x,u,du)-du -alpha*3*u.^2 ,@(x,u,du)-u};
  [x,u,inform] = BVP1DNL(interval,1,0,0,1,f,BCleft,BCright,u,
                'Display','iter');
  plot(x,u) ;xlabel('x'); ylabel('u(x)')
  pause(0.1)
endfor
legend('\alpha = 0.00','\alpha = 0.25','\alpha = 0.50','\alpha = 0.75','\alpha = 1.00',
       'location','northwest')
figure(1); plot(x,u)  ; xlabel('x'); ylabel('u(x)')
```

The output generated by the above code nicely illustrates the quadratic convergence of Newton's algorithm, i.e. the number of non-changing digits is approximately doubled at each step, after an initial search.

```
alpha = 0
iteration 1, RMS(correction) = 1.271951e+00, RMS(phi) = 9.217815e-01
iteration 2, RMS(correction) = 1.859463e-01, RMS(phi) = 1.978042e-01
iteration 3, RMS(correction) = 5.875778e-03, RMS(phi) = 5.886006e-03
iteration 4, RMS(correction) = 5.074170e-06, RMS(phi) = 5.074184e-06
alpha = 0.2500
iteration 1, RMS(correction) = 2.750162e-01, RMS(phi) = 5.594625e-01
iteration 2, RMS(correction) = 1.661185e-01, RMS(phi) = 2.075803e-01
iteration 3, RMS(correction) = 2.746584e-02, RMS(phi) = 2.827483e-02
iteration 4, RMS(correction) = 5.465197e-04, RMS(phi) = 5.468251e-04
iteration 5, RMS(correction) = 2.064430e-07, RMS(phi) = 2.064488e-07
alpha = 0.5000
iteration 1, RMS(correction) = 1.734538e-01, RMS(phi) = 4.004352e-01
iteration 2, RMS(correction) = 1.315342e-01, RMS(phi) = 1.798543e-01
iteration 3, RMS(correction) = 3.380237e-02, RMS(phi) = 3.581552e-02
iteration 4, RMS(correction) = 1.465643e-03, RMS(phi) = 1.469075e-03
iteration 5, RMS(correction) = 2.503296e-06, RMS(phi) = 2.503313e-06
alpha = 0.7500
iteration 1, RMS(correction) = 1.283952e-01, RMS(phi) = 3.183453e-01
iteration 2, RMS(correction) = 1.084098e-01, RMS(phi) = 1.567254e-01
iteration 3, RMS(correction) = 3.454831e-02, RMS(phi) = 3.743495e-02
iteration 4, RMS(correction) = 2.185036e-03, RMS(phi) = 2.195112e-03
iteration 5, RMS(correction) = 7.654415e-06, RMS(phi) = 7.654543e-06
alpha = 1
iteration 1, RMS(correction) = 1.031818e-01, RMS(phi) = 2.665817e-01
iteration 2, RMS(correction) = 9.226889e-02, RMS(phi) = 1.381142e-01
iteration 3, RMS(correction) = 3.318699e-02, RMS(phi) = 3.653564e-02
iteration 4, RMS(correction) = 2.598465e-03, RMS(phi) = 2.615903e-03
iteration 5, RMS(correction) = 1.360277e-05, RMS(phi) = 1.360325e-05
iteration 6, RMS(correction) = 3.689710e-10, RMS(phi) = 3.715615e-10
```

Another approach to find good initial values is the parametrization

$$f_\alpha(x,u,u') = \alpha\,(-u'\,u - u^3) + e^x\ .$$

With smaller steps for $0 \le \alpha \le 1$ this parametrization leads to the same final result.

9.33 Fisher's equation

Examine Fisher's equation (named after statistician and biologist Ronald Fisher), also known as the Kolmogorov–Petrovsky–Piskunov equation, or shorter Fisher-KPP equation.

$$\frac{\partial}{\partial t}\,u(x,t) - \frac{\partial^2}{\partial x^2}\,u(x,t) = f(u(x,t)) = u(x,t)\,(1-u(x,t)) \quad \text{with } u(-\infty,t) = 1 \text{ and } u(+\infty,t) = 0\ . \tag{137}$$

This initial boundary value problem can exhibit traveling wave solutions, switching between the two equilibrium states given by $f(u) = 0$, i.e. at $u = 0$ and $u = 1$. This type of equation occurs, e.g., in ecology, physiology, combustion, crystallization, plasma physics, and in general phase transition problems.

9.33.1 A travelling wave solution

When searching for a traveling wave solution with speed c use $u(x,t) = u(x - ct)$ and the above partial differential equation turns into an ordinary differential equation.

$$-c\,u'(x) - u''(x) = u(x)\,(1 - u(x)) \quad \text{with } u(-\infty) = 1 \text{ and } u(+\infty) = 0 \tag{138}$$

This ODE does not have a unique solution. Each translation $u(x+D)$ of a solution $u(x)$ is a solution too. Thus a naive usage of `BVP1DNL()` will not succeed. It also turns out that the condition $u(+\infty) = 0$ leads to numerical problems. A solution of (138) was generated with the ideas and algorithm spelled out below.

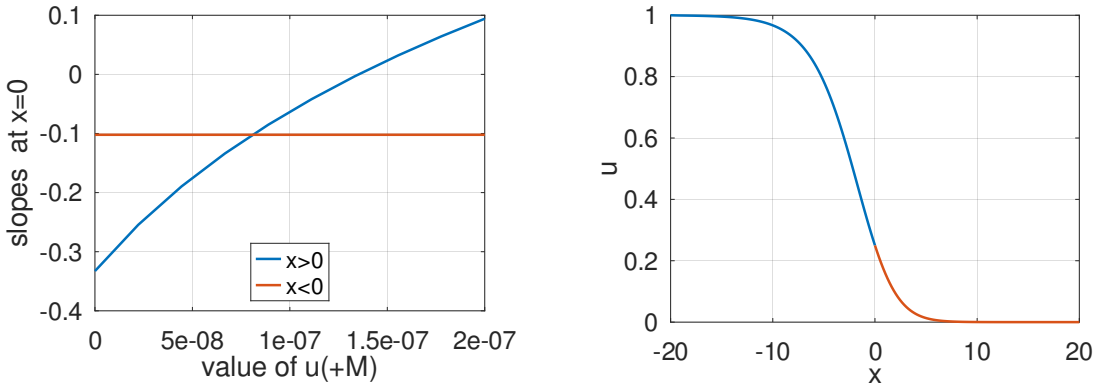
- Use $c = \frac{6}{\sqrt{5}} \approx 2.0412$[57] and select a value of $M = 20 \gg 1$. Then examine two boundary value problems:

$$\begin{aligned} -c\,u_n' - u_n'' &= u_n\,(1 - u_n) \quad \text{on } -M < x < 0 \text{ and } u_n(-M) = 1 \text{ and } u_n(0) = \tfrac{1}{4} \\ -c\,u_p' - u_p'' &= u_p\,(1 - u_p) \quad \text{on } 0 < x < +M \text{ and } u_p(0) = \tfrac{1}{4} \text{ and } u_p(+M) = \text{ValueAtM} \end{aligned}$$

  Consider the value of $u_P(+M)$ as parameter and determine its value such that $u_n'(0) = u_p'(0)$. Use the graphical hint in Figure 204(a) to select an initial interval for the *Octave* command `fzero()` to determine a numerical solution of $u_n'(0) = u_p'(0)$. The code below leads to the value $u_p(+20) \approx 8.0826 \cdot 10^{-8}$.

- With the obtained optimal value of $u(+M)$ solve the two boundary value problems on $-M < x < 0$ and $0 < x < +M$. Since $u_n(0) = u_p(0)$ and $u_n'(0) = u_p'(0)$ these two solutions can be patched together at $x = 0$ to obtain a solution of the ODE (138) on $-M < x < M$, leading to Figure 204(b) and the numerical confirmation of $u_n'(0) = u_p'(0)$.

```
first derivatives: u'(x-0)= -0.102206  and u'(x+0)= -0.102206, difference = 6.6294e-10
```



(a) The slopes at $x = 0$ as function of $u(+M)$

(b) The solution for $-M < x < M$

Figure 204: The solution of Fisher's equation

Fisher.m

```
function Fisher()
c = 5/sqrt(6); M = 20; N = 50; Join = 0.25;
f = {@(x,u)(u.*(1-u)), @(x,u)(1-2*u)};
interval = linspace(0,M,N)';  intervaln = linspace(-M,0,N)';
%% find the parameter
function slopes = FindSlopes(valueM,graphs)
```

[57] For $c = \frac{6}{\sqrt{5}}$ an exact solution is given by $u(x) = (1 + \exp(\frac{-x}{\sqrt{6}}))^{-2}$. This was also useful to find the good value for $u_p(+M)$.

```
 BCleft = Join; BCright = valueM;
 [xp,u0] = BVP1D(interval,1,-c,0,1,0,BCleft,BCright);
 [xp,up,inform] = BVP1DNL(interval,1,-c,0,1,f,BCleft,BCright,u0);
 BCleft = 1; BCright = Join;
 [xn,u0] = BVP1D(intervaln,1,-c,0,1,0,BCleft,BCright);
 [xn,un,inform] = BVP1DNL(intervaln,1,-c,0,1,f,BCleft,BCright,u0);
 dup = FEM1DEvaluateDu(xp,up); dun = FEM1DEvaluateDu(xn,un);
 slopes = [dun(end),dup(1)];
endfunction

%%ValueMList = linspace(0,1e-3,10);  %% for M = 10
ValueMList = linspace(0,2e-7,10);  %% for M = 20
for jj = 1:length(ValueMList)
 s = FindSlopes(ValueMList(jj)); sn(jj) = s(1); ,sp(jj) = s(2);
endfor

figure(1); plot(ValueMList,sp,ValueMList,sn);
xlabel('value of u(+M)'); ylabel('slopes  at x=0');
legend('x>0','x<0','location','south')

%% find the best value at x=+M
SameSlope = @(valueM)diff(FindSlopes(valueM));
ValueAtM = fzero(SameSlope,[5,9]*1e-8)     %% for M = 20

%% solve the two BVPs
BCleft = Join; BCright = ValueAtM;     %% for M = 20
u0 = @(x)(xp-M).^6/M^6*Join;
[xp,up,inform] = BVP1DNL(interval,1,-c,0,1,f,BCleft,BCright,u0,
                         'Display','off','tol',1e-8);
figure(11); plot(xp,u0(xp),xp,up); xlabel('x'); ylabel('u');
            legend('u_0','u','location','northeast')
intervaln = linspace(-M,0,N)'; BCleft = 1; BCright = Join;
u0 = @(x)1-(x+M).^4/M^4*(1-Join);
[xn,un,inform] = BVP1DNL(intervaln,1,-c,0,1,f,BCleft,BCright,u0,
                         'Display','off','tol',1e-8);
figure(12); plot(xn,u0(xn),xn,un); xlabel('x'); ylabel('u');
            legend('u_0','u','location','southwest')
figure(2); plot(xn,un,xp,up); xlabel('x'); ylabel('u');

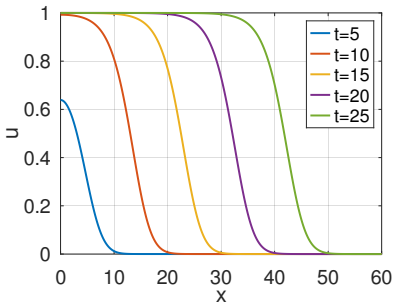
%% evaluate the  derivatives at x=0
[dup, ddup] = FEM1DEvaluateDu(xp,up);
[dun, ddun] = FEM1DEvaluateDu(xn,un);
disp(sprintf("first derivatives: u'(x-0)= %g  and u'(x+0)= %g, difference = %g",
              dun(end),dup(1),dun(end)-dup(1)))
endfunction
```

9.33.2 A dynamic solution

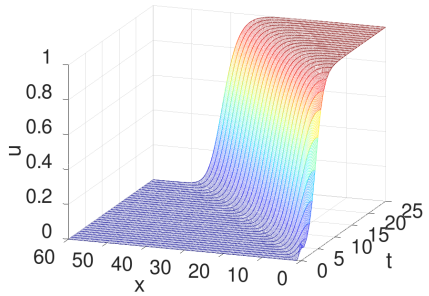
Fisher's equation (137) is a dynamic equation, for which the above approach was searching for traveling wave solutions. Use the command `IBVP1DNL()` to examine the dynamic behavior directly. On an interval $0 \le x \le 60$ use the boundary conditions $\frac{\partial}{\partial x} u(0,t) = 0$ and $u(60,t) = 0$. The initial condition $u(x,0)$ is a very small, positive contribution, localized at $x \approx 0$, e.g. $u(x,0) = 0.1 \exp(-3\,x^2)$. The code `FisherDynamic.m` leads to Figure 205. After a rather short time a traveling front forms and moves with a constant speed $c \approx 2$.

FisherDynamic.m

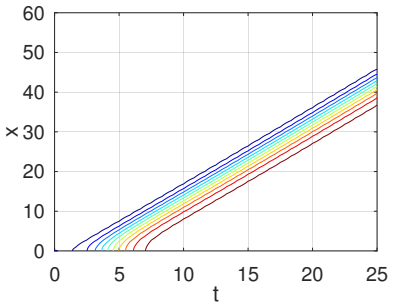
```
L = 60; N = 401; Interval = linspace(0,L,N)';
w = 1; a = 1; b = 0; c = 0; d = 1; alpha = 1.0;
f = {@(x,t,u)alpha*u.*(1-u),@(x,t,u)alpha*(1-2*u)};
t0 = 0; tend = 25; steps = [30,30];
BCleft = [0,0]; BCright = 0;
u0 = @(x)0.1*exp(-3*x.^2);
[x,u_all,t] = IBVP1DNL(Interval,w,a,b,c,d,f,BCleft,BCright,u0,t0,tend,steps,'tol', 1e-8);
```



(a) curves at different times



(b) the mesh



(c) contour curves

Figure 205: A dynamic solution of Fisher's equation

```
%% to use the Crank--Nicolson predictor corrector algorithm uncomment below
%%[x,u_all,t] = IBVP1DNL(Interval,w,a,b,c,d,f,BCleft,BCright,u0,t0,tend,steps,'solver','CNPC');
ind15 = find(abs(t-15)<1e-10); ind20 = find(abs(t-20)<1e-10); ind25 = find(abs(t-25)<1e-10);
ind5  = find(abs(t-5)<1e-10);  ind10 = find(abs(t-10)<1e-10);
figure(1); plot(x,u_all(:,ind5),x,u_all(:,ind10),x,u_all(:,ind15),...
               x,u_all(:,ind20),x,u_all(:,ind25)); xlabel('x'); ylabel('u');
           legend('t=5','t=10','t=15','t=20','t=25','location','northeast')
figure(2); mesh(t,x,u_all); xlabel('t'); xlim([t0,tend])
           ylabel('x'); zlabel('u'); view([-70,20])
figure(3); contour(t,x,u_all); xlabel('t'); ylabel('x');
```

9.33.3 A dynamic solution of the radially symmetric setup

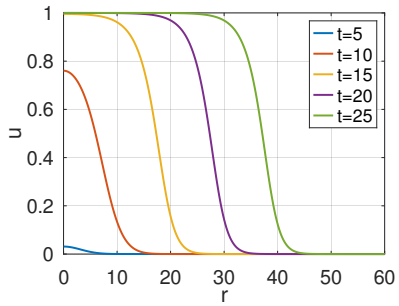
Fisher's equation (137) is for Cartesian coordinates and the planar front is moving in one direction. For the 3D case with the solution $u(r,t)$ depending on the radius $r = \sqrt{x^2+y^2+z^2}$ and time t examine

$$r^2 \frac{\partial}{\partial t} u(r,t) - \frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r} u(r,t)\right) = r^2 f(u(r,t)) = r^2 \frac{1}{\pi} \sin(\pi\, u(r,t)) \,. \tag{139}$$

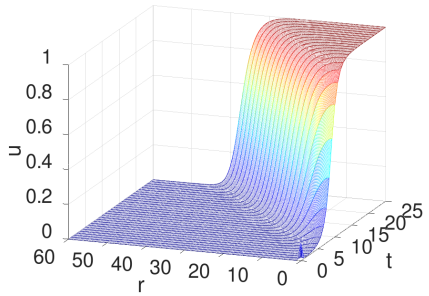
The nonlinear function $f(u) = u\,(1-u)$ is replaced by $f(u) = \frac{1}{\pi}\sin(\pi\, u)$, a function with the same key features $f(0) = f(1) = 0$, $f'(0) = 1$ and $f(u) > 0$ for $0 < u < 1$. With $f(u) = u\,(1-u)$ the problem was solved using a finite difference approach in [Stah08], This problem can be solved by `IBVP1DNL()` with very similar code and leads to similar results, visible in Figure 206.

FisherDynamicRadial.m

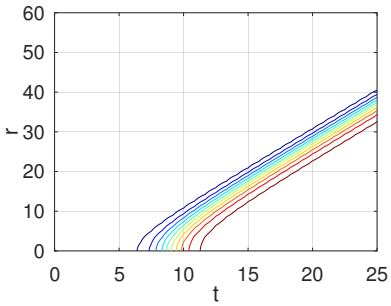
```
L = 60; N = 401; Interval = linspace(0,L,N)';
w = @(r)r.^2; a = @(r)r.^2; b = 0; c = 0; d = @(r)r.^2; alpha = 1.0/pi;
%%f = {@(r,t,u)alpha*u.*(1-u),@(r,t,u)alpha*(1-2*u)};
f = {@(r,t,u)alpha*sin(pi*u),@(r,t,u)alpha*pi*cos(pi*u)};
t0 = 0; tend = 25; steps = [30,60];
BCleft = [0,0]; BCright = 0;
u0 = @(r)0.1*exp(-3*r.^2);
```



(a) curves at different times



(b) the mesh



(c) contour curves

Figure 206: A dynamic solution of Fisher's equation for the radially symmetric setup

```
[r,u_all,t] = IBVP1DNL(Interval,w,a,b,c,d,f,BCleft,BCright,u0,t0,tend,steps,'tol', 1e-8);
ind15 = find(abs(t-15)<1e-10); ind20 = find(abs(t-20)<1e-10); ind25 = find(abs(t-25)<1e-10);
ind5 = find(abs(t-5)<1e-10); ind10 = find(abs(t-10)<1e-10);
figure(1); plot(r,u_all(:,ind5),r,u_all(:,ind10),r,u_all(:,ind15),...
                r,u_all(:,ind20),r,u_all(:,ind25)); xlabel('r'); ylabel('u');
           legend('t=5','t=10','t=15','t=20','t=25','location','northeast')
figure(2); mesh(t,r,u_all); xlabel('t'); xlim([t0,tend])
           ylabel('r'); zlabel('u'); view([-70,20])
figure(3); contour(t,r,u_all); xlabel('t'); ylabel('r');
```

9.33.4 A dynamic solution in the plane

In a plane $\mathbb{R}^2$ Fisher's equation can be examined by an initial boundary value problem on a square $0 \le x, y \le L$.

$$\begin{aligned} \frac{\partial u}{\partial t} - (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) &= u\,(1-u) && \text{for } 0 < x, y < L \text{ and } t > 0 \\ \frac{\partial u}{\partial n} &= 0 && \text{along } x = 0 \text{ and } y = 0 \\ u &= 0 && \text{along } x = L \text{ and } y = L \\ u(x, y, 0) &= u_0(x, y) && \text{for } t = 0 \end{aligned} \tag{140}$$

The problem can be solved by `IBVP2DNL()`. See the code `Fisher2D.m` below and the resulting graphs in Figure 207.

When the diffusion coefficients depend on the direction some of the above symmetries will disappear. Assuming only half of the diffusion in y–direction can be modeled by

$$(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \quad \longrightarrow \quad (\frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\frac{\partial^2 u}{\partial y^2})$$

and in the code by `a=[1 1/2 0]`, leading the the results in Figure 208. The slower speed of propagation in y–direction is clearly visible.

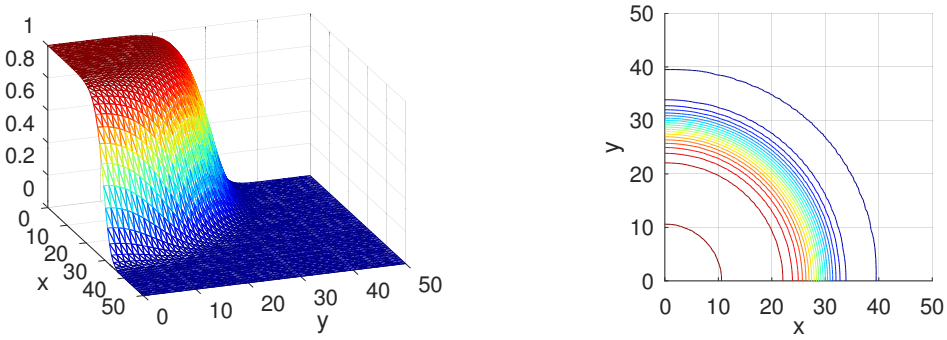


Figure 207: A dynamic solution of Fisher's equation on the plane

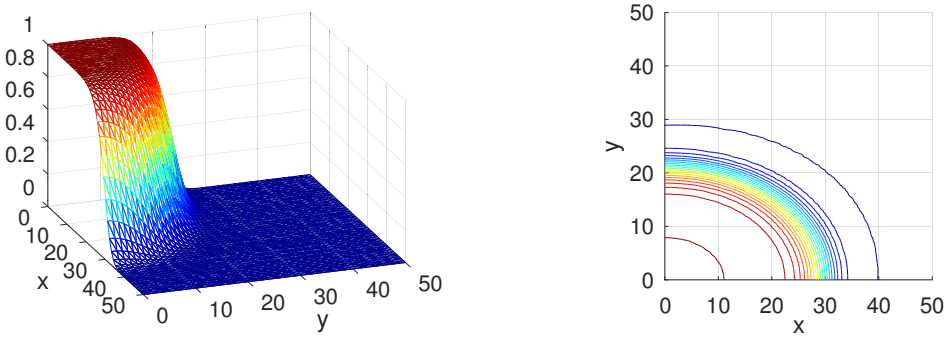


Figure 208: A dynamic solution of Fisher's equation with anisotropic diffusion coefficients

Fisher2D.m

```
N = 25;  L = 50;
FEMmesh = CreateMeshRect(linspace(0,L,N),linspace(0,L,N),-2,-1,-2,-1);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

m = 1; a = 1; b0 = 0; bx = 0; by = 0;
%% a = [1 1/2 0];  %% for nonistropic diffusion
gD = 0; gN1 = 0; gN2 = 0;
t0 = 0; tend = 20; steps = [25,10];
f = @(xy,t,u)u.*(1-u);
u0 = 0.02*exp(-(FEMmesh.nodes(:,1).^2+FEMmesh.nodes(:,2).^2));

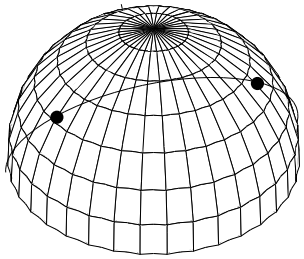
tic()
[u_dyn,t] = IBVP2DNL(FEMmesh,m,a,b0,bx,by,f,gD,gN1,gN2,u0,t0,tend,steps,'solver','CNPC');
toc()

u_end = u_dyn(:,end);
figure(1); FEMtrimesh(FEMmesh,u_end);    xlabel('x'); ylabel('y'); view([70,30])
figure(2); FEMtricontour(FEMmesh,u_end); xlabel('x'); ylabel('y');
           xlim([0,L]); ylim([0,L]); axis equal

if 1 %% animation
  figure(99)
  for jj = 1: size(u_dyn,2)
    FEMtrimesh(FEMmesh,u_dyn(:,jj));  xlabel('x'); ylabel('y'); view([50,30]); zlim([0,1]);
    drawnow(); pause(0.3)
  endfor
endif
```

9.34 From Salt Lake City to Zürich, the shortest connection on a sphere

On the right find a graphics of the northern hemisphere (radius $R = 6300$ km) with the two cities Zürich and Salt Lake City shown. The geographic data of the towns is given by



	latitude $\frac{\pi}{2} - \theta$	longitude ϕ
Salt Lake City	$41°$	$-112°$
Zürich	$47°$	$+8°$

Airlines prefer to use the shortest connection possible and will fly north first, and then turn back south. Use calculus of variations to find the corresponding boundary value problem and verify that the shortest connection follows a great circle.

To examine this problem use spherical coordinates.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} \sin\theta\ \cos\phi \\ \sin\theta\ \sin\phi \\ \cos\theta \end{pmatrix}$$

The longitude equals the angle ϕ and the latitude is given by $\frac{\pi}{2} - \theta$. Represent the connection from Salt Lake City to Zürich by writing the latitude as a function of the longitude, i.e. $\theta = u(\phi)$. To find the total length L express the length element dl in terms of spherical coordinates. Use basic calculus to find

$$\begin{aligned} dl &= R\sqrt{\sin^2\theta\, d\phi^2 + d\theta^2} = R\sqrt{\sin^2(u(\phi)) + (u'(\phi))^2}\, d\phi \\ L(u) &= R\int_{\phi_0}^{\phi_1} \sqrt{\sin^2(u(\phi)) + (u'(\phi))^2}\, d\phi = R\int_{\phi_0}^{\phi_1} g(\phi, u(\phi), u'(\phi))\, d\phi\,. \end{aligned}$$

To be determined is the function $\theta = u(\phi)$ such that the above functional $L(u)$ is minimized, respecting the boundary conditions $u(\phi_{SLC}) = \theta_{SLC}$ and $u(\phi_{ZH}) = \theta_{ZH}$. The Euler–Lagrange equation is given by

$$\begin{aligned} -\frac{d}{d\phi}\left(\frac{\partial}{\partial u'}\, g\right) &= -\frac{\partial}{\partial u}\, g \\ -\frac{d}{d\phi}\left(\frac{u'}{\sqrt{\sin^2(u) + (u')^2}}\right) &= -\frac{\cos(u)\,\sin(u)}{\sqrt{\sin^2(u) + (u')^2}}\,. \end{aligned} \tag{141}$$

This leads to a nonlinear boundary value problem for the function $u(\phi)$.

9.34.1 A solution based on successive substitution

With `FEMoctave` use the method of successive substitution and hope for convergence. Start with a straight line connection, i.e. the solution of $\frac{d^2}{d\phi^2}\, u(\phi) = 0$. Then hope for the iteration $u_{n-1} \to u_n$ to converge, where u_n solves

$$\frac{d}{d\phi}\left(\frac{u_n'}{\sqrt{\sin(u_{n-1})^2 + (u_{n-1}')^2}}\right) = \frac{\cos(u_{n-1})\,\sin(u_{n-1})}{\sqrt{\sin(u_{n-1})^2 + (u_{n-1}')^2}}\,.$$

To use the command `BVP1D()` for this iteration the following steps are performed.

- Determine the Gauss integration points by `phiG = FEM1DGaussPoints(phi).`
- Evaluate $u(\phi) = \theta(\phi)$ and $u'(\phi) = \frac{d}{d\phi}\,\theta(\phi)$ at the Gauss points with the help of `[thetaG,DthetaG] = pwquadinterp(phi,theta,phiG)`
- Evaluate the coefficients $a(\phi)$ and $d(\theta)$ at the Gauss points by
  - `a = (sin(thetaG).^2 + DthetaG.^2).^(-1/2)`

```
- d = sin(thetaG).*cos(thetaG)./sqrt(sin(thetaG).^2 + DthetaG.^2)
```

- Call `[phi,theta] = BVP1D(interval,a,0,0,d,-1,BCleft,BCright)`

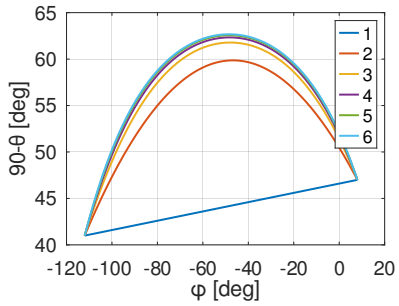
To verify that the shortest connection is on a great circle use the vectors $\vec{p}_{SLC}$ and $\vec{p}_{ZH}$ connecting the center of the earth to the two cities. Then determine the vector $\vec{n}$ orthogonal to the plane of the great circle connecting Salt Lake City and Zürich.

$$\vec{n} = \frac{1}{\|\vec{p}_{SLC} \times \vec{p}_{ZH}\|} \vec{p}_{SLC} \times \vec{p}_{ZH}$$

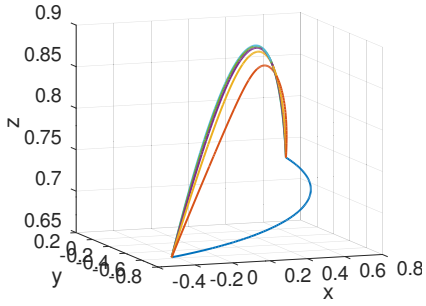
The scalar product $\langle \vec{p}, \vec{n} \rangle$ will determine the distance of the point $\vec{p}$ from the plane containing the great circle. In Figure 209 find the results of successive substitution performed by the code `SaltLakeCity2Zuerich.m` below. The (slow) animation also shows the length L for each iteration.

```
Iteration 1: L =  9510.058645 km
Iteration 2: L =  8532.915369 km
Iteration 3: L =  8491.805282 km
Iteration 4: L =  8485.252338 km
Iteration 5: L =  8483.738234 km
```

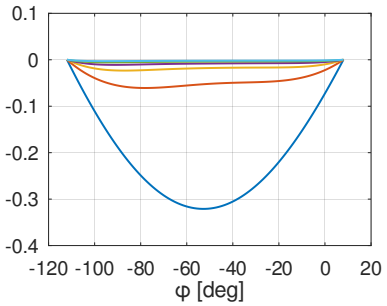
- Figure 209(a) shows the latitude as function of the longitude. The result of the first computations stays clearly to far south. The sequence of solutions is monotonically increasing and converging.
- Figure 209(b) shows the resulting curves in space. The length unit is the radius R of the earth.
- Figure 209(c) shows the distance from the plane of the great circle connecting Salt Lake City and Zürich. This distance converges to zero, i.e. the shortest connection is along a great circle.



(a) graph of the angles



(b) the curves in space



(c) distance from the plane

Figure 209: The connections between Salt Lake City and Zürich. The plots of latitude as function of longitude, the curves in space on the earth as sphere with radius 1 and the distances from the plane with the great circle.

SaltLakeCity2Zuerich.m

```
N = 51; R = 6300;
Angles_ZH = [8,90-47]/180*pi;   Angles_SLC = [-112,90-41]/180*pi;
Vec_ZH = R*[sin(Angles_ZH(2))*cos(Angles_ZH(1));
            sin(Angles_ZH(2))*sin(Angles_ZH(1));cos(Angles_ZH(2))];
Vec_SLC = R*[sin(Angles_SLC(2))*cos(Angles_SLC(1));
             sin(Angles_SLC(2))*sin(Angles_SLC(1));cos(Angles_SLC(2))];
n = cross(Vec_SLC,Vec_ZH); n = n/norm(n);
BCleft = Angles_SLC(2); BCright = Angles_ZH(2);
interval = linspace(Angles_SLC(1),Angles_ZH(1),N);
[phi,theta] = BVP1D(interval,1,0,0,1,0,BCleft,BCright);
figure(1); plot(phi/pi*180,90-theta/pi*180);
           xlabel('\phi [deg]'); ylabel('90-\theta [deg]'); hold on
x = sin(theta).*cos(phi); y = sin(theta).*sin(phi); z = cos(theta);
Dtheta = FEM1DEvaluateDu(phi,theta);
figure(2); plot3(x,y,z);
           xlabel('x'); ylabel('y'); zlabel('z'); view([-20,10]); hold on
disp(sprintf("Iteration 1: L = %f km",R*trapz(phi,sqrt(sin(theta).^2+Dtheta.^2))))
figure(3); plot(phi*180/pi,[x,y,z]*n);
           xlabel('\phi [deg]'); hold on;
phiG = FEM1DGaussPoints(phi);
for jj = 2:5;
  [thetaG,DthetaG] = pwquadinterp(phi,theta,phiG);
  a = (sin(thetaG).^2 + DthetaG.^2).^(-1/2);
  d = -sin(thetaG).*cos(thetaG)./sqrt(sin(thetaG).^2 + DthetaG.^2);
  [phi,theta] = BVP1D(interval,a,0,0,d,1,BCleft,BCright);
  Dtheta = FEM1DEvaluateDu(phi,theta);
  disp(sprintf("Iteration %i: L = %f km",
              jj,R*trapz(phi,sqrt(sin(theta).^2+Dtheta.^2))))
  figure(1); plot(phi/pi*180,90-theta/pi*180);
  x = sin(theta).*cos(phi); y = sin(theta).*sin(phi); z = cos(theta);
  figure(2); plot3(x,y,z);
  figure(3); plot(phi*180/pi,[x,y,z]*n);
  pause(0.2)
endfor %% jj
figure(1); legend('1','2','3','4','5'); hold off;
figure(2); hold off; figure(3); hold off
```

9.34.2 A solution using `BVP1DNL()`

When solving the BVP (141)

$$-\frac{d}{d\phi}\left(\frac{u'}{\sqrt{\sin^2(u)+(u')^2}}\right) = -\frac{\cos(u)\,\sin(u)}{\sqrt{\sin^2(u)+(u')^2}}\,.$$

with the help of `BVP1DNL()` the partial derivatives of the right hand side with respect to u and u' are required.

$$\begin{aligned} f(\phi,u,u') &= -\frac{\cos(u)\,\sin(u)}{\sqrt{\sin^2(u)+(u')^2}} \\ f_u = \frac{\partial}{\partial u} f(\phi,u,u') &= -\frac{1-2\,\sin^2(u)}{\sqrt{\sin^2(u)+(u')^2}} + \frac{\cos^2(u)\,\sin^2(u)}{\sqrt{\sin^2(u)+(u')^2}^3} \\ f_{u'} = \frac{\partial}{\partial u'} f(\phi,u,u') &= \frac{\cos(u)\,\sin(u)\,u'}{\sqrt{\sin^2(u)+(u')^2}^3} \end{aligned}$$

Then the algorithm in the code below uses a combination of Newton's method and partial substitution. Thus the convergence will not be drastically faster than the pure partial substitution implemented in the above code, but there is some error control and the algorithm has not to be coded explicitly.

SaltLakeCity2Zuerich.m

```
N = 51; R = 6300;
Angles_ZH = [8,90-47]/180*pi;   Angles_SLC = [-112,90-41]/180*pi;
BCleft = Angles_SLC(2); BCright = Angles_ZH(2);
interval = linspace(Angles_SLC(1),Angles_ZH(1),N);
a =  @(x,u,du)(sin(u).^2+du.^2).^(-0.5);
f = {@(x,u,du)(-cos(u).*sin(u))./sqrt(sin(u).^2+du.^2),
     @(x,u,du)-(1-2*sin(u).^2)./sqrt(sin(u).^2+du.^2)...
              +(cos(u).^2.*sin(u).^2)./(sqrt(sin(u).^2+du.^2).^3),
     @(x,u,du)(cos(u).*sin(u).*du)./(sqrt(sin(u).^2+du.^2).^3)};
[phi2,theta2] = BVP1D(interval,1,0,0,1,0,BCleft,BCright);  %% generate an intial guess
[phi2,theta2,inform] = BVP1DNL(interval,a,0,0,1,f,BCleft,BCright,theta2,
                               'MaxIter',20,'Display','iter');
figure(4); plot(phi/pi*180,90-theta/pi*180);
           xlabel('\phi [deg]'); ylabel('90-\theta [deg]');
Dtheta2 = FEM1DEvaluateDu(phi2,theta2);
disp(sprintf("BVP1DNL(): L = %f km",R*trapz(phi2,sqrt(sin(theta2).^2+Dtheta2.^2))))
-->
iteration 1,  RMS(correction) = 3.564728e-02, RMS(phi) = 2.463522e-02
iteration 2,  RMS(correction) = 9.219981e-03, RMS(phi) = 3.524064e-03
iteration 3,  RMS(correction) = 3.832637e-03, RMS(phi) = 1.109014e-03
iteration 4,  RMS(correction) = 1.851140e-03, RMS(phi) = 4.995842e-04
iteration 5,  RMS(correction) = 9.461144e-04, RMS(phi) = 2.493060e-04
iteration 6,  RMS(correction) = 5.000805e-04, RMS(phi) = 1.302717e-04
iteration 7,  RMS(correction) = 2.705884e-04, RMS(phi) = 6.998341e-05
iteration 8,  RMS(correction) = 1.490106e-04, RMS(phi) = 3.832821e-05
iteration 9,  RMS(correction) = 8.320151e-05, RMS(phi) = 2.129955e-05
iteration 10, RMS(correction) = 4.698144e-05, RMS(phi) = 1.197436e-05
iteration 11, RMS(correction) = 2.677860e-05, RMS(phi) = 6.796374e-06
iteration 12, RMS(correction) = 1.538521e-05, RMS(phi) = 3.888693e-06

BVP1DNL(): L = 8483.161942 km
```

9.35 A 1D nonlinear bending beam problem

The bending of a slender beam can be described by the angle $\alpha(s)$ as function of the arc length s. For a beam with inertia of the cross section I and a material with Young's modulus E the curvature is given by

$$\kappa(s) = \alpha'(s) = \frac{M(s)}{EI} \ ,$$

where $M(s)$ is the total moment at position s. Examine a beam attached at $(x, y) = (0, 0)$ and starting out horizontally (i.e. $\alpha(0) = 0$) with a vertical force F_2 applied at the other endpoints $s = L$. The (x, y) coordinates of the beam at $s = l$ are given by an integral

$$\begin{pmatrix} x(s) \\ y(s) \end{pmatrix} = \int_0^s \begin{pmatrix} \cos(\alpha(\tau)) \\ \sin(\alpha(\tau)) \end{pmatrix} \, d\tau \ .$$

The moment $M(s)$ is given by

$$M(s) = F_2 \left(x(L) - x(s) \right) = -F_2 \int_s^L \cos(\alpha(\tau)) \, d\tau \ .$$

Using the above equation for the bending of the beam leads to

$$\begin{aligned} \alpha'(s) &= \frac{M(s)}{EI} = \frac{F_2}{EI} \int_s^L \cos(\alpha(\tau)) \, d\tau \\ \frac{d}{ds} \alpha'(s) &= \frac{F_2}{EI} \frac{d}{ds} \int_s^L \cos(\alpha(\tau)) \, d\tau = -\frac{F_2}{EI} \cos(s) \\ -\alpha''(s) &= \frac{F_2}{EI} \cos(\alpha(s))) \ . \end{aligned}$$

This ODE has to be supplemented with the boundary conditions $\alpha(0) = \alpha'(L) = 0$ to arrive at a nonlinear BVP. For small forces the approximation $\cos(\alpha) \approx 1$ leads to the solution

$$\alpha(s) \approx \frac{F_2}{2\,EI}\,(L^2 - (L-s)^2) = \frac{F_2}{2\,EI}\,(2\,L\,s - s^2)\,.$$

Use this result for code verification or as possible starting value for Newton's method.

9.35.1 Solving the BVP using Newton's algorithm

The nonlinear boundary value problem can be solved using Newton's method. Use the idea of linearization to arrive at an iterative algorithm.

$$\begin{aligned}
-\alpha''(s) &= \frac{F_2}{EI}\cos(\alpha(s)) \quad \text{for} \quad 0 < s < L \quad \text{and} \quad \alpha(0) = \alpha'(L) = 0 \\
-\alpha''(s) - \phi''(s) &= \frac{F_2}{EI}\cos(\alpha(s) + \phi(s)) \approx \frac{F_2}{EI}\,(\cos(\alpha(s)) - \sin(\alpha(s))\,\phi(s)) \\
-\phi''(s) + \frac{F_2}{EI}\sin(\alpha_n(s))\,\phi(s) &= +\alpha_n''(s) + \frac{F_2}{EI}\,(\cos(\alpha_n(s))) \quad \text{with} \quad \phi(0) = \phi'(L) = 0 \\
\alpha_{n+1}(s) &= \alpha_n(s) + \phi(s)
\end{aligned}$$

The code `BendingBeam1D.m` implements the above algorithm. For small values of F_2 the results are as expected. But for larger values of F_2 the resulting figure is at best surprising. In Figure 210(a) find the result for $F_2 = 1.5$, and this is in fact a solution of the nonlinear BVP, but not the expected solution. Newton's method works extremely well, if the initial guess is "close enough" to the desired solution. But the last point is critical and failed for this bending beam problem for large forces F_2.

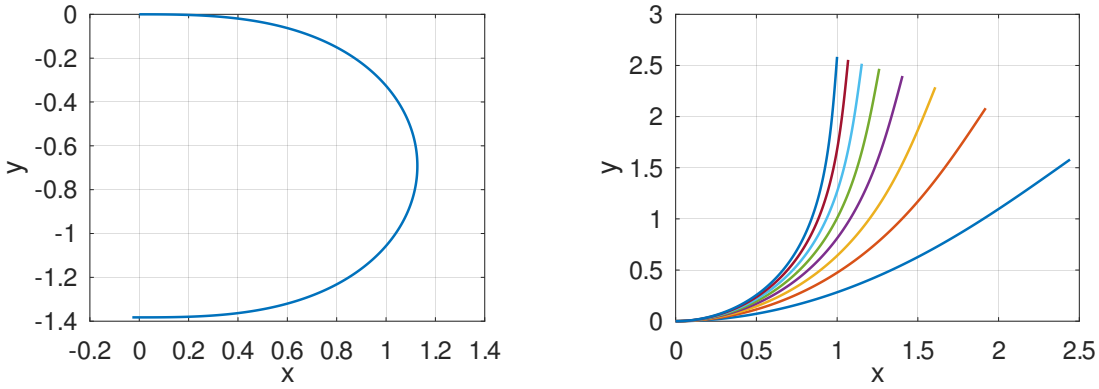
BendingBeam1D.m

```
L = 3; EI = 1;   %% as single run
F2 = 1.5; %% try values of 0.1 0.5 1.0 and 2
N = 1000; s = linspace(0,L,N);
[sn,alpha] = BVP1D(s,1,0,0,1,F2/EI,0,[0,0]);
figure(1); plot(sn,alpha); xlabel('arclength s'); ylabel('angle \alpha')
xGauss = FEM1DGaussPoints(sn);
[xGauss,Nodes2GaussU] = FEM1DGaussPoints(sn);
for jj = 1:20
  [dalpha,ddalpha] = FEM1DEvaluateDu(sn,alpha);  %% evaluate derivative at nodes
  RHS = ddalpha + F2/EI*cos(alpha);
  alphaGauss = Nodes2GaussU*alpha;               %% evaluate u at Gauss points
  [sn,phi] = BVP1D(s,1,0,+F2/EI*sin(alphaGauss),1,RHS,0,[0,0]);
  disp(sprintf('max(abs(phi)) = %g , max(abs(RHS)) = %g',max(abs(phi)), max(abs(RHS))))
  alpha = alpha + phi;
  figure(2); plot(sn,alpha); xlabel('x'); ylabel('angle \alpha')
  pause(0.2)
endfor
x = cumtrapz(sn,cos(alpha)); y = cumtrapz(sn,sin(alpha));
figure(3); plot(x,y); xlabel('x'); ylabel('y')
```

To obtain a reliable solution for $F_2 = 2$ it is advisable to use a parameterized approach. In the code below the value of F_2 is increased form 0 to 2.0 is steps of 0.25 and at each new level of F_2 the result of the previous computation is used as starting value for $\alpha(s)$. Find the graphical results in Figure 210(b).

BendingBeam1D.m

```
L = 3; EI = 1; F2_List = [0.25:0.25:2];  %% a parametrized approach
N = 100; s = linspace(0,L,N)';
[sn,alpha] = BVP1D(s,1,0,0,1,F2_List(1)/EI,0,[0,0]);
figure(1); plot(sn,alpha); xlabel('arclength s'); ylabel('angle \alpha')
xGauss = FEM1DGaussPoints(sn);
figure(3); clf; hold off
for F2 = F2_List
  for jj = 1:10
    [dalpha,ddalpha] = FEM1DEvaluateDu(sn,alpha);
    RHS = ddalpha + F2/EI*cos(alpha);
```



(a) the result for $F_2 = 1.5$

(b) the parametric results for F_2 up to 2

Figure 210: The solutions for a bending beam

```
    alphaGauss = pwquadinterp(sn,alpha,xGauss); %% evaluate u at Gauss points
    [sn,phi] = BVP1D(s,1,0,F2/EI*sin(alphaGauss),1,RHS,0,[0,0]);
    alpha = alpha + phi;
  endfor % jj
  x = cumtrapz(sn,cos(alpha)); y = cumtrapz(sn,sin(alpha));
  figure(3); plot(x,y); xlabel('x'); ylabel('y'); hold on
endfor %% F2
hold off
```

9.35.2 Solving the BVP with the command `BVP1DNL()`

The above bending beam problem can be solved with the help of `BVP1DNL()`. There is less coding involved, but the problem of converging to a non desired solution remains. Observing the results of the iterations confirms that at first the algorithm is "searching for a solution", but once it is close to a solution quadratic convergence sets in, i.e. the number of stable digits is doubled at each step. This is expected for Newton's method.

BendingBeam1D.m

```
F2 = 1.5; %% try values of 0.1 0.25 0.5 1.0 and 2
f = {@(s,alpha)F2/EI*cos(alpha), @(s,alpha)-F2/EI*sin(alpha)};
N = 100; s = linspace(0,L,N)';
%%% generate a good initial guess
[sn,alpha0] = BVP1D(s,1,0,0,1,F2/EI,0,[0,0]);  %% as solution of linear BVP
%% alpha0 = @(s)F2/(2*EI)*(L^2-(L-s).^2);      %% use the analytical solution
[sn,alpha,inform] = BVP1DNL(s,1,0,0,1,f,0,[0,0],alpha0,'tol',1e-8,'MaxIter',50,'Display','iter');
inform
figure(1); plot(sn,alpha); xlabel('arclength s'); ylabel('angle \alpha')
x = cumtrapz(sn,cos(alpha)); y = cumtrapz(sn,sin(alpha));
figure(3); plot(x,y); xlabel('x'); ylabel('y');
```

The output of the above code illustrates the long search for a solution by Newton's method, and the quadratic convergence as soon as close to one of the possible solutions, starting at iteration 11.

```
iteration 1,  RMS(correction) = 2.148167e+00, RMS(phi) = 3.658963e+01
iteration 2,  RMS(correction) = 1.534256e+00, RMS(phi) = 1.281984e+00
iteration 3,  RMS(correction) = 3.233207e-01, RMS(phi) = 6.380582e-01
iteration 4,  RMS(correction) = 1.715649e-01, RMS(phi) = 6.706732e-01
iteration 5,  RMS(correction) = 1.442408e-01, RMS(phi) = 5.281602e-01
iteration 6,  RMS(correction) = 1.380244e-01, RMS(phi) = 4.177463e-01
iteration 7,  RMS(correction) = 1.388972e-01, RMS(phi) = 3.168041e-01
iteration 8,  RMS(correction) = 1.277986e-01, RMS(phi) = 2.140754e-01
iteration 9,  RMS(correction) = 8.810050e-02, RMS(phi) = 1.124749e-01
```

```
iteration 10, RMS(correction) = 3.198417e-02, RMS(phi) = 3.424090e-02
iteration 11, RMS(correction) = 3.261890e-03, RMS(phi) = 3.282558e-03
iteration 12, RMS(correction) = 3.022821e-05, RMS(phi) = 3.022996e-05
iteration 13, RMS(correction) = 2.563226e-09, RMS(phi) = 2.562500e-09

inform =  scalar structure containing the fields:
          info    = 1
          iter    = 13
          AbsError = 2.5632e-09
```

The parametric approach will again reliably generate the desired solution with very few iterations for each value of F_2.

BendingBeam1D.m

```
L = 3; EI = 1;
F2_List = [0:0.25:2];
N = 100; s = linspace(0,L,N)'; sn = sort([s; s(1:end-1)+diff(s)/2]);
alpha = 0;
for F2 = F2_List
  f = {@(s,al)F2/EI*cos(al), @(s,al)-F2/EI*sin(al)};
  [sn,alpha,inform] = BVP1DNL(s,1,0,0,1,f,0,[0,0],alpha);
  inform
endfor
inform
figure(1); plot(sn,alpha*180/pi); xlabel('arclength s'); ylabel('angle \alpha [deg]')
x = cumtrapz(sn,cos(alpha)); y = cumtrapz(sn,sin(alpha));
figure(3); plot(x,y); xlabel('x'); ylabel('y');
```

9.35.3 Solving the BVP as final value of a dynamic problem, using `IBVP1DNL()`

The equation for the bent beam $-\alpha''(s) = \frac{F_2}{EI}\cos(\alpha(s))$ can be looked at as the steady state solution of an artificial dynamic problem.

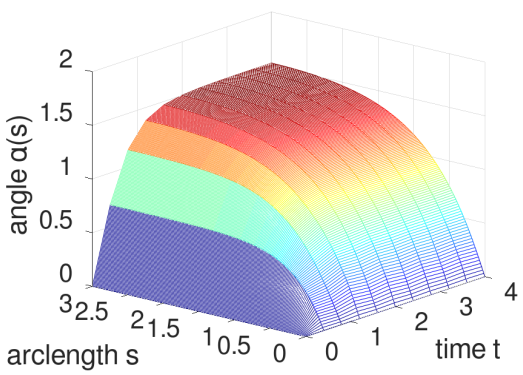
$$\begin{aligned} \tfrac{\partial}{\partial t}\alpha(s,t) - \tfrac{\partial^2}{\partial s^2}\alpha(s,t) &= \tfrac{F_2}{EI}\cos(\alpha(s,t)) && \text{for } 0<s<L \text{ and } t>0 \\ \alpha(0,t) = \tfrac{\partial}{\partial s}\alpha(L,t) &= 0 && \text{for } t>0 \\ \alpha(s,0) &= 0 && \text{for } 0<s<L \end{aligned}$$

If this solution converges to a final solution $\alpha_\infty(s) = \lim_{t\to\infty}\alpha(s,t)$ then $\alpha_\infty(s)$ is a solution of the bending beam problem[58]. The virtual time t takes over the role of a parameter, just like the force F_2 in the above approach. The code `BeamNonlinear1DDynamic.m` below implements this approach and Figure 211 confirms the expected behavior. The left part of the figure shows the stepwise approximation of the steady state solution. By selecting other initial values $\alpha_0(s) = u_0(s)$ one can search for different solutions. As example examine the result for $u_0(s) = \frac{-5\,\pi}{4}\,(1 - (\frac{s}{L}-1)^2)$.

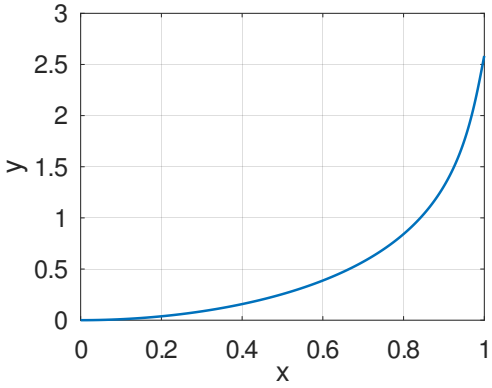
BeamNonlinear1DDynamic.m

```
%% solve the nonlinear beam problem as dynamic problem
L = 3; EI = 1; F2 = 2.0;
Interval = linspace(0,L,100)'; T0 = 0; Tend = 4; steps = [10,1];
f = {@(x,t,u)F2*cos(u), @(x,t,u)-F2*sin(u)};
u0 = 0;                              %% the naive initial guess
%%u0 = @(s,t)pi/2*(1-(s/L-1).^2);     %% with a better initial guess
%%u0 = @(s,t)-5/4*pi*(1-(s/L-1).^2);  %% to aim for a different solution
[s,u_all,t] = IBVP1DNL(Interval,1,EI,0,0,1,f,0,[0,0],u0,T0,Tend,steps);
figure(1); mesh(t,s,u_all); view([-50,20])
           xlabel('time t'); ylabel('arclength s'); zlabel('angle \alpha(s)')
u = u_all(:,end);
x = cumtrapz(s,cos(u)); y = cumtrapz(s,sin(u));
figure(2); plot(x,y); xlabel('x'); ylabel('y');
```

[58] For symmetric, positive definite matrices one can verify that for solutions of $\mathbf{W}\frac{d}{dt}\vec{u}(t) = -\mathbf{A}\vec{u}(t) + \mathbf{M}\cos(\vec{u}(t))$ the expression $G(\vec{u}(t)) := \frac{1}{2}\langle \mathbf{A}\,\vec{u}(t)\,,\,\vec{u}(t)\rangle - \sum_i (\mathbf{M}\,\sin(\vec{u}(t)))_i$ is decreasing. If it converges to a steady state $\vec{u}_\infty$ then $\mathbf{A}\vec{u}_\infty = \mathbf{M}\cos(\vec{u}_\infty)$.



(a) the angles $\alpha(s,t)$



(b) the final solution $(x(s), y(s))$

Figure 211: Time evolution and the final solution for a dynamic bending beam

9.36 Mass transfer in a porous catalyst

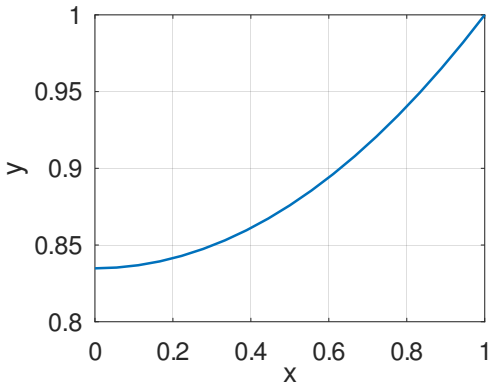
In [KubiHlav08, p. 92, Example 4.3, p. 255, Example 5.7] a nonlinear boundary value problem is examined, describing the mass transfer in a porous catalyst.

$$-y'' - \frac{a}{x}\,y' = -\alpha^2\, y\, \exp\left(\frac{\gamma\,\beta\,(1-y)}{1+\beta\,(1-y)}\right) \quad \text{with} \quad y'(0) = 0 \quad \text{and} \quad y(1) = 1$$

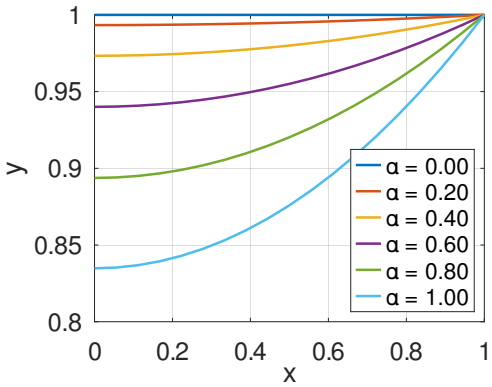
For $\alpha = 0$ there is the trivial solution $y(x) = 1$. The nonlinear contribution and its partial derivative are given by

$$\begin{aligned} f(y) &= -\alpha^2\, y\, \exp\left(\frac{\gamma\,\beta\,(1-y)}{1+\beta\,(1-y)}\right) \\ \frac{\partial}{\partial y} f(y) &= -\alpha^2\, \exp\left(\frac{\gamma\,\beta\,(1-y)}{1+\beta\,(1-y)}\right) + \alpha^2\, y\, \exp\left(\frac{\gamma\,\beta\,(1-y)}{1+\beta\,(1-y)}\right) \frac{-\gamma\,\beta\,(1+\beta\,(1-y)) + \gamma\,\beta\,(1-y)\,\beta}{(1+\beta\,(1-y))^2} \\ &= -\alpha^2\, \exp\left(\frac{\gamma\,\beta\,(1-y)}{1+\beta\,(1-y)}\right) \left(1 + y\, \frac{-\gamma\,\beta}{(1+\beta\,(1-y))^2}\right) . \end{aligned}$$

In [KubiHlav08] the values $a = 2$, $\gamma = 20$ and $\beta = 0.05$ are used and the parameter $0 \le \alpha \le 1$ is increased from 0 to 1. `BVP1DNL()` can solve the problem directly for $\alpha = 1$, but a parametric solution is possible too. Find the solutions of the code below in Figure 212. The results coincide with the numbers in [KubiHlav08, Example 5.7].



(a) the solution for $\alpha = 1$



(b) The solution for different values of α

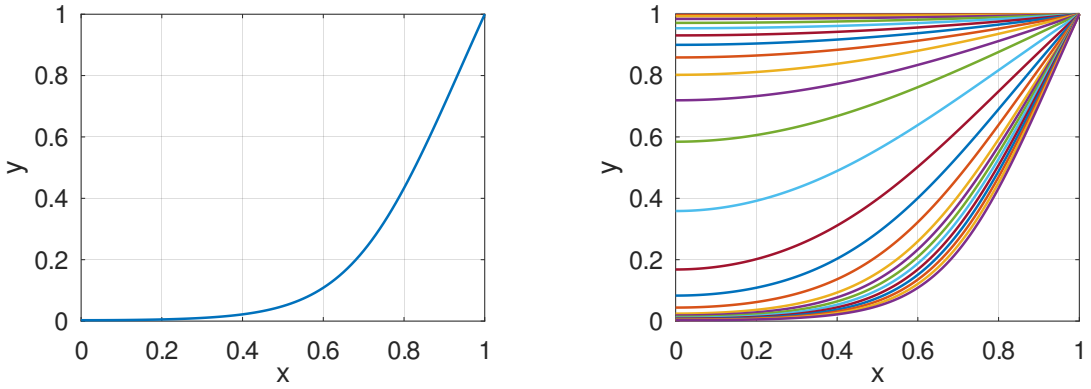
Figure 212: A BVP describing mass transfer in a porous catalyst

PorousCatalyst.m

```
N = 10; interval = linspace(0,1,N)';
a  = 2; gamma = 20; beta = 0.05;
BCleft = [0,0]; BCright = 1;
y = 1;
figure(2); clf; hold on; box on; xlabel('x'); ylabel('y')
for alpha = 0:0.2:1
  f = {@(x,y) -alpha^2*y.*exp(gamma*beta*(1-y)./(1+beta*(1-y))),
    @(x,y) -alpha^2   *exp(gamma*beta*(1-y)./(1+beta*(1-y))).*...
            (1-gamma*beta*y./( (1+beta*(1-y)).^2) )};
  [x,y] = BVP1DNL(interval,1,@(x)-a./x,0,1,f,BCleft,BCright,y);
  plot(x,y)
  disp(sprintf('alpha = %#3.2f, y(0) = %#g',alpha,y(1)))
endfor
legend('\alpha = 0.00','\alpha = 0.20','\alpha = 0.40','\alpha = 0.60',
       '\alpha = 0.80','\alpha = 1.00','location','southeast')
figure(1); plot(x,y); xlabel('x'); ylabel('y'); box on
-->
alpha = 0.00, y(0) = 1.00000
alpha = 0.20, y(0) = 0.993333
alpha = 0.40, y(0) = 0.973339
alpha = 0.60, y(0) = 0.940066
alpha = 0.80, y(0) = 0.893713
alpha = 1.00, y(0) = 0.834810
```

For a second set of values $a = 0$, $\gamma = 20$ and $\beta = 0.4$ the algorithm converges nicely up to $\alpha \approx 0.37$. For $\alpha > 0.4$ convergence is difficult to obtain.

In [KubiHlav08, Example 4.3] find the values for $a = 2$, $\gamma = 20$, $\beta = 0.2$ and $\alpha = 2$. A rather sophisticated iteration scheme was used in [KubiHlav08]. By increasing the value of α from 0 to 2 the results in the last row of [KubiHlav08, Table 4-12] can be reproduced by `BVP1DNL()`. Find the results of the code below in Figure 213.



(a) the solution for $\alpha = 2$

(b) The solution for different values of α

Figure 213: A BVP describing mass transfer in a porous catalyst, second setup

PorousCatalyst.m

```
N = 51; interval = linspace(0,1,N)';
BCleft = [0,0]; BCright = 1;
a = 2; gamma = 20; beta = 0.2;
y = 1;
figure(2); clf; hold on; box on; xlabel('x'); ylabel('y')
for alpha = [0:0.1:1.6, 1.65:0.05:2]
  f = {@(x,y) -alpha^2*y.*exp(gamma*beta*(1-y)./(1+beta*(1-y))),
      @(x,y) -alpha^2   *exp(gamma*beta*(1-y)./(1+beta*(1-y))).*...
              (1-gamma*beta*y./( (1+beta*(1-y)).^2) )};
```

```
  [x,y] = BVP1DNL(interval,1,@(x)-a./x,0,1,f,BCleft,BCright,y);
  plot(x,y)
endfor
figure(1); plot(x,y); xlabel('x'); ylabel('y'); box on
x_n = [0:0.2:0.8]; y_n = pwquadinterp(x,y,x_n);
disp(sprintf('y(%g)=%5.4e, y(%g)=%5.4e, y(%g)=%5.4e, y(%g)=%5.4e, y(%g)=%5.4e',
             x_n(1),y_n(1),x_n(2),y_n(2),x_n(3),y_n(3),x_n(4),y_n(4),x_n(5),y_n(5)))
-->
y(0)=2.7643e-3, y(0.2)=5.3166e-3, y(0.4)=2.1871e-2, y(0.6)=1.0853e-1, y(0.8)=4.3316e-1
```

9.37 Troesch's equation

In [IzadSuayNoei21] or [KubiHlav08, p. 244, Example 5.4] Troesch's nonlinear boundary value problem is examined.

$$y''(x) = \alpha \sinh(\alpha\, y(x)) \quad \text{with} \quad y(0) = 0 \quad \text{and} \quad y(1) = 1$$

The results for $\alpha = 0.5$ in [IzadSuayNoei21, Table 1] can be generated by `BVP1DNL()` with very few lines of code.

Troesch.m

```
N = 11; interval = linspace(0,1,N)'; BCleft = 0; BCright = 1; alpha = 0.5;
f = {@(x,y) -alpha*sinh(alpha*y), @(x,y)-alpha^2*cosh(alpha*y)};
[x,y,inform] = BVP1DNL(interval,1,0,0,1,f,BCleft,BCright,@(x)x);
figure(1); plot(x,y); xlabel('x'); ylabel('y(x)')
xd = [ 0.1:0.1:0.9]';  yd = pwquadinterp(x,y,xd);
Results = [xd,yd]
-->
Results =    0.1   0.095944
             0.2   0.192129
             0.3   0.288794
             0.4   0.386185
             0.5   0.484547
             0.6   0.584133
             0.7   0.685201
             0.8   0.788017
             0.9   0.892854
```

To obtain the results for $\alpha = 1$ in [IzadSuayNoei21, Table 2] just change the value of α in the above code.

In [KubiHlav08, Example 5.4] a shooting method is used to determine α as function of $y'(0)$. The results in [KubiHlav08, Table 5-6] are confirmed by the code below, using `BVP1DNL()`.

Troesch.m

```
N = 51; interval = linspace(0,1,N)'; BCleft = 0; BCright = 1;
figure(1); clf; hold on; box on
y = 1;
for alpha = [0.792, 1.151 1.753 2.394 3.308 4.129 5.0]
f = {@(x,y) -alpha*sinh(alpha*y), @(x,y)-alpha^2*cosh(alpha*y)};
[x,y] = BVP1DNL(interval,1,0,0,1,f,BCleft,BCright,y);
figure(1); plot(x,y); xlabel('x'); ylabel('y(x)'); drawnow()
[y0,dy0] = pwquadinterp(x,y,0);
disp(sprintf("alpha = %#6.5g, y'(0)= %g",alpha,dy0))
endfor
-->
alpha = 0.79200, y'(0)= 0.900027
alpha = 1.1510,  y'(0)= 0.800195
alpha = 1.7530,  y'(0)= 0.600109
alpha = 2.3940,  y'(0)= 0.400003
alpha = 3.3080,  y'(0)= 0.200008
alpha = 4.1290,  y'(0)= 0.0999462
alpha = 5.0000,  y'(0)= 0.0457183
```

9.38 Motion of a string

Examine the motion of a string of length 5. For 2 seconds apply a force to the very left section ($0 \le x \le 1$). Use a small damping factor, e.g. 0.25. A possible IBVP describing this setup is

$$\begin{aligned} \frac{\partial^2}{\partial t^2}\,u(x,t) + 0.25\,\frac{\partial}{\partial t}\,u(x,t) - \frac{\partial^2}{\partial x^2}\,u(x,t) &= f(x,t) && \text{for } 0 < x < 5 \text{ and } 0 < t < 10 \\ u(0,t) \;=\; u(5,t) &= 0 && \text{for } 0 \le t \le 10 \\ u(x,0) \;=\; \frac{\partial}{\partial t}\,u(x,0) &= 0 && \text{for } 0 \le x \le 5 \end{aligned}$$

with the driving force function

$$f(x,t) = \begin{cases} \cos(\frac{\pi}{2}\,x)\,\sin(2\,\pi\,t) & \text{for } 0 \le x \le 1 \text{ and } 0 \le t \le 2 \\ 0 & \text{otherwise} \end{cases}$$

This problem is examined with the help of `I2BVP1D()`. Find the graphical result in Figure214. Observe the two cos–shaped pulses moving with speed $+1$. Then they are reflected at the border at $x = 5$ and travelling with speed -1. The amplitudes are slowly decaying, caused by the damping.

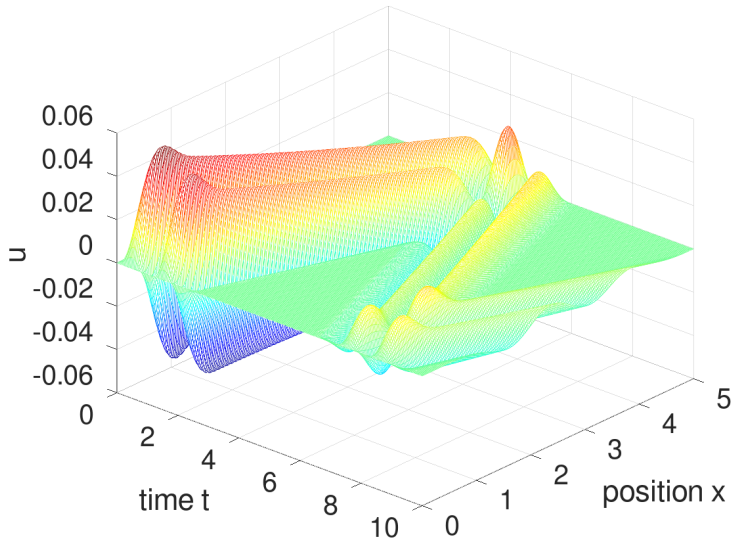


Figure 214: Motion of a string, excited by an initial pulse close to $x = 0$

MovingString.m

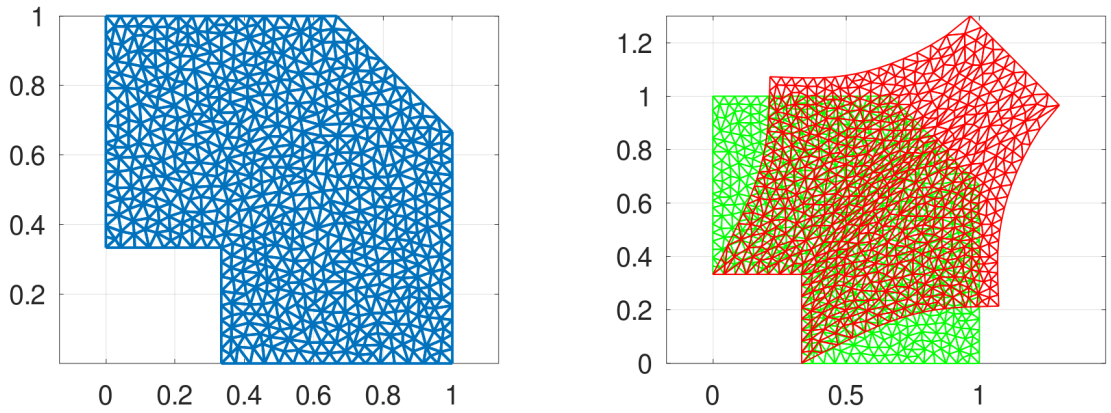
```
N = 101; x_max = 5; interval = linspace(0,x_max,N)';
omega = 2*pi;
f = @(x,t)(cos(x*pi/2).*(x<=1))*(sin(omega*t)*(t<=2));
u0 = 0;   u1 = 0;
w2 = 1; w1 = 0.25; a = 2; b = 0; c = 0; d = 1;
BCleft = [0]; BCright = [0];
t0 = 0; tend = 10; steps = [250,10];
[x,u,t] = I2BVP1D(interval,w2,w1,a,b,c,d,f,BCleft,BCright,u0,u1,t0,tend,steps);

figure(1); mesh(t,x,u); xlabel('time t'); ylabel('position x'); zlabel('u')
           xlim([min(t),max(t)]); ylim([min(x),max(x)]); view([20,20])
```

9.39 A plane stress example by Wait and Mitchell

This example is taken from [WaitMitc85, §4.4.3] with the material parameters suitable for rubber. The domain is visible in Figure 215.

- On the two sections connecting the points $(0, \frac{1}{3})$, $(\frac{1}{3}, \frac{1}{3})$ and $(\frac{1}{3}, 0)$ zero displacements are enforced.
- On the $45°$ degree segment connecting $(\frac{2}{3}, 1)$ with $(1, \frac{2}{3})$ the horizontal and vertical displacements are $+0.3$. Thus the edge is displaced by $0.3\sqrt{2} \approx 0.42$.
- The other four sections of the boundary are assumed to be force free.
- As a consequence the solid is severely deformed, visible on the right in Figure 215.



(a) the original mesh (b) the deformed mesh

Figure 215: The original and deformed Mesh for a plane stress example

The problem is solved using a linear plane stress model. Find the implementation in the code `WaitMitchell.m` below.

- Define the material parameters for a rubber like material: Young's modulus E and Poisson's ratio ν.
- Define the domain and the boundary conditions. Then call `CreateMeshTriangle()` to generate the mesh, shown in Figure 215. The mesh consists of cubic elements. There are $5948 + 5948 = 11'896 \approx 12'000$ degrees of freedom. Use `Mesh.nDOF` to obtain this information.
- A function

$$\text{Disp}(x, y) = \begin{cases} 0.3 & \text{if} \quad x + y > 1 \\ 0 & \text{if} \quad x + y \leq 1 \end{cases}$$

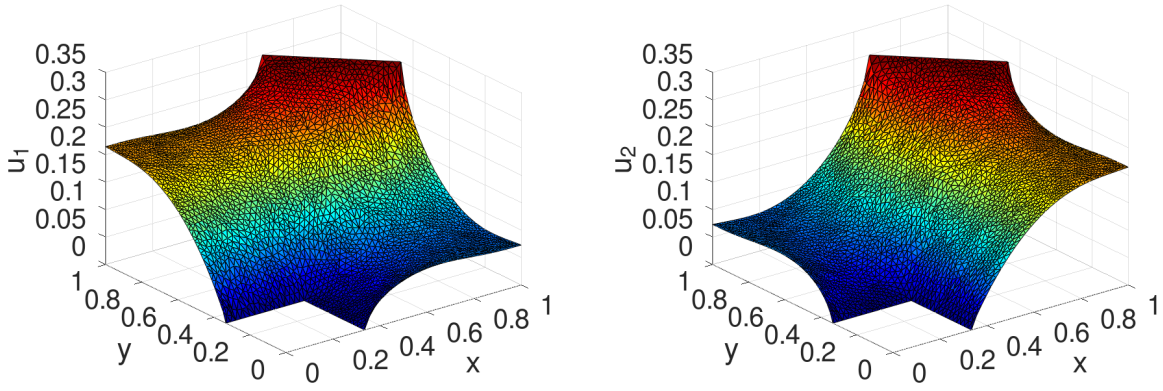
  is defined to evaluate the known displacements on the boundary.
- A call of `PlaneStress()` will evaluate the horizontal and vertical displacements (u_1, u_2). These are shown in Figure 216.

WaitMitchell.m

```
E = 3e6; nu = 0.45;     %% rubber
Dom = [0,1/3,-11;1/3,1/3,-11;1/3,0,-22;1,0,-22;1,2/3,-11;2/3,1,-22;0,1,-22];
Mesh = CreateMeshTriangle('Domain',Dom,1e-3);
figure(1); FEMtrimesh(Mesh); axis equal
Mesh = MeshUpgrade(Mesh,'cubic');

function res = Disp(xy)
  res = 0.3 .*(sum(xy,2)>1) ; %% if x+y>1
endfunction

[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{'Disp','Disp'},{0,0});
figure(2); FEMtrisurf(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(3); FEMtrisurf(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')
figure(100); ShowDeformation(Mesh,u1,u2,1); axis equal
```



(a) the horizontal displacement u_1

(b) the vertical displacement u_2

Figure 216: The displacements for a plane stress example

- With the displacements use `EvaluateStress()` to determine the two normal stresses σ_x and σ_x and the shearing stress τ_{xy}.
- Then generate images, leading to Figure 217. The stress concentration at the different corners is obvious.

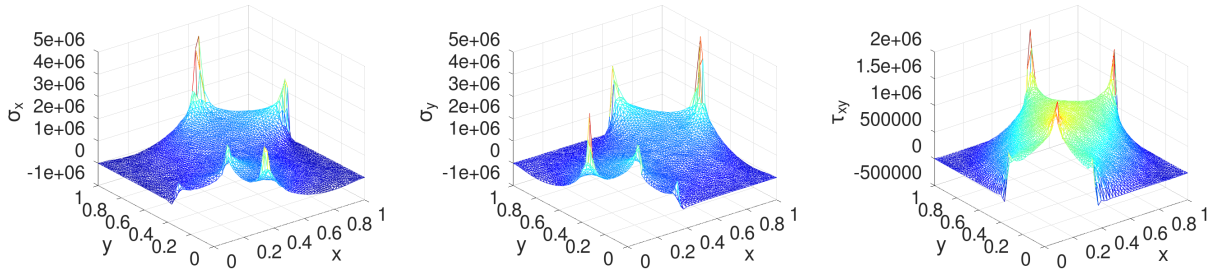


Figure 217: The stresses for the Wait–Mitchell plane stress example

WaitMitchell.m

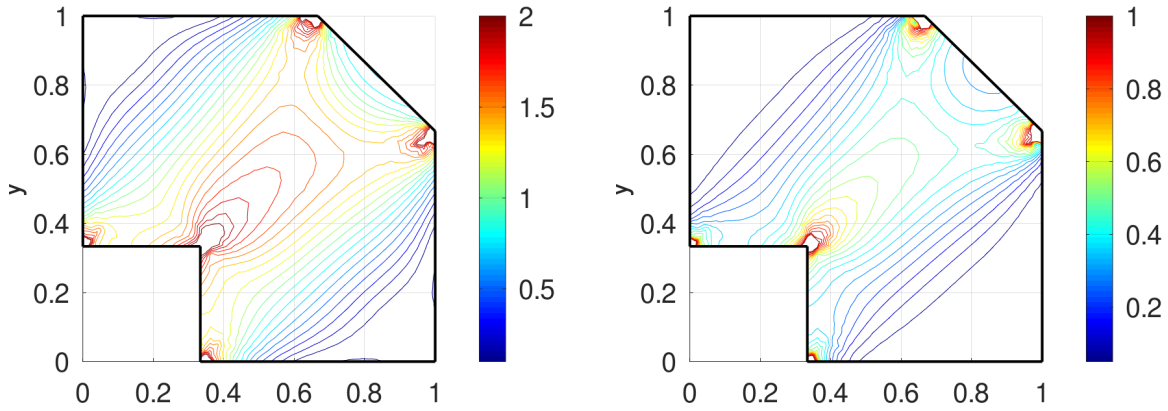
```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu);
figure(4); FEMtrimesh(Mesh,sigma_x); xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(5); FEMtrimesh(Mesh,sigma_y); xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(6); FEMtrimesh(Mesh,tau_xy); xlabel('x'); ylabel('y'); zlabel('\tau_{xy}')
```

To analyze the situation more information might be useful. Find the graphical results in Figure 218.

- Use `EvaluateVonMises()` to determine the von Mises stress. The values have to be compared with the yield stress of the material.
- The energy density generated by `EvaluateEnergyDensity()` shows the concentration at the corners too.
- With `FEMIntegrate()` the total elastic energy $E_{elast} \approx 2.0 \cdot 10^5$ N m can be determined. The displacement of the $45°$ edge is $D = 0.3\sqrt{2}$ and the total force F (per meter of thickness of the block) is given by

$$F = \frac{2\,E_{elast}}{D} = \frac{2\,E_{elast}}{0.3\sqrt{2}}\,.$$

The results by the code `WaitMitchell.m` is $F \approx 9.6 \cdot 10^5\,\frac{\mathrm{N}}{\mathrm{m}}$.



(a) the von Mises stress [MPa]

(b) the energy density

Figure 218: The von Mises stress and the energy density for a plane stress example

WaitMitchell.m

```
VonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(Mesh,u1,u2);
EnergyDensity = EvaluateEnergyDensity(Mesh,eps_xx,eps_yy,eps_xy,E,nu);
ElasticEnergy = FEMIntegrate(Mesh,EnergyDensity)
Force = 2*ElasticEnergy/(sqrt(2)*0.3)
Contour = [Dom(:,1:2);Dom(1,1:2)];
figure(7); clf; FEMtricontour(Mesh,VonMises/1e6,[0:0.1:2]); xlabel('x'); ylabel('y');
           hold on; plot(Contour(:,1),Contour(:,2),'k'); colorbar();
figure(8); clf;FEMtricontour(Mesh,EnergyDensity/1e6,[0:0.05:1]);xlabel('x');ylabel('y');
           hold on; plot(Contour(:,1),Contour(:,2),'k'); colorbar();
```

Of particular interest is the $45°$ line connecting the points $(\frac{1}{3},\frac{1}{3})$ and $(\frac{5}{6},\frac{5}{6})$. Using the transformation rule for the strain tensor[59] leads to

$$\varepsilon_{45} = \frac{1}{2}\left(\varepsilon_{xx} + \varepsilon_{yy} + 2\,\varepsilon_{xy}\right) .$$

Integrating this normal strain along the connecting line should lead to the total displacement, i.e.

$$\int_{1/3}^{5/6} \varepsilon_{45}(x,x)\,\sqrt{2}\,dx \approx 0.3\,\sqrt{2} .$$

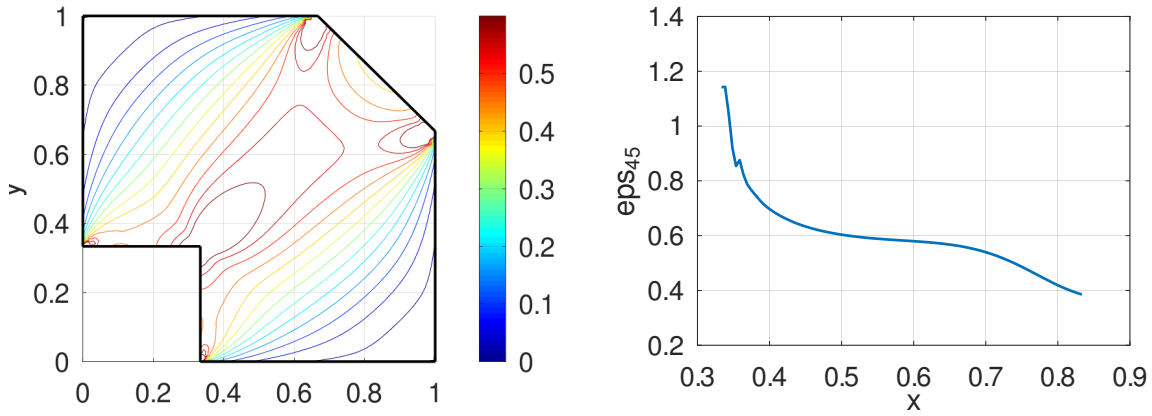
This is confirmed by the numerical integration using `trapz()`. Find the contour lines of ε_{45} on the full domain and the values along the diagonal line in Figure 219. The very large values of the strain clearly indicate that the linear material model used is not appropriate. For large deformation nonlinear material properties have to be taken into account.

WaitMitchell.m

```
eps_45 = (eps_xx+eps_yy+2*eps_xy)/2;
x = linspace(1/3,1-1/6)';
eps_45line = FEMgriddata(Mesh,eps_45,x,x);
```

[59] Strain is a tensor of order two and thus for a rotation angle $\phi = \frac{\pi}{4} = 45°$ obtain

$$\begin{aligned}
\begin{bmatrix} \varepsilon'_{x'x'} & \varepsilon'_{x'y'} \\ \varepsilon'_{x'y'} & \varepsilon'_{y'y'} \end{bmatrix} &= \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx}+\varepsilon_{xy} & -\varepsilon_{xx}+\varepsilon_{xy} \\ \varepsilon_{xy}+\varepsilon_{yy} & -\varepsilon_{xy}+\varepsilon_{yy} \end{bmatrix} \\
\varepsilon_{45} = \varepsilon'_{x'x'} &= \frac{1}{2}\left(\varepsilon_{xx} + 2\,\varepsilon_{xy} + \varepsilon_{yy}\right)
\end{aligned}$$



(a) the nornal strain in 45° direction

(b) normal strain along diagonal

Figure 219: The normal strain in the 45° direction for a plane stress example

```
figure(9); clf; FEMtricontour(Mesh,eps_45,[0:0.05:0.6]); xlabel('x'); ylabel('y');
           hold on; plot(Contour(:,1),Contour(:,2),'k'); colorbar();
figure(10); plot(x,eps_45line); xlabel('x'); ylabel('eps_{45}')
Stretch = [trapz(sqrt(2)*x,eps_45line),sqrt(2)*0.3]
-->
Stretch =   0.4209   0.4243
```

9.40 Thermal bending of a bimetallic strip

On en.wikipedia.org/wiki/Bimetallic_strip find a good description of a bimetrallic strip:
A bimetallic strip or bimetal strip is a strip that consists of two strips of different metals which expand at different rates as they are heated. The different expansion rates cause the strip to bend one way if heated, and in the opposite direction if cooled below its initial temperature. Thus, a bimetal strip converts a temperature change into mechanical displacement. The metal with the higher coefficient of thermal expansion is on the outer side of the curve when the strip is heated and on the inner side when cooled. Common applications include temperature sensing (thermometer) and regulation (thermostat).

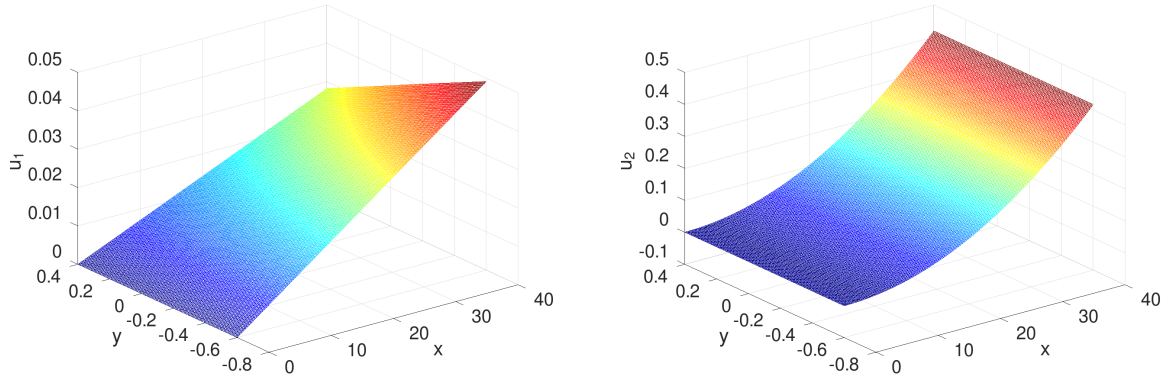
The code `Bimetallic.m` shown below examines the following situation:

- The strip of length 40 mm and height 1 mm consists of a bottom layer of copper (height 0.6 mm, $-0.6 < y < 0$) an a top layer of steel (height 0.4 mm, $0 < y < 0.4$).
- A plane stress situation is assumed. The strip is clamped[60] at the left edge ($x = 0$) and free to move on the other edges.
- It is heated up by 70 $C°$.

The results are:

- The maximal horizontal displacement u_1 is ≈ 0.05 mm. This is the stretching of the strip by thermal expansion. It is visible in Figure 220(a). The copper section is stretched more than the steel section.
- The maximal vertical displacement u_2 is ≈ 0.42 mm. This is the bending of the bimetallic strip and could be used to determine the temperature. It is considerably larger than the horizontal expansion! It is visible in Figure 220(b).
- In Figure 221 find graphs of the normal, horizontal stress σ_x. This stress leads to a bending moment and thus to the bending upwards of the strip. Observe the different values in the copper ($-0.6 < y < 0$) and steel ($0 < y < 0.4$) sections.

[60] Only the x–displacement is set to zero along all of the y–axis. Only one point is restricted in the y–direction, by using `MeshAddConstraint()`. This is to avoid the visual effect of the Saint–Venant effect along the y–axis.



(a) the horizontal displacement u_1

(b) the vertical displacement u_2

Figure 220: The displacements for a heated bimetallic strip

- In Figure 221(b) the effect of the different expansion coefficients α for copper and steel can be observed:
  - The bottom layer of copper at $y < 0$ tries to expand more than the steel at $0 < y$ and thus is pulled back. Thus the normal stress σ_x is negative, i.e. a compressive stress.
  - The top layer of steel at $0 < y$ tries to expand less than the copper at $y < 0$ and thus is pushed forward. Thus the normal stress σ_x is positive, i.e. observe traction.
  - The surface in 221(a) show surprising effects along the axis $y = 0.6$, where the material constants have jumps. The strains and stresses are evaluated at the nodes and along $y = 0$ with `EvaluateStress()`[61]. Thus the average of the values in the neighboring triangles is used. The strains (e.g. $\frac{\partial u_1}{\partial x}$, $\frac{\partial u_2}{\partial y}$) are not continuous across boundaries of the triangular elements. The effect of these jumps is visible in Figure 221(a).
  - When using `EvaluateStress()` with an option `curve` one can evaluate strictly inside the elements[62], leading to the better results in Figure 221(b).
  - Find more information on this effect in Sections 4.2.5 (page 75) and Section 4.18.6 (page 100).
- In 1925 S. Timoshenko generated an analytical formula for the curvature κ of a bimetallic strip, find the result in e.g. [KhatMirc17]. With the parameters $n = \frac{E_1}{E_2}$ and $m = \frac{D_1}{D_2}$ use

$$\kappa = \frac{3\,(1+m)^2 + (1+n\cdot m)\cdot(m^2 + \frac{1}{n\cdot m})}{6\,(\alpha_1 - \alpha_2)\cdot\Delta T\cdot(1+m)^2}\ .$$

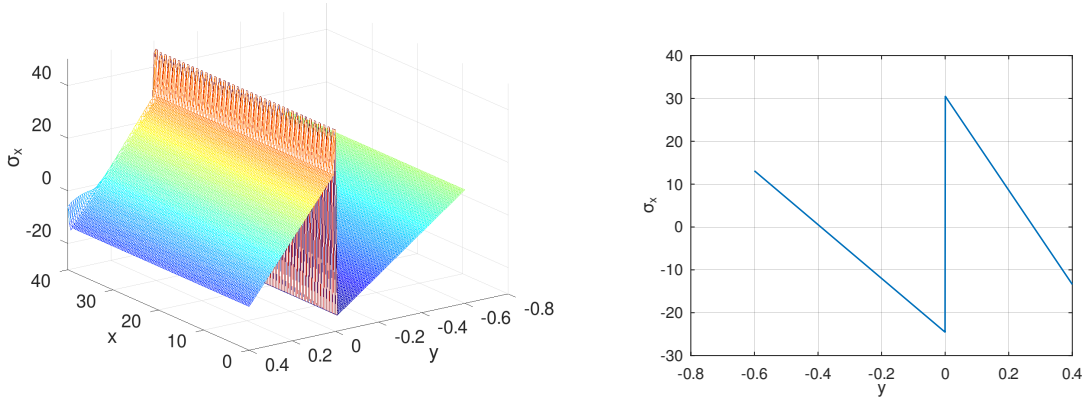
  For the example in `Bimetallic.m` this leads to $\kappa \approx 5.23\cdot 10^{-4}$. The code `Bimetallic.m` generates Figure 222 for the vertical displacement and a linear regression leads to a value of $\approx 5.23\cdot 10^{-4}$ for the second derivative, i.e it coincides the Timoshenko's analytical formula.
- Neither Timoshenko's result or the results by the FEM code depend on the values ν_1 or ν_2 of Poisson ratios[63].

Bimetallic.m

```
E1 = 120e3; nu1 = 0.34; alpha1 = 16e-6; %% copper
E2 = 210e3; nu2 = 0.30; alpha2 = 11e-6; %% steel
L = 40; D1 = 0.6; D2 = 0.4; DT = 70;
%%nu1 = 0; nu2 = 0;
E  = @(xy) E1*(xy(:,2)<0) +  E2*(xy(:,2)>= 0);
nu = @(xy)nu1*(xy(:,2)<0) + nu2*(xy(:,2)>= 0);
ThermalCoeff = @(xy)DT*alpha1*(xy(:,2)<0) + DT*alpha2*(xy(:,2)>= 0);

Mesh = CreateMeshRect(linspace(0,L,41),linspace(-D1,D2,41),-22,-22,-12,-22);
```

[61] The calling sequence is `EvaluateStress()` ⟶ `EvaluateStrain()` ⟶ `FEMEvaluateGradient()`.
[62] The calling sequence is `EvaluateStress()` ⟶ `EvaluateStrain()` ⟶ `FEMgriddata()`.
[63] Let me know if you have a mechanical justification for this observation.



(a) the normal stress σ_x as surface

(b) the normal stress σ_x along a slice

Figure 221: The normal stress σ_x as surface and along a slice at $x = L/2$

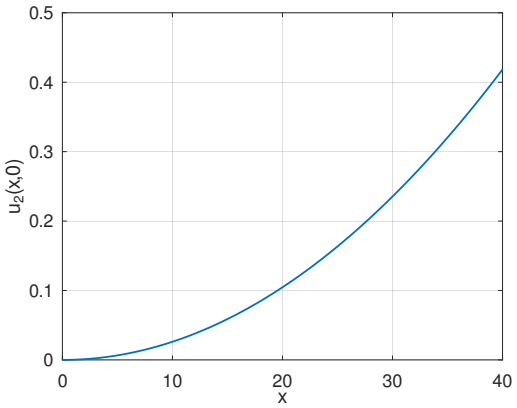


Figure 222: The vertical displacements $u_2(x, 0)$ along the axis $y = 0$

```
Mesh = MeshAddConstraint(Mesh,[0,0],[-1,-1]);
Mesh = MeshUpgrade(Mesh,'cubic');

[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{0,0},{0,0},'thermal',ThermalCoeff);
figure(1); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(2); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')
Max_u1u2 = [max(u1),max(u2)]

[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',ThermalCoeff);
figure(3); FEMtrimesh(Mesh,sigma_x); xlabel('x'); ylabel('y'); zlabel('\sigma_x');
           zlim([-30,50]); view([-125,25])

y = linspace(-D1,D2,1000)'; xy = [L/2*ones(size(y)),y];
[sigma_xc,sigma_yc,tau_xyc] = ...
EvaluateStress(Mesh,u1,u2,E,nu,'curve',xy,'thermal',ThermalCoeff);
figure(4); plot(y,sigma_xc); xlabel('y'); ylabel('\sigma_x')

x = linspace(0,L)'; u2_x = FEMgriddata(Mesh,u2,x,zeros(size(x)));
figure(5); plot(x,u2_x); xlabel('x'); ylabel('u_2(x,0)')
Param_u2 = polyfit(x,u2_x,2)
Curvature_FEM = 2*Param_u2(1)
--->
Max_u1u2 =   0.049138   0.418703
Param_u2 =   2.6150e-04  -1.4407e-09  -1.5918e-08
Curvature_FEM = 5.2301e-04
```

9.41 A copper plate stretched by thermal expansion

A copper plate has an inner disk with radius $R = 0.5$ mm which is prestressed by precooling the inner cylinder.

1. The copper plate has a circular hole with radius 0.5 mm and the copper cylinder has a radius slightly larger then 0.5 mm.
2. The cylinder is cooled down. If cooled by -6 C$^\circ$, the radius will be exactly 0.5 mm, but it is cooled down even more. Then it will fit nicely into the hole in the plate.
3. When warming up the expanding cylinder will press into the plate, leading to a tight fit.

The code `PlateThermalExpansion.m` will simulate the above final situation.

- The first few lines define the material parameters, the key temperature difference $\Delta T = 6$ and the dimensions of plate and cylinder.
- The mesh is generated with a rather fine mesh at radius $R = 0.5$. `CreateMeshTriangle()` is used with an optional `Segment`. The segment describes the circle at radius $R = 0.5$ mm. The outer boundary of the plate is free to move.
- To prevent constant displacements and rotations additional constraints are applied by calling the command `MeshAddConstraint()` twice. The origin can not move at all and the point $(R, 0)$ can not move in y–direction. The mesh is shown in Figure 223. Then the mesh is upgraded to cubic elements.
- The corresponding boundary value problem is solved by `PlaneStress()` with an option `thermal`. The displacements u_1 and u_2 are shown in Figure 224.
- With `EvaluateStrain()` the strains are evaluated at the nodes, leading to Figure 225. Along the circle $x^2 + y^2 = R^2 = 0.5^2$ the sudden jumps of the normal strains ε_{xx} and ε_{yy} are obvious.
- With `EvaluateStress()` the stresses are evaluated at the nodes, the resulting figures are not shown in the notes.
- The next step is to evaluate the radial stress along the circle with radius $R = 0.5$ mm. Since the stress tensor is a tensor of order 2 the stresses in a rotated coordinate system are determined by

$$\begin{bmatrix} \sigma_{x'} & \tau_{x'y'} \\ \tau_{x'y'} & \sigma_{y'} \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} .$$

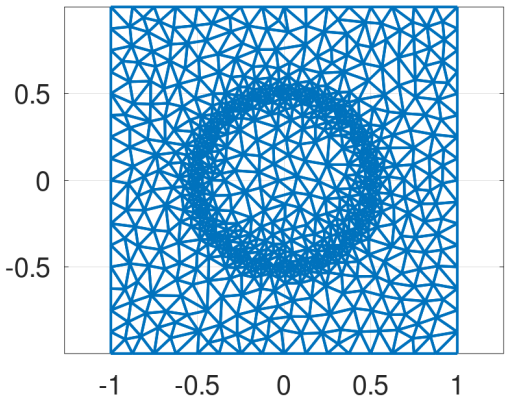


Figure 223: The mesh of the copper plate with the heated disk

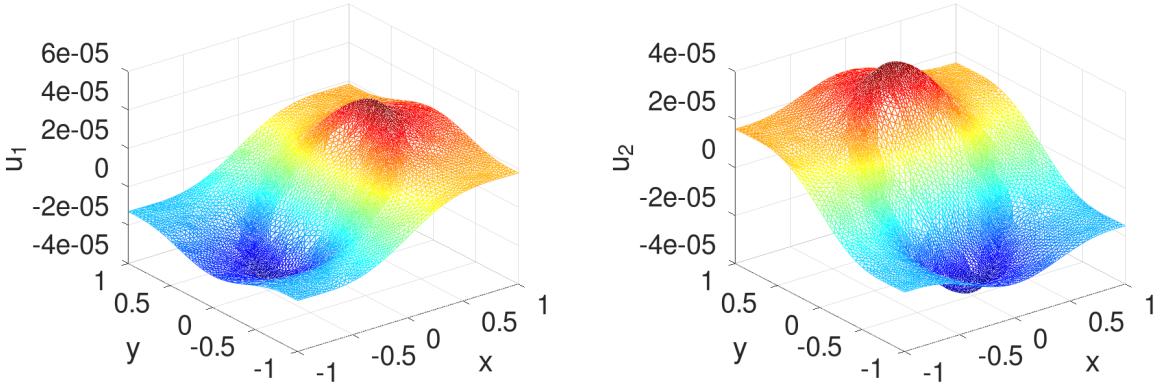


Figure 224: The displacements of the copper plate with the heated disk

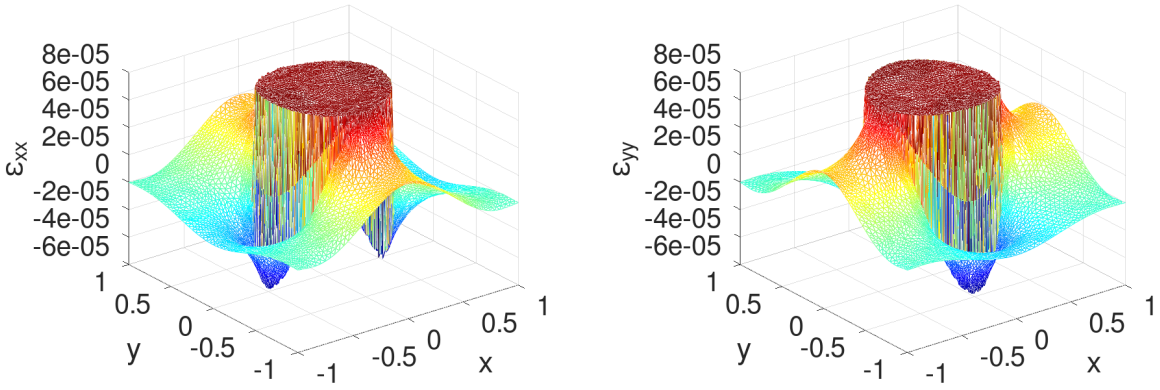


Figure 225: The normal strains of the copper plate with the heated disk

For the normal stress at an angle ϕ along the circle this leads to[64]

$$\sigma_n = \sigma_x \cos^2(\phi) + 2\,\tau_{xy} \cos(\phi) \sin(\phi) + \sigma_y \sin^2(\phi) \; .$$

The evaluation is made along a radius $1.001\,R$ slightly larger than R to assure that the points are strictly inside elements without thermal expansion. The command `EvaluateStress()` with an option `curve` is used. In Figure 226 find the two normal stresses σ_x and σ_y showing the expected periodic behavior. The normal stress σ_n along the circle is almost constant, as expected. The maximal compressive stress of σ_n coincides with the maximal compressive values of σ_x and σ_y.

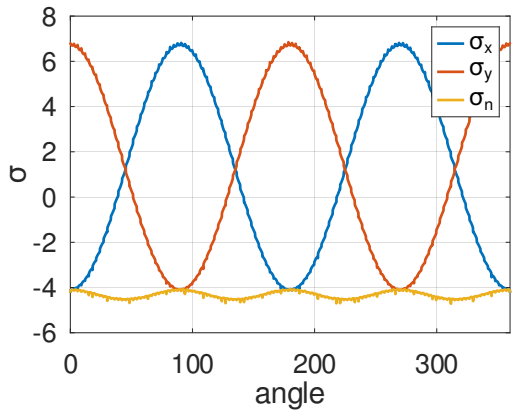


Figure 226: Normal stresses of the copper plate with the heated disk along the circle

PlateThermalExpansion.m

```
%% set the parameters
E = 110e3; nu = 0.35; alpha = 1.7e-5; DeltaT = 6; %% copper, length in mm
L = 1;  N = 11; R = L/2;
ThermalCoeff = @(xy)(DeltaT*alpha*( (xy(:,1).^2 + xy(:,2).^2) < R^2));

%% generate the mesh with high resolution along the circle
phi = linspace(0,2*pi,360/3)'; phi = phi(2:end);
Seg.name   = 'Segment';
Seg.border = [R*cos(phi),R*sin(phi),zeros(size(phi))];
BC = -22;
Mesh = CreateMeshTriangle('block',[-L,-L,BC;+L,-L,BC;+L,+L,BC;-L,+L,BC],(L/N)^2,Seg);
Mesh = MeshAddConstraint(Mesh,[0,0],[-1,-1]);
Mesh = MeshAddConstraint(Mesh,[R,0],[-2,-1]);
figure(1); FEMtrimesh(Mesh); axis equal
% Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

%% solve the problem
[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{0,0},{0,0},'thermal',ThermalCoeff);
figure(2); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(3); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')

%% display strains and stresses
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(Mesh,u1,u2);
figure(4); FEMtrimesh(Mesh,eps_xx);
           xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(5); FEMtrimesh(Mesh,eps_yy);
           xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')
```

64

$$\sigma_{x'} = \begin{bmatrix} \cos\phi & \sin\phi \end{bmatrix} \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \cdot \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} = \sigma_x \cos^2(\phi) + 2\,\tau_{xy} \cos(\phi) \sin(\phi) + \sigma_y \sin^2(\phi)$$

```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',ThermalCoeff);
figure(6); FEMtrimesh(Mesh,sigma_x);
           xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(7); FEMtrimesh(Mesh,sigma_y);
           xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(8); FEMtrimesh(Mesh,tau_xy);
           xlabel('x'); ylabel('y'); zlabel('\tau_{xy}')

%% evaluate along the circle
phi = linspace(0,2*pi,2001)'; phiD = phi/pi*180; Rc = R*1.001;
x = Rc*cos(phi); y = Rc*sin(phi);
[sigma_xc,sigma_yc,tau_xyc] ...\
    = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',ThermalCoeff,'curve',[x,y]);
sigma_nc = sigma_xc.*cos(phi).^2 + 2*tau_xyc.*cos(phi).*sin(phi)...
         + sigma_yc.*sin(phi).^2;
figure(9); plot(phiD,sigma_xc,phiD,sigma_yc,phiD,sigma_nc);
           xlabel('angle'); ylabel('\sigma'); xlim([0,360])
           legend('\sigma_x','\sigma_y','\sigma_n','location','northeast');
```

9.42 Deformation of a plate

The idea for this mechanical examples is taken from www.youtube.com/watch?v=AXL4ll3aRB8, a video by Michael Swanbom from Louisiana Tech University. In these notes the usual SI units are used, thus the numbers from the ones used by Michael Swanbom. The domain shown in Figure 227 consist of a steel plate with thickness 0.5 mm. The edge on the right is clamped and on the left edge a horizontal uniform load of 10 N is applied. A plane stress assumption is used. The main interest is in a possible stress concentration and the resulting maximal von Mises stress.

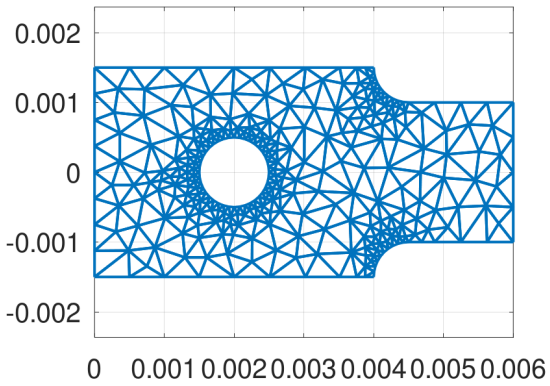


Figure 227: Domain with a hole in the plate of thickness 0.5 mm

9.42.1 Loading on the left boundary

- Construct the domain with the appropriate boundary conditions. Use the feature `Hole` of `CreateMeshTriangle()` to inculde the circular hole. Generate Figure 227.

SwanbomHole.m

```
%% based on www.youtube.com/watch?v=AXL4ll3aRB8
%% construct the domain
N = 10; al90 = linspace(0,pi/2,N+1)'; xr = 0.5*cos(al90); yr = 0.5*sin(al90);
al360 = linspace(0,2*pi,4*N+1)'; al360 = al360(1:end-1);
Hole.name = 'Hole'; Hole.point = 1e-3*[2,0]; Radius = 0.5e-3;
Hole.border=[2e-3+Radius*cos(al360),Radius*sin(al360),-22*ones(size(al360))];
x = 1e-3*[0;0;4.5-xr;6;6;4.5-flipud(xr)];
y = 1e-3*[-1.5;1.5;1.5-yr;1;-1;-1.5+flipud(yr)];
BC = [32;22;22*ones(size(al90));11;22;22*ones(size(al90))];
```

```
figure(1); plot(x,y,'-',Hole.border(:,1),Hole.border(:,2)); axis equal
Mesh = CreateMeshTriangle("Swanbom",[x,y,-BC],0.1e-6,Hole);
figure(2); FEMtrimesh(Mesh); axis equal
Mesh = MeshUpgrade(Mesh,'cubic');
```

- Define the material parameters for steel and use the command `PlaneStress()` to determine the displacements u_x and u_y shown in Figure 228.

SwanbomHole.m

```
Load = 10;
E = 200e9; nu = 0.25; gN = {-Load/(3*0.5)*1e6,0}; %% data for steel
  %% solve the plane stress problem
[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{0,0},gN);

figure(11); FEMtrimesh(Mesh,u1);
            xlabel('x'); ylabel('y'); zlabel('u_x'); view([-30,30])
figure(12); FEMtrimesh(Mesh,u2);
            xlabel('x'); ylabel('y'); zlabel('u_y'); view([-130,45])
```

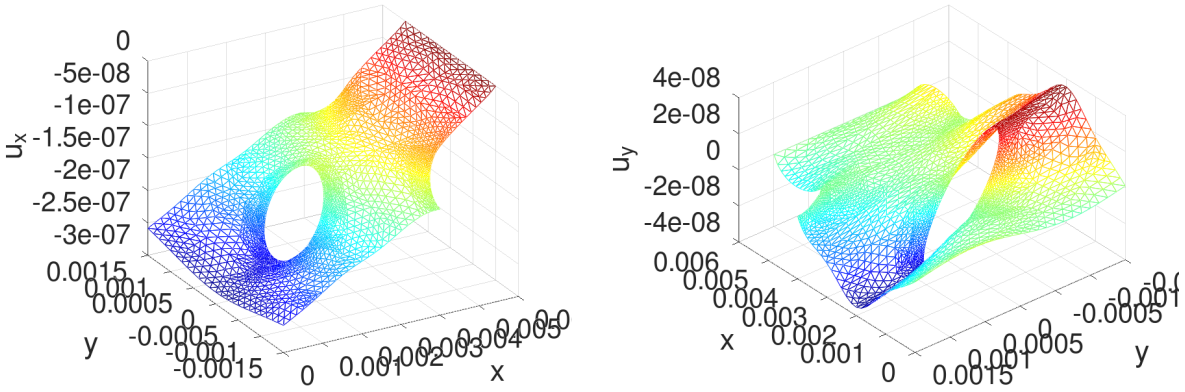


Figure 228: Horizontal and vertical displacement on the plate with the hole

- With the help of `EvaluateStrain()` determine the resulting strains, not shown in these notes. Then use the commands `EvaluateStress()` and `EvaluateVonMises()` to find the von Mises stress at the nodes. With `FEMtrimesh()` and `FEMtricontour()` generate Figure 229. The maximal von Mises stress of ≈ 25 MPa is visible at the upper and lower edge of the hole. The second stress concentration of ≈ 16 MPa at the rounded sections is not as flagrant. Use the zoom and rotate feature of the graphics, or a similar computation without the hole to examine the details.

SwanbomHole.m

```
[eps_xx,eps_yy,eps_xy] = EvaluateStrain(Mesh,u1,u2);
figure(13); FEMtrimesh(Mesh,eps_xx);
            xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(14); FEMtrimesh(Mesh,eps_yy);
            xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')

[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu);
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy)*1e-6;
figure(21); FEMtrisurf(Mesh,vonMises);
            xlabel('x'); ylabel('y'); zlabel('von Mises [MPa]')
figure(22); FEMtricontour(Mesh,vonMises,51);
            xlabel('x'); ylabel('y'); title('von Mises [MPa]')
            colorbar(); axis equal
            hold on; plot([x;x(1)],[y;y(1)],'k'); hold off
```

```
figure(23); FEMtrimesh(Mesh,sigma_x*1e-6);
            xlabel('x'); ylabel('y'); zlabel('\sigma_x [MPa]')
figure(24); FEMtrimesh(Mesh,sigma_y*1e-6);
            xlabel('x'); ylabel('y'); zlabel('\sigma_y [MPa]')

Max_vonMises = max(vonMises)
-->
Max_vonMises = 25.147
```

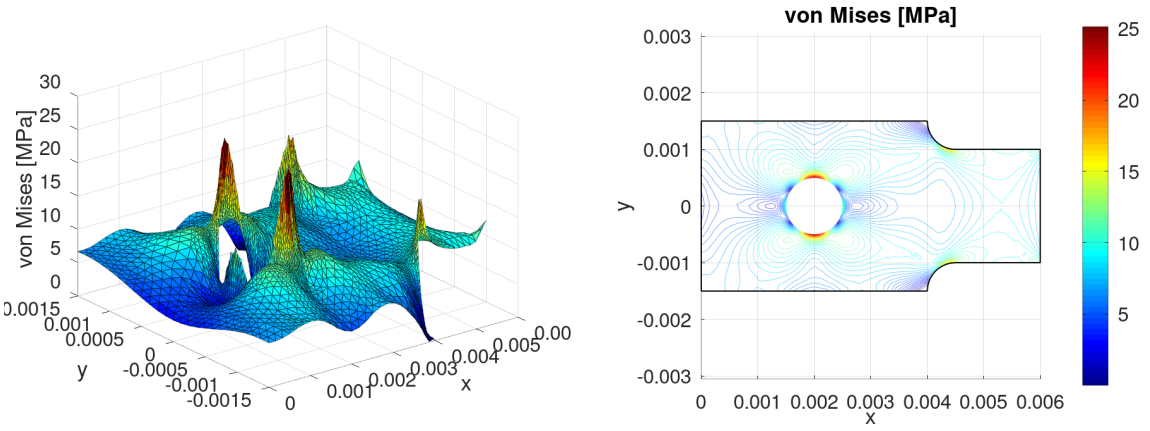


Figure 229: The von Mises stress on the plate with the hole

9.42.2 Loading inside of the hole

Instead of the loading along the left edge by 10 N the same load can be applied by a material filling the hole. This is implemented with the help of a surface loading $f_x(x,y)$ (units $\frac{\mathrm{N}}{\mathrm{m}^2}$) inside the hole. The code `SwanbomLoaded.m` shown below defines the surface loading by a function

$$f_x(x,y) = \begin{cases} \frac{-10}{\pi\, 0.0005^3} & \text{for } (x-0.002)^2 + y^2 < 0.0005^2 \\ 0 & \text{for } (x-0.002)^2 + y^2 \geq 0.0005^2 \end{cases} .$$

To examine different materials for the plate and the material filling the hole Young's modulud E is defined by

$$\mathrm{E}(x,y) = \begin{cases} 200 \cdot 10^9 & \text{for } (x-0.002)^2 + y^2 < 0.0005^2 \\ 150 \cdot 10^9 & \text{for } (x-0.002)^2 + y^2 \geq 0.0005^2 \end{cases} .$$

These values for the hole are close to copper. Find the results for the von Mises stress in Figure 230. Observe that the von Mises stress at the outside rounded edges is very similar to the above situation. This should not be a surprise, sine the total horizontal load of 10 N is the same. Inside the copper filled hole there is significant von Mises stress.

SwanbomLoaded.m

```
N = 20; R = 0.5e-3; H = 0.5e-3;
al90 = linspace(0,pi/2,N+1)'; xr = 0.5*cos(al90); yr = 0.5*sin(al90);
al360 = linspace(0,2*pi,6*N+1)'; al360 = al360(1:end-1);
Segment.name = 'Segment'; Radius = 0.5;
Segment.border = 1e-3*[2+Radius*cos(al360), Radius*sin(al360), zeros(size(al360))];
x = 1e-3*[0;0;4.5-xr;6;6;4.5-flipud(xr)];
y = 1e-3*[-1.5;1.5;1.5-yr;1;-1;-1.5+flipud(yr)];
BC = -[32;22;22*ones(size(al90));11;22;22*ones(size(al90))];
figure(1); plot(x,y,'-',Segment.border(:,1),Segment.border(:,2))

Mesh = CreateMeshTriangle("Swanbom",[x,y,BC],0.1e-6,Segment);
figure(2); FEMtrimesh(Mesh); axis equal
```

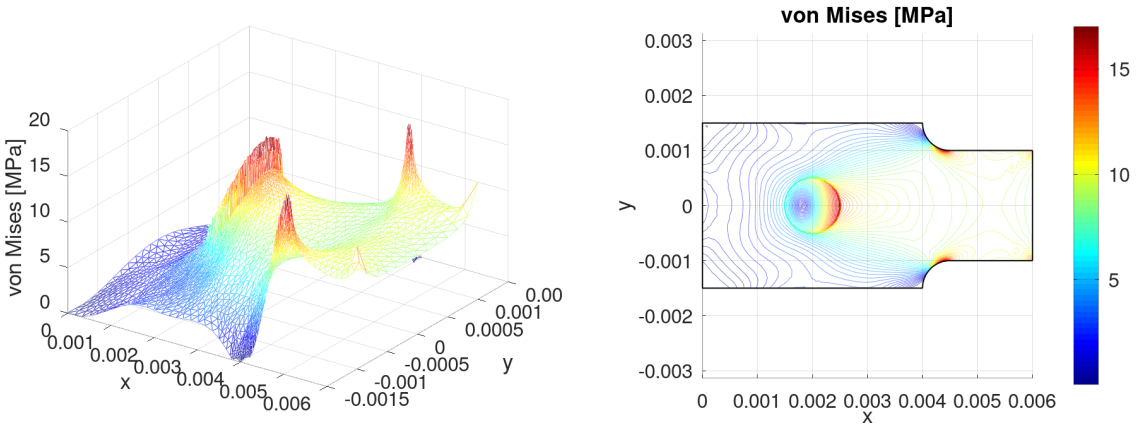


Figure 230: The von Mises stress on the plate with the loading inside of the hole

```
%%Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

Load = -10;  BoundaryLoad = 0*Load/(3e-3*H);  DiscLoad = 1*Load/(pi*R^2*H);
DeltaT = 0;  %% use 0 for no thermal prestress or 10 for thermal prestress
E_func = @(xy)E*(1 - 0.25*(((xy(:,1)-2e-3).^2+xy(:,2).^2)<R^2));
fx_func     = @(xy)(DiscLoad*(((xy(:,1)-2e-3).^2+xy(:,2).^2)<R^2));
alphaDeltaT = @(xy)(alpha*DeltaT*(((xy(:,1)-2e-3).^2+xy(:,2).^2)<R^2));

%% solve the plane stress problem
[u1,u2] = PlaneStress(Mesh,E_func,nu,{fx_func,0},{0,0},{BoundaryLoad,0},'thermal',alphaDeltaT);
figure(11); FEMtrimesh(Mesh,u1); xlabel('x'); ylabel('y'); zlabel('u_x'); view([-30,30])
figure(12); FEMtrimesh(Mesh,u2); xlabel('x'); ylabel('y'); zlabel('u_y'); view([-130,45])

[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',alphaDeltaT);
[eps_xx,eps_yy,eps_xy]   = EvaluateStrain(Mesh,u1,u2);
figure(13); FEMtrimesh(Mesh,eps_xx); xlabel('x'); ylabel('y'); zlabel('\epsilon_{xx}')
figure(14); FEMtrimesh(Mesh,eps_yy); xlabel('x'); ylabel('y'); zlabel('\epsilon_{yy}')

vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy)*1e-6;
figure(21); FEMtrimesh(Mesh,vonMises);
            xlabel('x'); ylabel('y'); zlabel('von Mises [MPa]'); view([35,35])
figure(22); FEMtricontour(Mesh,vonMises,51);
            xlabel('x'); ylabel('y'); title('von Mises [MPa]')
            colorbar(); axis equal
            hold on; plot([x;x(1)],[y;y(1)],'k'); hold off

figure(23); FEMtrimesh(Mesh,sigma_x*1e-6);
            xlabel('x'); ylabel('y'); zlabel('\sigma_x [MPa]'); view([35,35])
figure(25); FEMtricontour(Mesh,sigma_x*1e-6,51);
            xlabel('x'); ylabel('y'); title('\sigma_x [MPa]'); axis equal;
            colorbar(); hold on; plot([x;x(1)],[y;y(1)],'k'); hold off
figure(24); FEMtrimesh(Mesh,sigma_y*1e-6);
            xlabel('x'); ylabel('y'); zlabel('\sigma_y [MPa]')
Max_vonMises = max(vonMises)
```

In Figure 231 find the results for the normal stress σ_x in x–direction. Observe the behavior inside the hole:

- in the left part of the hole the material is compressed, $\sigma_x < 0$.
- in the right part of the hole the material is stretched, $\sigma_x > 0$.

If the copper and steal are not clued together the materials will separate. This is confimed by Figure 232(a). The normal stress σ_x is evaluated just outside of the (former) hole.

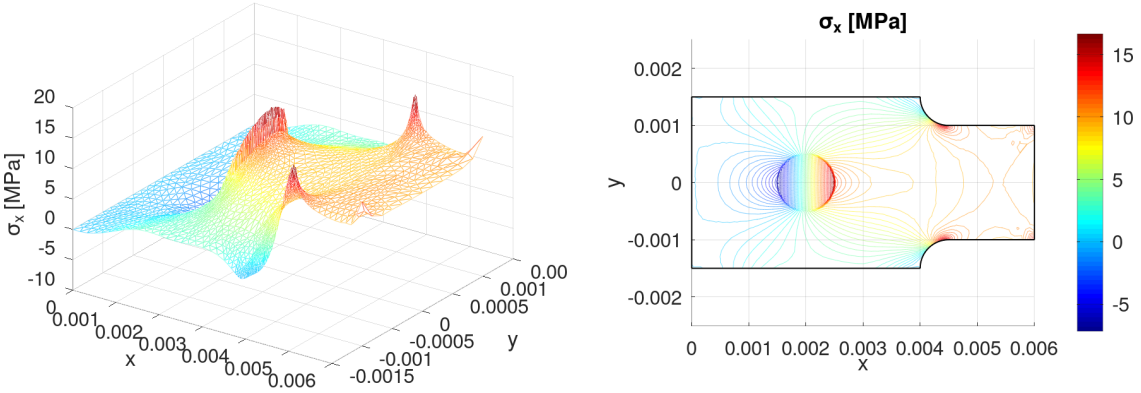
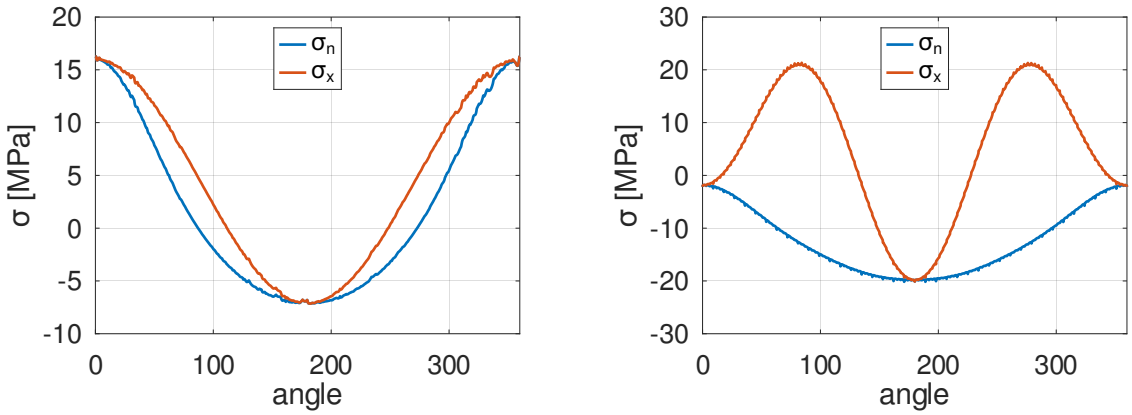


Figure 231: The normal stress σ_x on the plate with the loading inside of the hole

To decide where the two matreial will separte examine the stress normal to the boundary of the hole. The stress in a direction $\vec{n} = (\cos(\alpha), \sin(\alpha))^T$ is given by

$$\sigma_{\vec{n}} = \langle \vec{n}\,, \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \vec{n} \rangle = \langle \begin{pmatrix} n_x \\ n_y \end{pmatrix} \begin{pmatrix} \sigma_x\, n_x + \tau_{xy}\, n_y \\ \tau_{xy}\, n_x + \sigma_y\, n_y \end{pmatrix} \rangle \;=\; \sigma_x\, n_x^2 + 2\,\tau_{xy}\, n_x\, n_y + \sigma_y\, n_y^2\,.$$

This can be evaluated with `FEMoctave`, leading to Figure 232(a). The normal stress $\sigma_{\vec{n}}$ is negative for angles $0° < \alpha < 87°$ and $273° < \alpha < 360°$, approximately. In this area the two material might loose contact. Observe that the approach in `SwanbomLoaded.m` does not solve the contact problem, i.e. the computations assume that the two material remain in contact. For a sample of contact problems examine the computation in sections 9.52 and 9.53 on Hertz contact problems.



(a) no prestress

(b) with thermal prestress

Figure 232: The horizontal stress σ_x and the normal stress σ_n along the inner edge of the hole

SwanbomLoaded.m

```
phi = linspace(0,2*pi,2001)'; Rc = 0.5001e-3; phiD = phi/pi*180;
x = 0.002+Rc*cos(phi); y = Rc*sin(phi);
[sigma_xc,sigma_yc,tau_xyc] = EvaluateStress(Mesh,u1,u2,E,nu,'thermal',alphaDeltaT,'curve',[x,y]);
sigma_n = sigma_xc.*cos(phi).^2 + 2*tau_xyc.*cos(phi).*sin(phi) + sigma_yc.*sin(phi).^2;
figure(31); plot(phiD,sigma_n*1e-6,phiD,sigma_xc*1e-6);
```

```
xlabel('angle'); ylabel('\sigma [MPa]')
legend('\sigma_n','\sigma_x','location','north'); xlim([0,360])
```

9.42.3 Loading inside of the hole, with a thermal prestress

In the above code `SwanbomLoaded.m` set the prestress temperature to `DeltaT = 10;` to model the tighter fit of the inner cylinder. The process of thermal prestress is explaine in Example 9.41 on page 370. The final result is visible in Figure 232(b). The normal stress $\sigma_{\vec{n}}$ is now strictly negative for all angles, i.e. the two materials are hold together by a compressive stress $\sigma_{\vec{n}} < 0$.

9.43 A pipe under pressure

Examine a pipe with a circular cross section and inner radius R and a wall with thickness ΔR. On the inside a pressure P is applied. The pipe under pressure will expand and the wall material will stretch. For ductile materials (e.g. copper, steel) the maximal value of the von Mises stress is a good criterion to decide whether the pipe will withstand the pressure, or break. The problem can be examined by FEM as a plane strain problem or as an axially symmetric problem, or one can determine an exact solution. For a setup with a very thin wall there is an analytical solution.

As exemplary situation examine:

- a pipe with inner radius $R = 0.1$ m and a wall thickness of $\Delta R = 0.01$ m. A quarter of a cross section is visible in Figure 233.
- a pressure of $P = 10$ atm $= 10^6$ Pa $= 10^6 \frac{\mathrm{N}}{\mathrm{m}^2}$.
- with a copper pipe, i.e. a yield strength of ≈ 33 MPa or a steel pipe, i.e. a yield strength of ≈ 350 MPa.

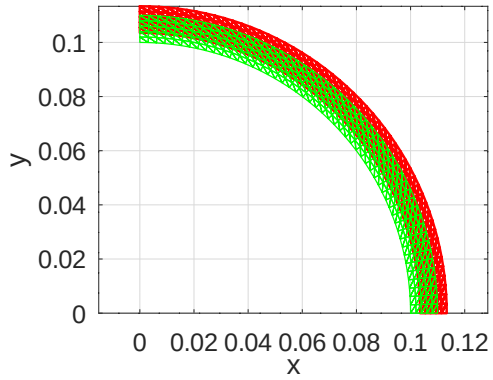


Figure 233: One quarter of a section through the pipe, the original domain (green) and the deformed domain (red)

9.43.1 As a plane strain problem

To examine this problem with FEMoctave start by defining the domain and the boundary conditions.

- Define the domain with the help of polar coordinates $R \le r \le R+\Delta R$ and $0 \le \phi \le \frac{\pi}{2}$. Use `CreateMeshRect()` to create a rectangular mesh and then `MeshDeform()` to create the domain in Figure 233.
- At the lower edge at $y = 0$ the edge is free to move in x–direction and no displacement in y–direction. This is implemented with the code -21 in the function `CreateMeshRect()` for the boundary condition. See Table 6 on page 62 for the coding of the boundary conditions.
- At the left edge at $x = 0$ the edge is free to move in y–direction and no displacement in x–direction. This is implemented with the code -12 for the boundary condition.
- At the inner edge at $r = R$ pressure P is applied, leading to the code -33 for the boundary condition.

- At the outer edge at $r = R + \Delta R$ there is no force, leading to the code -22 for the boundary condition.
- For good accuracy second order elements are used by calling `MeshUpgrade()`.

PipePressure.m

```
E = 110e9; nu = 0.35; %%% copper
%%E = 200e9; nu = 0.25; %%% steel
R = 0.1; dR = 0.01;
nR = 5; nPhi = 51;  %% number of layers in radial and angular direction
Estar = E/(1-nu^2);  nustar = nu/(1-nu);

global P
P = 10e5;  %% 10 atm pressure
FEMmesh = CreateMeshRect(linspace(R,R+dR,nR+1),linspace(0,pi/2,nPhi+1),-21,-12,-33,-22);
function new_xy = Deform(xy)
  new_xy = [xy(:,1).*cos(xy(:,2)),xy(:,1).*sin(xy(:,2))];
endfunction
FEMmesh = MeshDeform(FEMmesh,'Deform');
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
```

With this domain and the correct boundary conditions the problem can be solved.

- Start by defining the force density corresponding to the inside pressure P, i.e.

$$\text{at} \begin{pmatrix} R\cos\alpha \\ R\sin\alpha \end{pmatrix} \quad \text{apply the force density} \quad \begin{pmatrix} P\cos\alpha \\ P\sin\alpha \end{pmatrix}.$$

- Assuming that the pipe will not stretch in the direction orthogonal to the cross section we end up with a plane strain problem. Thus use `PlaneStrain()` to find approximations to the displacements u_1 and u_2.

PipePressure.m

```
%% define the radial pressure to be applied on the inside
function res = gN1(xy)
  global P
  angle = atan2(xy(:,2),xy(:,1));  res   = P*cos(angle);
endfunction
function res = gN2(xy)
  global P
  angle = atan2(xy(:,2),xy(:,1));  res   = P*sin(angle);
endfunction

[u1,u2] = PlaneStrain(FEMmesh,E,nu,{0,0},{0,0},{'gN1','gN2'});

factor = 400;
figure(111); ShowDeformation(FEMmesh,u1,u2,factor); axis equal; xlabel('x'); ylabel('y');
```

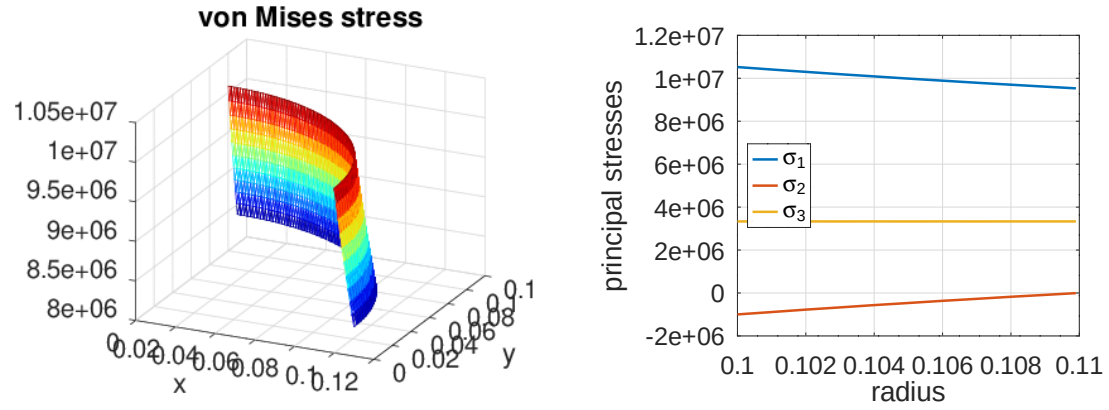
The last few lines in the above code generate the domain visible in Figure 233.

With the displacements determine all stresses at the nodes by using `EvaluateStress()`. Since four return arguments are asked for the plane strain setup is used. Then use `EvaluateVonMises()` to find the values of the von Mises stress, visible in Figure 234(a). The maximal value of the von Mises stress is approximated by 10 MPa, which is smaller than the yield strength 33 MPa of copper. Thus the pipe is expected to withstand the applied pressure, but the margin of error is not very large. The pipe will start cracking on the inside, where the von Mises stress is largest.

PipePressure.m

```
[sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(FEMmesh,u1,u2,E,nu);
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy,sigma_z);

figure(2); FEMtrimesh(FEMmesh,vonMises); xlabel('x'); ylabel('y');
           title('von Mises stress'); view([25,25])
vonMises_min_max = [min(vonMises),max(vonMises)]
-->
vonMises_min_max =  8.3695e+06   1.0082e+07
```



(a) the von Mises stress

(b) the principal stresses

Figure 234: The von Mises stress in the cross section and the principal stresses along a radius of the pipe under pressure

To analyze the pipe further choose an angle, e.g. $\alpha = \frac{\pi}{4} = 45°$, and evaluate along a straight line with this angle for radii $R \le r \le R + \Delta R$.

- Start by selecting the angle $0 \le \alpha \le \frac{\pi}{2}$ and define the x and y values along the arc with this angle.
- Use the above values of the stresses and `EvaluatePrincipalStress()` find the values of the principle stresses at the nodes.
- Then calls of `FEMgriddata()` will determine the values of the principle stresses along the selected arc. Use $\sigma_3 = \nu\,(\sigma_1 + \sigma_2)$ to compute the third principle stress. Then a call of `plot()` will generate Figure 234(b).
- The minimal value $-9.9770 \cdot 10^5 \approx -1$ MPa of σ_2 shows that this is the normal stress in radial direction on the inside of the pipe, coinciding with the given pressure P.
- The maximal value $-7 \cdot 10^0 \approx 0$ MPa of σ_2 corresponds to the zero pressure on the outside.
- The values of σ_1 are considerably larger than the values of σ_2. This illustrates that the wall of the pipe is severely stretched in angular direction.

PipePressure.m

```
%% evaluation at one angle, all radii
alpha = pi/4; Nr = 101; Nmid = (Nr+1)/2; %% use an odd number for Nr
r = linspace(R,R+dR,Nr)'; x = r*cos(alpha); y = r*sin(alpha);

[sigma_1,sigma_2] = EvaluatePrincipalStress(sigma_x,sigma_y,tau_xy);
sigma_1r = FEMgriddata(FEMmesh,sigma_1,x,y);
sigma_2r = FEMgriddata(FEMmesh,sigma_2,x,y);
sigma_3r = nu*(sigma_1r+sigma_2r);
sigma_2r_min_max = [min(sigma_2r),max(sigma_2r)]

figure(3); plot(r,[sigma_1r,sigma_2r,sigma_3r]);
               xlabel('radius'); ylabel('principal stresses')
               legend('\sigma_1','\sigma_2','\sigma_3','location','west')
-->
sigma_2r_min_max = -9.9770e+05  -7.1114e+03
```

At the midpoint in the wall of the pipe the stress matrix (tensor, to be precise) can be evaluated with the help of three calls of `FEMgriddata()`.

$$\begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \approx \begin{bmatrix} +4.7572 & -5.2252 \\ -5.2252 & +4.7616 \end{bmatrix} \cdot 10^6$$

Then use a rotation matrix and the transformation rule for second order tensors to determine the stresses in the rotated coordinate system.

$$\begin{bmatrix} +\cos\alpha & +\sin\alpha \\ -\sin\alpha & +\cos\alpha \end{bmatrix} \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{bmatrix} +\cos\alpha & -\sin\alpha \\ +\sin\alpha & +\cos\alpha \end{bmatrix} \approx \begin{bmatrix} -0.46582 & +0.00217 \\ +0.00217 & +9.9846 \end{bmatrix} \cdot 10^6$$

The result shows the normal, compressing pressure of -0.47 MPa in radial direction and the stretching pressure of $+10$ MPa in angular direction.

PipePressure.m

```
%% examine stress at middle point
x_mid = x(Nmid); y_mid = y(Nmid);

sigma_x = FEMgriddata(FEMmesh,sigma_x,x_mid,y_mid);
sigma_y = FEMgriddata(FEMmesh,sigma_y,x_mid,y_mid);
tau_xy  = FEMgriddata(FEMmesh,tau_xy ,x_mid,y_mid);
RotMat = [cos(alpha) -sin(alpha);+sin(alpha) cos(alpha)];
stress   = [sigma_x,tau_xy;tau_xy,sigma_y]
stress_rotated   = RotMat'*stress*RotMat
-->
stress         =   4.7572e+06  -5.2252e+06
                  -5.2252e+06   4.7616e+06

stress_rotated =  -4.6582e+05   2.1668e+03
                   2.1668e+03   9.9846e+06
```

With the provided code in `PipePressure.m` it is easy the modify the parameters of the above problem, e.g. change from copper to steel, examine larger radii or thinner walls.

For a pipe with a thin wall an analytical approximation is possible. Examine the section shown in Figure 233 and assume that the normal stress σ_φ in angular direction is independent on the radius. Then use a balance of force law in y–direction and an integration over the angle to conclude

$$\sigma_\varphi \, \Delta R = \int_0^{\pi/2} P \, \sin\varphi \, R \, d\varphi = P \, R \, .$$

In the above example this leads to

$$\sigma_\varphi = \frac{P \, R}{\Delta R} = \frac{10^6 \cdot 0.1}{0.01} = 10^7 \, ,$$

which is very close to the above result generated by FEMoctave. With the known angular stress and Hooke's law estimate the angular stretch, i.e.

$$\varepsilon_\varphi = \frac{\sigma_\varphi}{E} = \frac{10^7}{110 \cdot 10^9} \approx 9.09 \cdot 10^{-5} \, .$$

Since the angular stretching factor ε_φ equals the radial stretching factor ε_r estimate the change of radius by

$$R \quad \longrightarrow \quad R\,(1+\varepsilon_r) = R + 9.09 \cdot 10^{-6} \, .$$

This is not too far from the FEMoctave result of $\max\{u_1\} \approx 8.8 \cdot 10^{-6}$. The FEM approximation allows to examine pipes with thick walls and also examines behavior within the wall.

9.43.2 As an axisymmetric problem

The above problem can be examined as an axially symmetric problem. The domain is given by $R \le r \le R + \Delta R$ and $0 \le z \le R$, and then rotated about the z–axis.

- At the inner edge at $x = r = R$ the pressure P is applied in r–direction and the edge is free to move in z direction.
- The outer edge at $x = r = R + \Delta R$ is free to move.
- The lower and upper edge are fixed in z–direction and free to move in $x = r$–direction.

Start out be defining the parameters and generating the mesh. Then determine the radial displacement u_r and the z–displacement u_z by calling `AxiStress()`.

PipePressureAxi.m

```
R = 0.1; dR = 0.01;
if 0 %% regular mesh
  Mesh = CreateMeshRect(R+linspace(0,dR,20),linspace(0,R,10),-21,-21,-32,-22);
else  %% irregular mesh
  Mesh = CreateMeshTriangle('Test',...
         [R 0 -21;R+dR 0 -22; R+dR R -21; R R -32],1e-5);
endif
Mesh = MeshUpgrade(Mesh,'quadratic');

P = 10e5;  E = 110e9; nu = 0.35; f = {0,0}; gD = {0,0}; gN = {P,0};
[ur,uz] = AxiStress(Mesh,E,nu,f,gD,gN);
figure(2); FEMtrimesh(Mesh,ur);
           xlabel('r'); ylabel('z'); zlabel('u_r')
figure(3); FEMtrimesh(Mesh,uz);
           xlabel('r'); ylabel('z'); zlabel('u_z')
```

Determine all strains by using `EvaluateStrainAxi()`.

PipePressureAxi.m

```
[eps_xx,eps_yy,eps_zz,eps_xz] = EvaluateStrainAxi(Mesh,ur,uz);
figure(11); FEMtrimesh(Mesh,eps_xx)
            xlabel('r'); ylabel('z'); zlabel('\epsilon_{xx}')
figure(12); FEMtrimesh(Mesh,eps_yy)
            xlabel('r'); ylabel('z'); zlabel('\epsilon_{yy}')
figure(13); FEMtrimesh(Mesh,eps_zz)
            xlabel('r'); ylabel('z'); zlabel('\epsilon_{zz}')
```

Determine the normal and shearing stresses by using `EvaluateStressAxi()`.

PipePressureAxi.m

```
[sigma_x,sigma_y,sigma_z,tau_xz] = EvaluateStressAxi(Mesh,ur,uz,E,nu);
figure(21); FEMtrimesh(Mesh,sigma_x)
            xlabel('r'); ylabel('z'); zlabel('\sigma_x')
figure(22); FEMtrimesh(Mesh,sigma_y)
            xlabel('r'); ylabel('z'); zlabel('\sigma_y')
figure(23); FEMtrimesh(Mesh,sigma_z)
            xlabel('r'); ylabel('z'); zlabel('\sigma_z')
```

Determine the von Mises stress, the principal stresses and the Tresca stress by using the functions of `FEMoctave`: `EvaluateVonMisesAxi()`, `EvaluatePrincipalStressAxi()` and `EvaluateTrescaAxi()`. The results coincide with the values from the plane strain approach in the previous section.

PipePressureAxi.m

```
vonMises = EvaluateVonMisesAxi(sigma_x,sigma_y,sigma_z,tau_xz);
figure(24); FEMtrimesh(Mesh,vonMises)
            xlabel('r'); ylabel('z'); zlabel('von Mises')
[sigma_1,sigma_2] = EvaluatePrincipalStressAxi(sigma_x,sigma_z,tau_xz);
r = R + linspace(0,dR,100)';
sigma_1i = FEMgriddata(Mesh,sigma_1,r,R/2*ones(size(r)));
sigma_2i = FEMgriddata(Mesh,sigma_2,r,R/2*ones(size(r)));
sigma_3i = FEMgriddata(Mesh,sigma_y,r,R/2*ones(size(r)));
figure(25); plot(r,sigma_1i,r,sigma_2i,r,sigma_3i); xlabel('r'); ylabel('z');
            legend('\sigma_1','\sigma_2','\sigma_3', 'location','west')
Tresca = EvaluateTrescaAxi(sigma_x,sigma_y,sigma_z,tau_xz);
figure(26); FEMtrimesh(Mesh,Tresca); xlabel('r'); ylabel('z'); zlabel('Tresca')
```

9.43.3 The analytical solution

For this axisymmetric setup use that $u_z = 0$ and $u_r(r,z) = u_r(r)$ to determine an exact solution. The energy of the system is given by

$$\frac{U(u_r)}{2\,\pi} = \iint\limits_{\Omega} \frac{r\,E}{2\,(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,((\frac{\partial\,u_r}{\partial r})^2 + \frac{1}{r^2}\,u_r^2) + \frac{2\,\nu}{r}\,u_r\,\frac{\partial\,u_r}{\partial r} \right) dA - R\,P\,u_r(R)\;.$$

With the constant $k = \frac{E}{(1+\nu)\,(1-2\,\nu)}$ and the notation $u(r) = u_r(r)$ the expression to be minimized is

$$\begin{aligned}
U_r(u) &= \int_R^{R+\Delta R} \frac{r\,k}{2}\left((1-\nu)\,((u'(r))^2 + \frac{1}{r^2}\,u^2(r)) + \frac{2\,\nu}{r}\,u(r)\,u'(r)\right)\,dr - R\,P\,u(R) \\
&= \int_R^{R+\Delta R} \frac{k}{2}\left((1-\nu)\,(r\,(u'(r))^2 + \frac{1}{r}\,u^2(r))\right)\,dr + \frac{k\,\nu}{2}\,u^2(r)\Big|_{r=R}^{R+\Delta R} - R\,P\,u(R) \\
U_r(u+\phi) &= U_r(u) + \int_R^{R+\Delta R} k\,(1-\nu)\,(r\,u'\phi' + \frac{1}{r}\,u\,\phi)\,dr + k\,\nu\,u(r)\,\phi(r)\Big|_{r=R}^{R+\Delta R} - R\,P\,\phi(R) + O(\phi^2) \\
&= U_r(u) + \int_R^{R+\Delta R} k\,(1-\nu)\,(-(r\,u')' + \frac{1}{r}\,u)\,\phi\,dr + \\
&\quad + k\,((1-\nu)\,r\,u'(r)\,\phi(r) + \nu\,u(r)\,\phi(r))\Big|_{r=R}^{R+\Delta R} - R\,P\,\phi(R) + O(\phi^2)\,.
\end{aligned}$$

Use the Euler–Lagrange equation for this problem and determine the exact solution.

$$\begin{aligned}
0 &= -r\,(r\,(u'(r))' + u(r) \\
\text{Ansatz:}\quad & u(r) = r^\alpha \\
0 &= -r\,(r\,\alpha\,r^{\alpha-1})' + r^\alpha \;=\; -\alpha^2\,r^\alpha + r^\alpha \\
0 &= -\alpha^2 + 1 \quad\Longrightarrow\quad \alpha = \pm 1 \\
u(r) &= c_1\,r + c_2\,\frac{1}{r}
\end{aligned}$$

The two natural boundary conditions are

$$\begin{aligned}
(1-\nu)\,R\,u'(R) + \nu\,u(R) &= -\frac{R}{k}\,P \\
(1-\nu)\,(R+\Delta R)\,u'(R+\Delta R) + \nu\,u(R+\Delta R) &= 0\,.
\end{aligned}$$

Using the above solution $u(r) = c_1\,r + c_2\,\frac{1}{r}$ leads to

$$\begin{aligned}
R\,(1-\nu)\,(c_1 - \frac{1}{R^2}\,c_2) + \nu\,(c_1\,R + c_2\,\frac{1}{R}) &= -\frac{R}{k}\,P \\
(R+\Delta R)\,(1-\nu)\,(c_1 - \frac{1}{(R+\Delta R)^2}\,c_2) + \nu\,(c_1\,(R+\Delta R) + c_2\,\frac{1}{R+\Delta R}) &= 0
\end{aligned}$$

or as a system of linear equations

$$\begin{bmatrix} R & -\frac{1-2\,\nu}{R} \\ (R+\Delta R) & -\frac{1-2\,\nu}{R+\Delta R} \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} -\frac{R}{k}\,P \\ 0 \end{pmatrix}\,.$$

Using the above parameters the solutions are $c_1 \approx 1.7532 \cdot 10^{-5}$ and $c_2 \approx 7.0714 \cdot 10^{-7}$ and thus

$$u_r(r) = u(r) = c_1\,r + c_2\,\frac{1}{r} \approx 1.7532 \cdot 10^{-5}\,r + 7.0714 \cdot 10^{-7}\,\frac{1}{r}\,.$$

Using the results in Section 4.19.2 (page 106) all stresses and strains can be computed. The values coincide with the above FEM solutions.

```
k = E/((1+nu)*(1-2*nu));
c = [R -(1-2*nu)/R; (R+dR) -(1-2*nu)/(R+dR)] \ [-R/k*P; 0];
r = R + linspace(0,dR,100)';
u = c(1)*r + c(2)./r;
```

9.44 A sphere under hydrostatic pressure

A solid sphere with radius R is submitted to a hydrostatic pressure P. This will lead to uniform strains $\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{zz} = -\frac{1-2\,\nu}{E}\,P$ and no shearing strains. This leads to a constant energy density $w = \frac{3\,(1-2\,\nu)}{2\,E}\,P^2$. The numbers generated by the code `SphereHydrostatic.m` confirm these results.

- The material parameters $E = 1$ and $\nu = 0.25$ are rather theoretical.
- A quarter of a circle in the xz–plane with radius R is rotated about the z–axis to model the upper half of the sphere.
- Along the lower edge require $u_z = 0$ and no force in x–direction. Along the left edge require $u_x = u_r = 0$ and no force in z–direction. Along the curved section at an angle α the force density is given by

$$\vec{g}_N(\alpha) = -\begin{pmatrix} P\,\cos(\alpha) \\ P\,\sin(\alpha) \end{pmatrix}$$

  and implemented by two functions `gNr()` and `gNz()`.
- A mesh is generated by `CreateMeshTriangle()` and then the displacements u_r and u_z determined by calling `AxiStress()`. The figures generated by the code show these displacements.
- With a call of `EvaluateStrainAxi()` the strains are evaluated at the nodes, followed by a call of `EvaluateEnergyDensi` to find the energy density W.
- The total energy is the determined by the integral

$$\text{Energy} = \iint_{\Omega} 2\,\pi\,r\,W(r,z)\;dA$$

  and evaluated by calling `FEMIntegrate()`. Dividing by the volume $\frac{2}{3}\,R^3$ of the half sphere leads to the average energy density.
- The figures for the strains and the energy density are rather boring, since all these expressions are constant for the hydrostatic loading situation.

SphereHydrostatic.m

```
global P
P = -0.1;
E = 1; nu = 0.25; R = 1;
N = 51; alpha = linspace(0,+pi/2,N)'; x = R*cos(alpha); z = R*sin(alpha);
Dom = [0,0,-21;[x,z,-33*ones(size(x))]]; Dom(end,3) = -12;
function res = gNr(rz)
  global P
  alpha = atan2(rz(:,2),rz(:,1));
  res   = P*cos(alpha);
endfunction
function res = gNz(rz)
  global P
  alpha = atan2(rz(:,2),rz(:,1));
  res   = P*sin(alpha);
endfunction

Mesh = CreateMeshTriangle('quart',Dom,1e-2);
Mesh = MeshUpgrade(Mesh,'quadratic');
[ur,uz] = AxiStress(Mesh,E,nu,{0,0},{0,0},{'gNr','gNz'});
figure(1); FEMtrimesh(Mesh,ur); xlabel('r'); ylabel('z'); zlabel('u_r'); view([20,40])
figure(2); FEMtrimesh(Mesh,uz); xlabel('r'); ylabel('z'); zlabel('u_z'); view([-120,20])

[eps_xx,eps_yy,eps_zz,eps_xz] = EvaluateStrainAxi(Mesh,ur,uz);
figure(3); FEMtrimesh(Mesh,eps_xx); xlabel('r'); ylabel('z'); zlabel('eps_{xx}')
figure(4); FEMtrimesh(Mesh,eps_yy); xlabel('r'); ylabel('z'); zlabel('eps_{yy}')
figure(5); FEMtrimesh(Mesh,eps_zz); xlabel('r'); ylabel('z'); zlabel('eps_{zz}')
figure(6); FEMtrimesh(Mesh,eps_xz); xlabel('r'); ylabel('z'); zlabel('eps_{xz}')
W = EvaluateEnergyDensityAxi(Mesh,eps_xx,eps_yy,eps_zz,eps_xz,E,nu);
figure(7); FEMtrimesh(Mesh,W); xlabel('r'); ylabel('z'); zlabel('energy density')
r = Mesh.nodes(:,1);
EnergyIntegrated = FEMIntegrate(Mesh,2*pi*r.*W)
EnergyDensity    = EnergyIntegrated/(4/3*pi*R^3/2)
```

9.45 A crook with a weight attached

Examine the two L–shaped steel beams in Figure 235(a). Each beam has length $L = H = 0.1$ with a square cross section of 0.01×0.01. The top edge is fixed and at the right end there is a force of 100 N (i.e. a weight of 10 kg) pulling the beam downwards. The corner at $(x, y) = (0, 0)$ is slightly rounded, since the highest stresses are expected to show up in this area, see Figure 235(b). This is alos to avoid stress concentration. The applied force of 100 N leads to a surface force density of $gN_2 = \frac{100\,\mathrm{N}}{0.01^2\,\mathrm{m}^2} = 10^6\,\frac{\mathrm{N}}{\mathrm{m}^2}$.

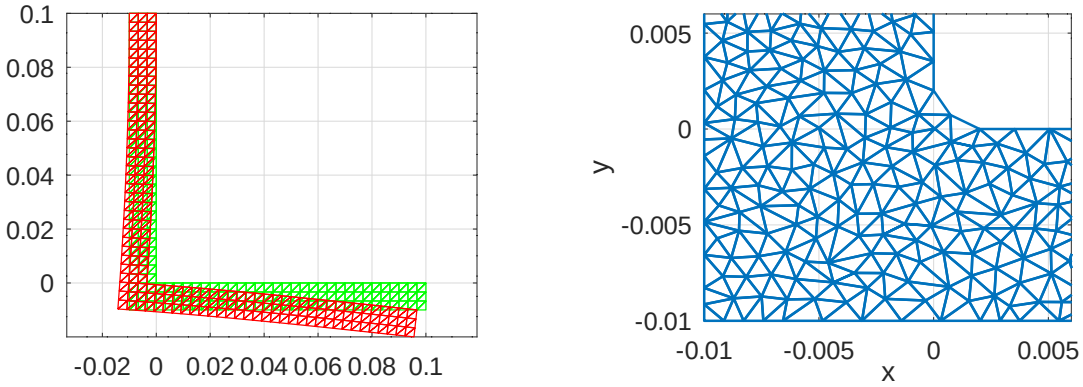
Start out by defining the domain and generating the mesh with the help of `CreateMeshTriangle()`. To avoid shear–locking use `MeshUpgrade()` to generate second order elements.

Crook.m

```
W = 0.01; H = 0.1; Load = 1e6;;
Layers = 2*5; gap = W/5;

if 0 %% no rounding
  Domain = [-W -W -22; -W H -11; 0 H -22; 0 gap -22;..
             0 0 -22; gap 0 -22; H 0 -23; H -W -22];
else %% with a rounded corner
  Domain = [-W -W -22; -W H -11; 0 H -22; 0 gap -22;...
            gap*0.366 gap*0.366 -22; gap 0 -22; H 0 -23; H -W -22];
endif

FEMmesh = CreateMeshTriangle ('Crook1',Domain,(W/Layers)^2);
figure(1); FEMtrimesh(FEMmesh); xlabel('x'); ylabel('y'); axis([-W 3*gap -W 3*gap])
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
```



(a) the domain (b) the grid at the corner

Figure 235: Original and deformed domain for the crook with attached weight at the right edge

Then find the approximate displacements u_1 and u_2 by calling `PlaneStress()`. The code segment below estimates the maximal vertical displacement by $-8.96 \cdot 10^{-4}$ m, i.e. approximately -0.9 mm. Then examine the vertical displacement generated by the horizontal section of the crook. To verify the order of magnitude of the displacement use two elementary mechanical arguments:

1. For a bending Euler beam with the dimensions of one arm obtain

$$u_2(L) = -\frac{4\,F}{E\,W\,H^3}\,L^3 \approx -\frac{4 \cdot 10^2}{200 \cdot 10^9\,0.01^4}\,0.1^3 = -2 \cdot 10^{-4}\,,$$

   i.e. a displacement of 0.2 mm.

2. The slope of the lower arm at the left starting point is estimated by $-6.56 \cdot 10^{-3}$ and with the length $H = L = 0.1$ this leads to another contribution of ≈ 0.56 mm.

The sum of the two contributions is not too far from the result 0.9 mm by FEMoctave.

Crook.m

```
E = 200e9; nu = 0.25; %%% steel
[u1,u2] = PlaneStress(FEMmesh,E,nu,{0,0},{0,0},{0,-Load});

MaximalDisplacement = min(u2)
[~,slope_x,~] = FEMgriddata(FEMmesh,u2,0,-W/2)
i = linspace(-0.01,0.1)'; xi =-0.005*ones(size(yi));
u1i = FEMgriddata(FEMmesh,u1,xi,yi);
figure(8); plot(yi,u1i)
           xlabel('y'); ylabel('u_1')
p = polyfit(yi,u1i,2);               %% linear regression of a polynomial of degree 2
slope = polyval([2*p(1) p(2)],-W/2)  %% evaluate the derivative of the polynomial
-->
MaximalDisplacement = -8.9570e-04
slope_x = -6.5645e-03
slope   =  6.6130e-03
```

To determine the slope of the horizontal beam at the left starting point the result by FEMoctave was used above. One can use an analytical approximation by using the moment applied to the vertical beam, generated by the force at the right endpoint. Along the centerline of the vertical beam use

$$\frac{\partial^2 u_1(y)}{\partial y^2} = \frac{-F\,(H+W/2)}{E\,I} = \frac{-F\,(H+W/2)}{E\,\frac{1}{12}\,W^3\,W} \approx 6.3\cdot 10^{-2}\ .$$

Then use the conditions $u_1(H) = u_1(0.1) = \frac{\partial\, u_1(H)}{\partial y} = 0$ at the top edge to estimate $\frac{\partial\, u_1(-W/2)}{\partial y} \approx 6.62\cdot 10^{-3}$, which is rather close to the FEMoctave result of $6.56\cdot 10^{-3}$. The horizontal displacement u_1 along the centerline of the vertical beam is shown in Figure 238(a).

To generate Figure 235(a) with the scaled deformation also shown, start out by creating a coarse mesh and evaluate the displacement at those nodes. Then show the original and deformed mesh with different colors.

Crook.m

```
CoarseMesh = CreateMeshRect([-W:W/3:H],[-W:W/3:H],-11,-11,-11,-11);
x = CoarseMesh.nodes(:,1); y = CoarseMesh.nodes(:,2);
u1i = FEMgriddata(FEMmesh,u1,x,y); u2i = FEMgriddata(FEMmesh,u2,x,y);
x(isnan(u1i)) = NaN;

figure(2); clf; factor = H/10/abs(min(u2));
  trimesh(CoarseMesh.elem,x,y,'color','green','linewidth', 1); hold on;
  trimesh(CoarseMesh.elem,x+factor*u1i,y+factor*u2i,'color','red','linewidth',1)
  axis equal; hold off
```

To examine the mechanical load of the structure evaluate the stresses by calling `EvaluateStress()`. By asking for three return arguments a plane stress model is used. It is easy to generate graphs of the whole structure, but more insight might be gained by a closer look at some slices.

- At height $y = \frac{H}{2} = 0.05$ examine the normal stress σ_y in y–direction. The result in Figure 236(a) show a compression in the left segment and traction on the right. This corresponds to the bending on the vertical arm. By integrating σ_y along this slice one should obtain the value of the force applied on the right edge of the crook, i.e.

  $$W\int_{-W}^{0} \sigma_y(x,0.05)\,dx \approx \text{Force} = 100\ .$$

  For the moment with respect to the origin $(0,0)$ we expect

  $$W\int_{-W}^{0} x\,\sigma_y(x,0.05)\,dx \approx \text{H}\cdot\text{Force} = 10\ .$$

  Both results are confirmed by the code below. The values of the normal stress σ_x are approximately zero.

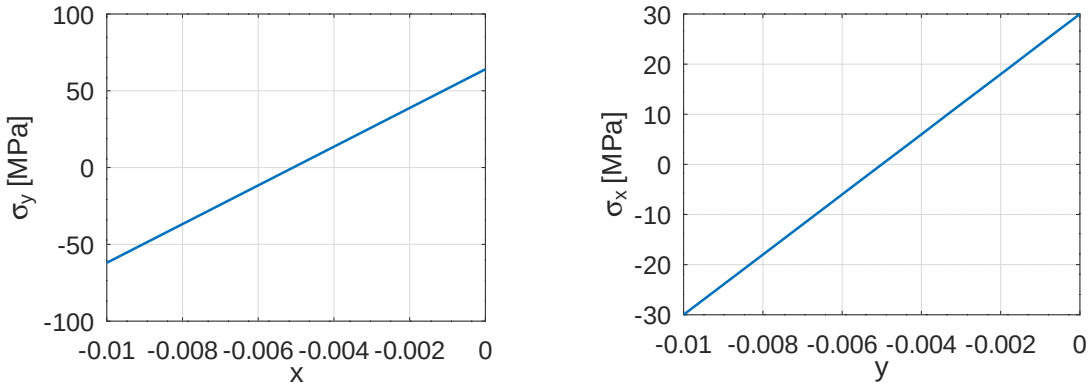
- At $x = \frac{H}{2} = 0.05$ examine the normal stress σ_x in x–direction along a vertical slice. The result in Figure 236(b) shows a compression in the lower segment and traction in the upper segment. This corresponds to the downward bending on the horizontal arm. For the moment with respect to the point $(\frac{H}{2}, 0)$ we expect

$$W \int_{-W}^{0} y\, \sigma_x(0.05, y)\, dy \approx \frac{1}{2}\,\mathrm{H} \cdot \mathrm{Force} = 5\ .$$

  The values of the normal stress σ_y are approximately zero. By integrating the shearing stress τ_{xy} obtain again the applied force, i.e.

$$W \int_{-W}^{0} \tau_{xy}(0.05, y)\, dy \approx \mathrm{Force} = -100\ .$$

- Observe that the stress values in the vertical arm are considerably larger than in the horizontal arm. The values in Figure 236(b) are at $x = 0.05$. For larger values of x the strains σ_x will be even smaller.



(a) a horizontal slice at $y = 0.05$ (b) a vertical slice at $x = 0.05$

Figure 236: A horizontal slice with σ_y shown and a vertical slice with σ_x shown

Crook.m

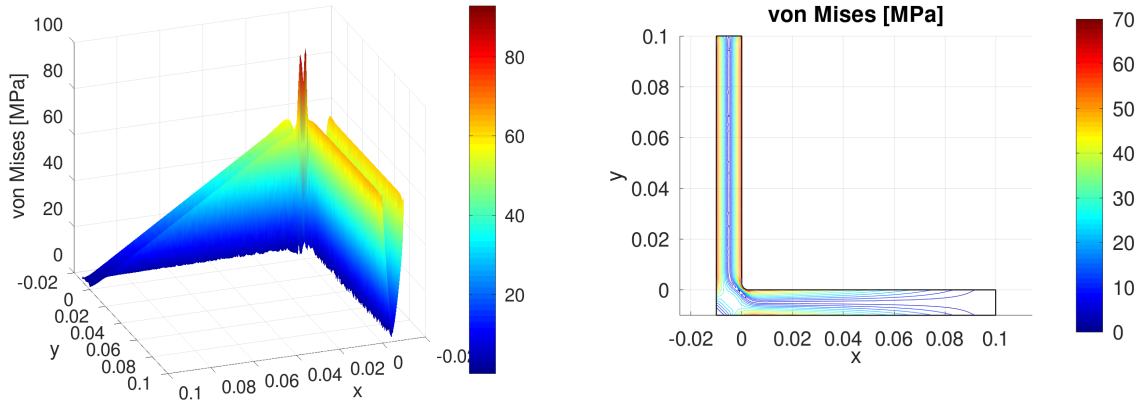
```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(FEMmesh,u1,u2,E,nu);
dist = linspace(-W,0,100)'; HH = H/2*ones(size(dist));
sigma_y_slice_H = FEMgriddata(FEMmesh,sigma_y,dist,HH);
figure(3); plot(dist,sigma_y_slice_H/1e6);
           xlabel('x'); ylabel('\sigma_y [MPa]');xlim([-W,0])

sigma_y_slice_H(isnan(sigma_y_slice_H)) = 0;
Integral_sigma_y = W*trapz(dist,sigma_y_slice_H)
Integral_Moment  = W*trapz(dist,dist.*sigma_y_slice_H)

sigma_x_slice_V = FEMgriddata(FEMmesh,sigma_x,HH,dist);
tau_xy_slice_V  = FEMgriddata(FEMmesh,tau_xy,HH,dist);
figure(4); plot(dist,sigma_x_slice_V/1e6);
           xlabel('y'); ylabel('\sigma_x [MPa]');xlim([-W,0])
Integral_Moment_x = W*trapz(dist,dist.*sigma_x_slice_V)
Integral_tau_xy   = W*trapz(dist,tau_xy_slice_V)
-->
Integral_sigma_y  =  99.769
Integral_Moment   =  10.002
Integral_Moment_x =   5.0005
Integral_tau_xy   = -99.985
```

Since steel is a ductile material one can use the von Mises stress to decide whether the crook will withstand the force of 100 N. Use `EvaluateVonMises()` to find the values of the von Mises stress at the nodes and then

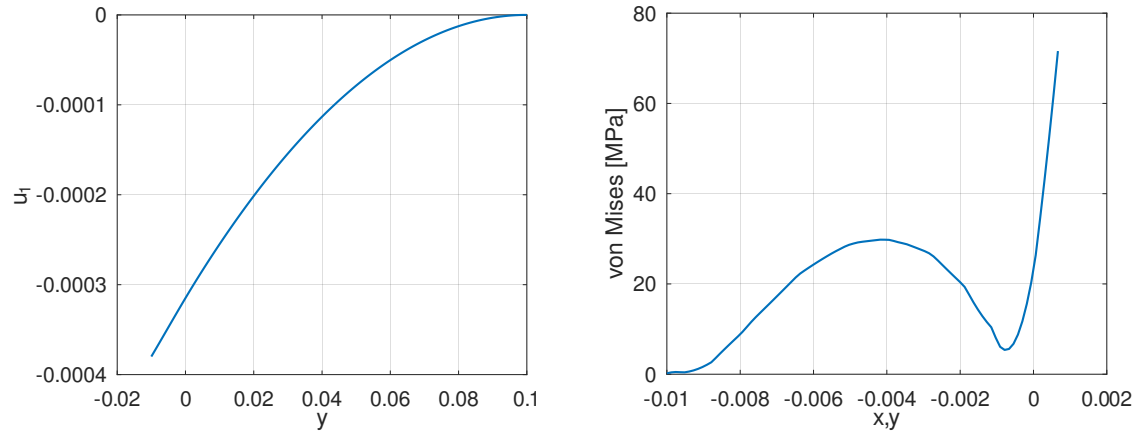
`FEMtrisurf()` and `FEMtricontour()` to generate Figure 237. The contour lines in Figure 237 are supplemented with the borders of the domain. The spikes of the von Mises stress at the corner $(0, 0)$ should be no surprise to mechanical engineers. One possible measure to reduce the maximal value of von Mises is the rounding visible in Figure 235(b). To obtain more insight the von Mises stress is evaluated along the straight line connecting $(-W, -W)$ and $(0, 0)$, using `FEMgriddata()`, leading to Figure 238(b). Since the yield strength of steel is $\approx$ 330 MPa the crook should be able to support the force of 100 N .



(a) the surface, in MPa

(b) the contour lines

Figure 237: The von Mises stress on the crook, as surface and level curves



(a) bending in vertical beam

(b) von Mises stress across corner

Figure 238: The bending of the centerline of the vertical beam and the von Mises stress on the $45°$ line through the origin

Crook.m

```
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);
figure(5); FEMtrisurf(FEMmesh,vonMises/1e6);
           xlabel('x'); ylabel('y'); zlabel('von Mises [MPa]'); view(160,25)
           colorbar(); shading interp
figure(6); clf;  FEMtricontour(FEMmesh,vonMises/1e6,1e1*[0:0.5:6]);
           xlabel('x'); ylabel('y'); title('von Mises [MPa]');
           caxis(1e2*[0 0.7]); axis equal; colorbar(); hold on
           plot([Domain(:,1);Domain(1,1)],[Domain(:,2);Domain(1,2)],...
```

```
            'color','black','linewidth',1); hold off

dist = linspace(-W,gap,100)'; HH = H/2*ones(size(dist));
vonMises_slice = FEMgriddata(FEMmesh,vonMises,dist,dist);
figure(7); plot(dist,vonMises_slice*1e-6);
            xlabel('x,y'); ylabel('von Mises [MPa]')
```

9.46 A wrench

A classical example application for mechanical FEM is a wrench. With a digital image of a typical wrench the tool `xinput()` is used in *Octave* to grab the contour data from the screen and written to the file `WrenchData.m`, see [Stah22, §3.9]. Then rescale the contour to obtain a typical length of 0.15 m of the wrench in Figure 239(a). Then setup an appropriate configuration of the wrench.

- The material is steel with the parameters $E = 200$ GPa and $\nu = 0.25$.
- Most of the boundary is force free, thus with the code -22, according to Table 6 on page 62.
- Along the two horizontal sections on the very left the displacements are zero, modeling the screw head in the wrench. A closer look at the contour data shows that these are sections 1 and 4 of the contour, used with the code -11 for the boundary condition.
- The applied force is 100 N over a length of $0.05\ m$ and width 0.005 m, leading to a force density of $\frac{100}{0.05 \cdot 0.005} = 4 \cdot 10^5\ \frac{\mathrm{N}}{\mathrm{m}^2}$. This load is applied on segment 17 of the contour, used with the code -23 for no force in x direction and the given load in y direction.

With this data the mesh is generated by calling `CreateMeshTriangle()`. Then the mesh of linear elements should be upgraded to quadratic or cubic elements with the help of `MeshUpgrade()`. Then use `PlaneStress()` to solve for the displacements u_1 and u_2.

Wrench.m

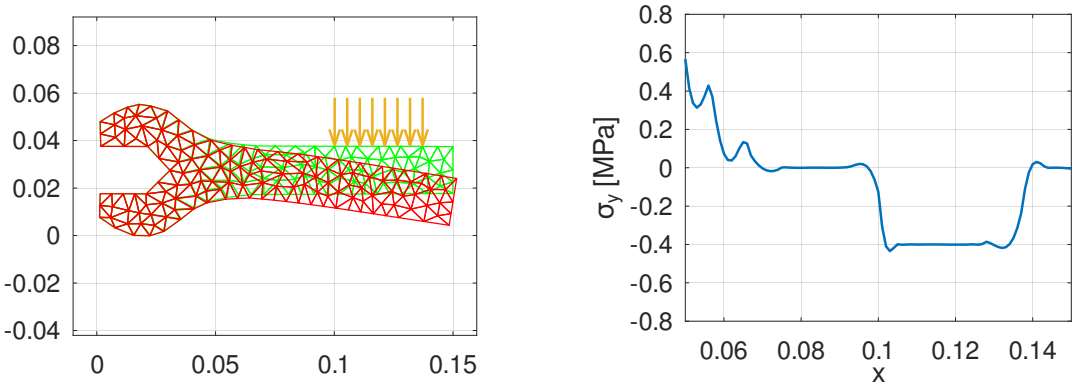
```
load WrenchData.m                 %% load the contour data
scale = 0.15/max(x);              %% scale the contour data
x = scale*x; y = scale*y;
Order = 3;                        %% select the order of the elements 1,2 or 3
BC = -22*ones(size(x));           %% default is a force free boundary
BC([1 4]) = -11; BC(17) = -23;    %% fixed at the two horizontal section on the left
                                  %% vertical force on top right segment
Load = 100/(0.05*0.005);          %% 100 N, distributed over length 0.05 and width 0.005

Mesh = CreateMeshTriangle('Wrench',[x,y,BC],0.01^2/4); %% create the mesh
switch Order
  case 2  Mesh = MeshUpgrade(Mesh,'quadratic');
  case 3  Mesh = MeshUpgrade(Mesh,'cubic');
endswitch
E = 200e9; nu = 0.25; gN = {0,-Load};              %% data for steel
[u1,u2] = PlaneStress(Mesh,E,nu,{0,0},{0,0},gN);   %% solve the plane stress problem
```

With the solution the original and deformed shape can be displayed with the applied load visualized by a few vectors, see Figure 239(a).

Wrench.m

```
%%display the original and deformed wrench, with the applied force
scale = 0.001*max(y)/max(u2);
x_force = linspace(x(17),x(18),8); y_force = 0.038*ones(size(x_force))+0.02;
vec_x = zeros(size(x_force)); vec_y = -0.02*ones(size(x_force));
figure(1); clf
trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),...
        'color','green','linewidth',1); hold on
trimesh(Mesh.elem,Mesh.nodes(:,1)+scale*u1,Mesh.nodes(:,2)+scale*u2,...
        'color','red','linewidth',1)
quiver(x_force,y_force,vec_x,vec_y,0)
hold off; axis equal; xlim([-0.01, 0.16])
```



(a) the wrench, original and deformed

(b) σ_y along upper edge

Figure 239: The deformed wrench and the stress σ_y along upper edge with the applied load

Evaluate the stresses at the nodes, including the von Mises Stress. By asking for three return arguments the plane stress situation is used. By a piecewise linear interpolation and `FEMgriddata()` the vertical stress σ_y can be evaluated along the upper edge, leading to Figure 239(b). The external load of -0.4 MPa is clearly visible.

Wrench.m

```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(Mesh,u1,u2,E,nu);  %% basic stress
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);         %% von Mises stress
xi = linspace(0.05,0.15,101)'; yi = interp1(x(14:19),y(14:19),xi);
sigma_y_interp = FEMgriddata(Mesh,sigma_y,xi,yi);
figure(2); plot(xi,sigma_y_interp/1e6)
           xlabel('x'); ylabel('\sigma_y [MPa]'); xlim([0.05,0.15])
```

Since steel is a ductile metal the von Mises stress can be used to examine the effect on the wrench. In Figure 240(a) find the surface plot of the von Mises stress. The highest stress is on the boundary at the mid section, but spikes are also visible at the sharp corners on the left. Figure 240(b) shows the contour lines and the position of the highest and lowest von Mises stress. It should be no surprise that the section on the very right is almost stress free.

Wrench.m

```
figure(3); clf; FEMtrimesh(Mesh,vonMises/1e6)
          zlabel('von Mises stress');  colorbar(); view([40 75])
          xlim([0 0.15]); ylim([-0.025 0.09]);
          set(gca, 'XTickLabel', [], 'yTickLabel', [], 'zTickLabel', [])
MaxVonMises = max(vonMises); MinVonMises = min(vonMises);
Max_Min_vonMises_MPa = [MaxVonMises,MinVonMises]/1e6
MaxInd = find(vonMises == MaxVonMises); MaxPosition = Mesh.nodes(MaxInd,:);
MinInd = find(vonMises == MinVonMises); MinPosition = Mesh.nodes(MinInd,:);
figure(4); clf; FEMtricontour(Mesh,vonMises/1e6,41)
            hold on; plot([x;x(1)],[y;y(1)],'k');
            plot(MaxPosition(1),MaxPosition(2),'*r',MinPosition(1),MinPosition(2),'*b');
            hold off; axis equal
-->
 Max_Min_vonMises_MPa =  1.2415e+01   2.4078e-03
```

9.47 A rotating rubber cylinder

A cylinder with radius $R = 0.2$ and height $2\,H = 0.2$ is rotating about the z–axis with 10 revolutions per second. The wall consist of a Silicone rubber of thickness 0.01 and cover and bottom are 0.02 thick. The goal is to determine the resulting deformation and the von Mises stress.

Using an axially symmetric setup only a cross section in the $y = 0$ plane for $x = r > 0$ have to be examined. Since the setup is symmetric with respect to the plane $z = 0$ only the upper half has to be modeled, using the zero z displacement at the lower edge.

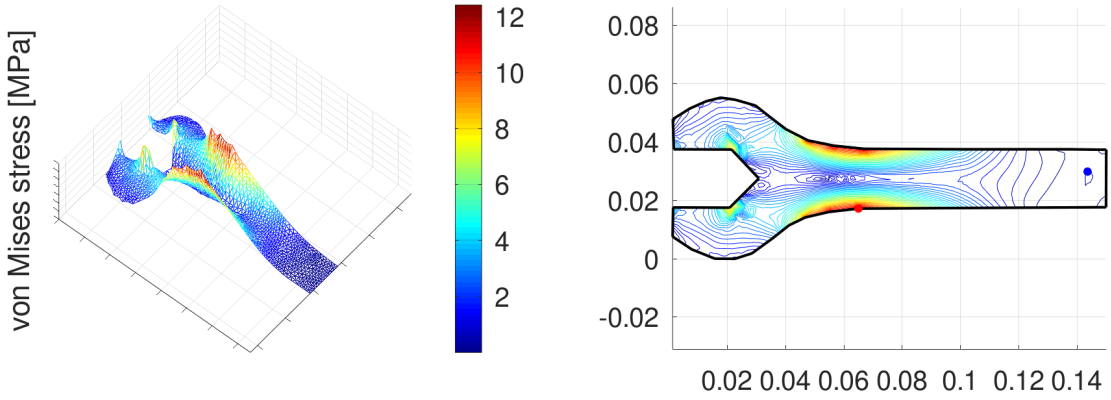


Figure 240: Surface and contour plot of the von Mises stress in [MPa]

Start by defining the parameters and generating the mesh. In this case third order elements are used. Then define the function for the centrifugal force and solve for the two displacements u_r and u_z with the help of `AxiStress()`. Then display the original and the deformed domain in Figure 241 and the displacements in Figure 242.

RubberBox.m

```
rho = 1100; E = 1e6; nu = 0.47;  %% Silicone rubber
H = 0.1; R = 0.2; W = 0.01;
Contour = [0 H -11;0 H-2*W -22; R-2*W H-2*W -22; R-W H-2*W -22;
           R-W 0 -21; R 0 -22; R H-W -22; R-W H -22];
Mesh = CreateMeshTriangle('RubberBox',Contour,3e-5);
Mesh = MeshUpgrade(Mesh,'cubic');

function res=fr(xy,dummy)
  freq = 10; omega = freq*2*pi; rho = 1100;
  res = rho*xy(:,1)*omega^2;
endfunction

[ur,uz] = AxiStress(Mesh,E,nu,{'fr',0},{0,0},{0,0});

figure(10); ShowDeformation(Mesh,ur,uz,1); axis equal; xlabel('x'); ylabel('y');
figure(11); FEMtrimesh(Mesh,ur); xlabel('r'); ylabel('z'); zlabel('u_r')
figure(12); FEMtrimesh(Mesh,uz); xlabel('r'); ylabel('z'); zlabel('u_z')
```

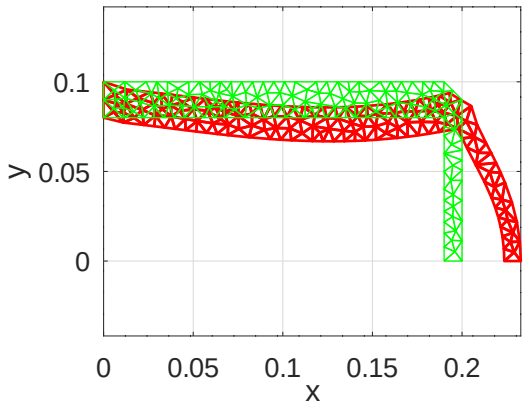
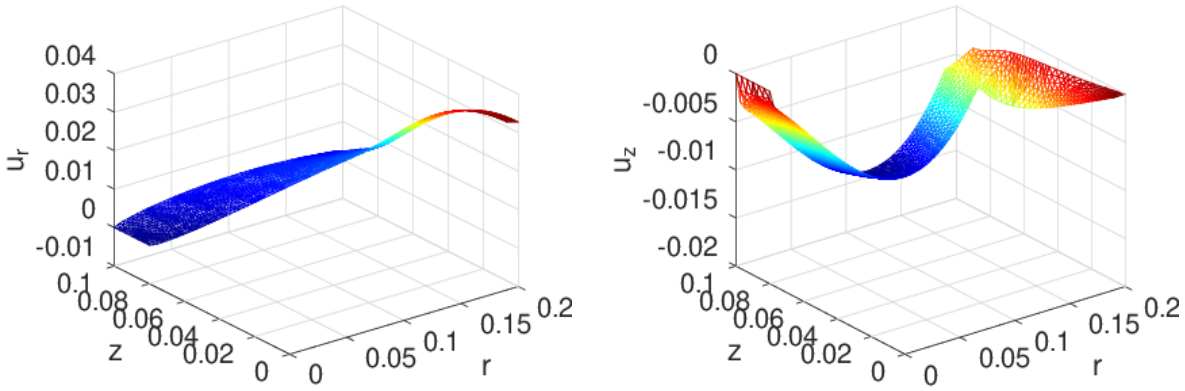


Figure 241: The upper half of the original and deformed domain for the rotating rubber box



(a) radial displacement u_r

(b) z–displacement u_z

Figure 242: The displacements u_r and u_z for the rotating rubber box

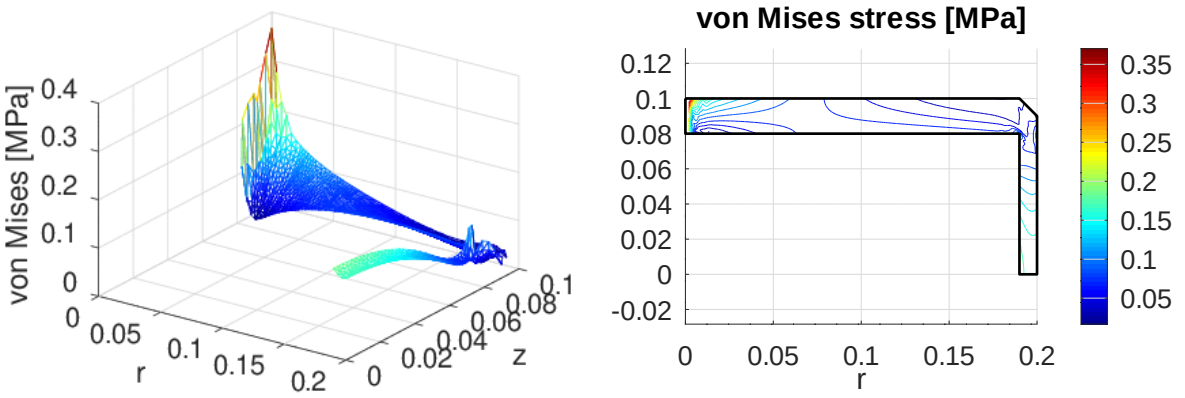


Figure 243: The von Mises stress for the rotating rubber box

As last step evaluate the stresses and then the von Mises stress, leading to the surface and contour plots in Figure 243.

RubberBox.m

```
[sigma_x,sigma_y,sigma_z,tau_xz] = EvaluateStressAxi(Mesh,ur,uz,E,nu);
vonMises = EvaluateVonMisesAxi(sigma_x,sigma_y,sigma_z,tau_xz);
figure(13); FEMtrimesh(Mesh,vonMises/1e6)
            xlabel('r'); ylabel('z'); zlabel('von Mises [MPa]'); view([35 30])
figure(14); clf; FEMtricontour(Mesh,vonMises/1e6)
            xlabel('r'); ylabel('z'); zlabel('von Mises [MPa]')
            hold on; plot([Contour(:,1);Contour(1,1)],[Contour(:,2);Contour(1,2)],'k')
            hold off; axis equal; colorbar; title('von Mises stress [MPa]')
```

9.48 A washer fastener examined as spring

In this example a washer fastener design is examined. The goal is to determine the force required to deform the washer.

9.48.1 The setup

- The material is aluminum, with density $\rho = 2700\,\frac{\text{kg}}{\text{m}^3}$, Young's modulus $E = 70\,\text{GPa}$ and Poisson ratio $\nu = 0.33$.
- The intersection of the washer with the plane $y = 0$ is almost rectangular. The inner part is moved up slightly and there are two horizontal sections, one at the inner/upper location at height $z = 0.001$ and the second at the lower/outer section at height $z = 0$. Find the domain in Figure 244. The corners of the domain are given by the six points

r [m]	0.0020	0.0020	0.0044	0.0050	0.0050	0.0026
z [m]	0.0010	0.0004	0	0	0.0006	0.0010

  and connected by straight line segments. This domain in then rotated about the z–axis to obtain the washer in $\mathbb{R}^3$.

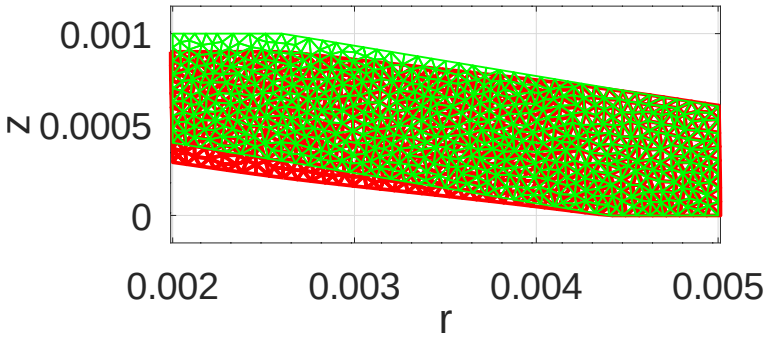


Figure 244: The original and deformed domain of the washer

To determine the resulting deformation of the washer the boundary conditions have to be specified.

- The outer/lower edge at height $z = 0$ is fixed in z–direction, but free in radial direction.
- The inner/upper edge at height $z = 0.001$ is moved downward by 0.0001 m and free in radial direction.
- All other edges are force free.

With this information the boundary value problem can be solved, using the command `AxiStress()`. The code contains additional configurations with different boundary conditions on the inside and outside. Display the radial displacement u_r in Figure 245 and the height displacement u_z in Figure 246.

WasherSpring.m

```
rho = 2700; E = 70e9; nu = 0.33;  %% Aluminum
H = 0.001; Ri = 0.002; Ro = 0.005; D = 0.0006; H = 0.0004;
global Offset; Offset = 1*1e-4;

if 1 %% free sides
  Contour = [Ri H+D -22; Ri H -22;Ro-D 0 -21; Ro 0 -22; Ro D -22;Ri+D H+D -21];
elseif 0 %% clamped on the outside
  Contour = [Ri H+D -22; Ri H -22;Ro-D 0 -21; Ro 0 -12; Ro D -22;Ri+D H+D -21];
else  %% clamped on both sides
  Contour = [Ri H+D -12; Ri H -22;Ro-D 0 -21; Ro 0 -12; Ro D -22;Ri+D H+D -21];
endif
Mesh = CreateMeshTriangle('Washer',Contour,2.5e-9);
%%Mesh = MeshUpgrade(Mesh,'quadratic');
Mesh = MeshUpgrade(Mesh,'cubic');

function res = gDz(xy,dummy)
  global Offset
  res = -Offset*(xy(:,2)>Offset);
endfunction

[ur,uz] = AxiStress(Mesh,E,nu,{0,0},{0,'gDz'},{0,0});
figure(10); ShowDeformation(Mesh,ur,uz,1); xlabel('r'); ylabel('z');
            axis equal; xticks([2:5]/1000); yticks([0:0.5:1]/1000)
```

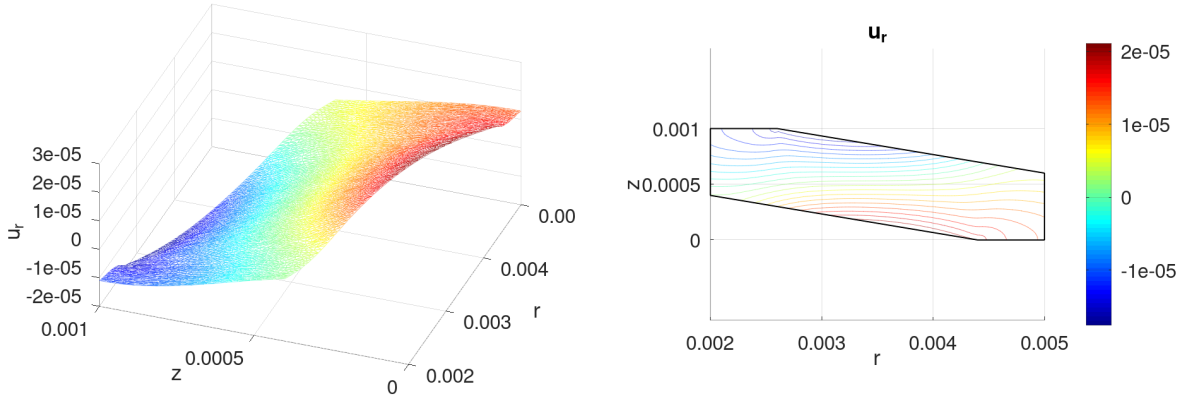


Figure 245: The radial displacement u_r

WasherSpring.m

```
figure(11); FEMtrimesh(Mesh,ur); xlabel('r'); ylabel('z'); zlabel('u_r')
            xticks([2:5]/1000); yticks([0:0.5:1]/1000)
Cx = [Contour(:,1);Contour(1,1)]; Cy = [Contour(:,2);Contour(1,2)];
figure(21); clf; FEMtricontour(Mesh,ur);  hold on ;  plot(Cx,Cy,'k'); hold off
            xlabel('r'); ylabel('z'); title('u_r');
            axis equal; colorbar;xticks([2:5]/1000); yticks([0:0.5:1]/1000)
figure(12); FEMtrimesh(Mesh,uz); xlabel('r'); ylabel('z'); zlabel('u_z');
            xticks([2:5]/1000); yticks([0:0.5:1]/1000)
figure(22); clf; FEMtricontour(Mesh,uz);  hold on ;  plot(Cx,Cy,'k'); hold off
            xlabel('r'); ylabel('z'); title('u_z');
            axis equal; colorbar;xticks([2:5]/1000); yticks([0:0.5:1]/1000)
```

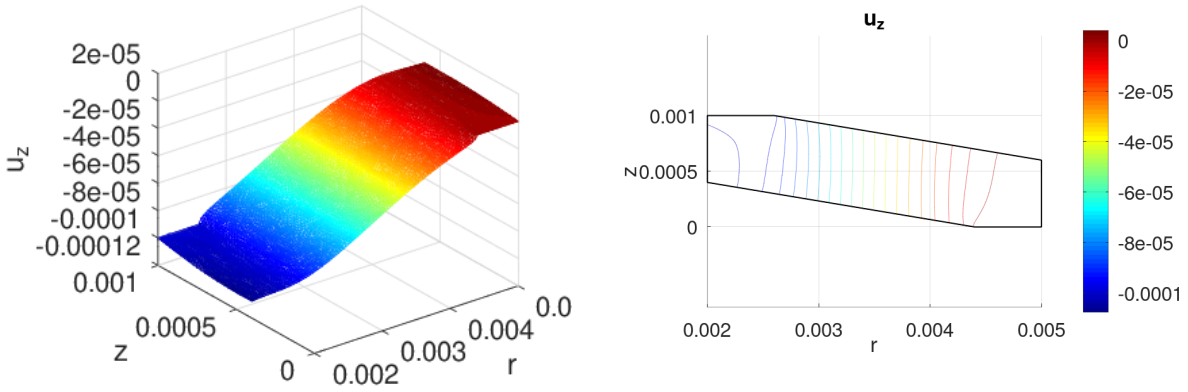


Figure 246: The height displacement u_z

9.48.2 Evaluate the force by integrating the normal stress

To determine the force F required to push the upper edge down by 0.1 mm use the normal stress σ_z in vertical direction.

WasherSpring.m

```
[sigma_x,sigma_y,sigma_z,tau_xz] = EvaluateStressAxi(Mesh,ur,uz,E,nu);
figure(13); FEMtrimesh(Mesh,sigma_z*1e-6)
            xlabel('r'); ylabel('z'); zlabel('\sigma_z [MPa]');
            xticks([2:5]/1000); yticks([0:0.5:1]/1000)
figure(23); clf; FEMtricontour(Mesh,sigma_z/1e6,[-20:1:20]*100)
            hold on ;  plot(Cx,Cy,'k'); hold off
            xlabel('r'); ylabel('z'); title('\sigma_z [MPa]');
            axis equal; colorbar;xticks([2:5]/1000); yticks([0:0.5:1]/1000)
```

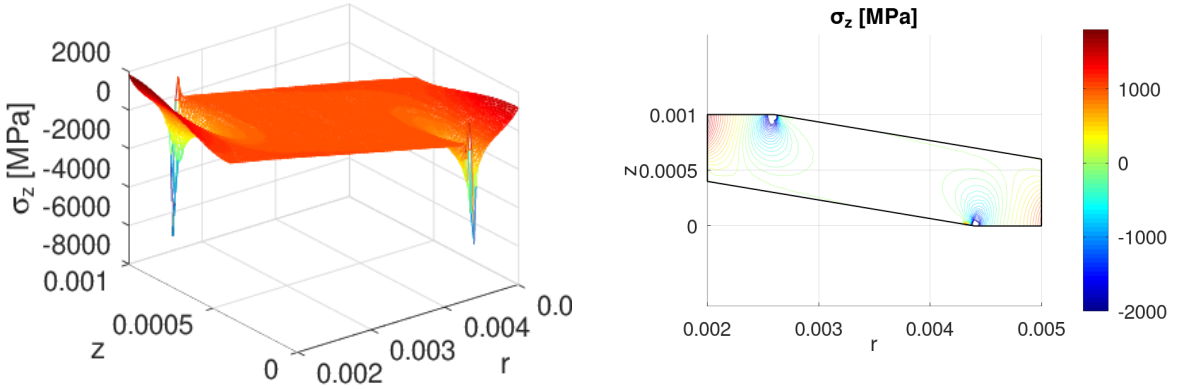


Figure 247: The normal stress σ_z

At any height $0 \le h \le 0.001$ examine the slice $a \le r \le b$ in the domain visible in Figure 244 and perform an integration to determine F.

$$F \;=\; 2\,\pi \int_{r=a}^{b} r\,\sigma_z(r,h)\;dr$$

Examine the graph of the normal stress σ_z in Figure 247 and observe the singularities at the corners of the edges with fixed displacement. These singularities cause serious numerical trouble when trying to integrate along the upper or lower edges.

On a mesh with elements of order 3 with 5968 free nodes obtain the numerical results[65]

$$\begin{aligned} F_{up} &\approx 1415.1 \text{ N} \\ F_{middle} &\approx 2401.6 \text{ N} \\ F_{low} &\approx 1085.8 \text{ N} \end{aligned}$$

and the graphs in Figure 248. By changing the element types or the size of the meshes the results at the lower and upper edges can change substantially, while the result at half height remains stable and thus is more reliable. By changing the height of the slice in the code below (modify the value of `s`) one may observe that the results for heights between 20% and 80% are stable.

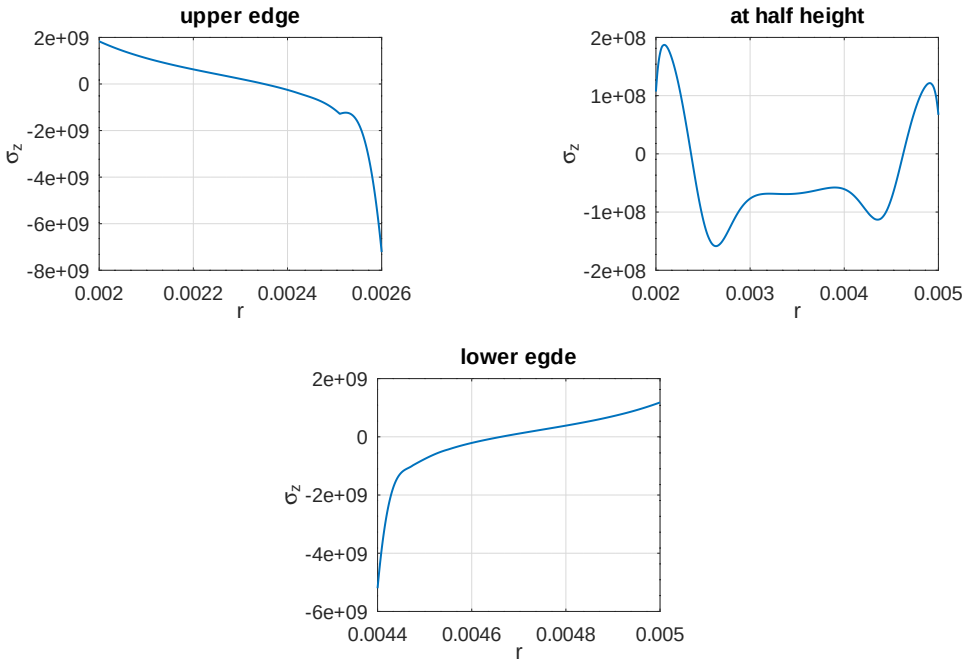


Figure 248: The normal pressures σ_z along upper and lower edge and at half height

WasherSpring.m

```
r = linspace(0,D,1000)';
sigma_up = FEMgriddata(Mesh,sigma_z,Ri+r,(H+D)*ones(size(r)));
figure(31); plot(Ri+r,sigma_up); xlabel('r'); ylabel('\sigma_z'); title('upper edge')
xlim([Ri,Ri+D]); xticks([2:0.2:2.6]/1000);
Force_up = 2*pi*trapz(Ri+r,sigma_up.*(Ri+r))

sigma_low = FEMgriddata(Mesh,sigma_z,Ro-D+r,zeros(size(r)));
figure(32); plot(Ro-D+r,sigma_low); xlabel('r'); ylabel('\sigma_z'); title('lower egde')
xlim([Ro-D, Ro]);  xticks([4.4:0.2:5]/1000);
Force_low = 2*pi*trapz(Ro-D+r,sigma_low.*(Ro-D+r))

s = 0.5;  %% select the height
r_mid = linspace(Ri,Ro,1000)';
sigma_mid = FEMgriddata(Mesh,sigma_z,r_mid,s*(H+D)*ones(size(r_mid)));
ind = find(isfinite(sigma_mid));
r_mid = r_mid(ind); sigma_mid = sigma_mid(ind);
figure(33); plot(r_mid,sigma_mid); xlabel('r'); ylabel('\sigma_z');
            title('at half height');  xticks([2:5]/1000);
Force_mid = 2*pi*trapz(r_mid,sigma_mid.*r_mid)
```

[65] A computation with Comsol Multiphysics leads to a force of 2395.5 N at half height and an elastic energy of 0.12077 J. The shape of the graphs in Figure 248 is confirmed.

9.48.3 Evaluate the force by an energy argument

Since the above evaluation of the required force F is rather delicate, it is a good idea to examine an alternative approach. Since the problem is linear the force F depends linearly on the displacement d of the upper edge, i.e. with a displacement $0 \le s \le d$ obtain $F(s) = k\,s = \frac{s}{d}\,F(d)$. An elementary integration leads to the energy U in the system.

$$U = \int_0^d F(s)\,ds \;=\; \int_0^d \frac{s}{d}\,F(d)\,ds \;=\; \frac{1}{2}\,d\,F(d)$$

Thus find the force $F = F(d) = \frac{2\,U}{d}$. Using the results in Section 8.7 (starting on page 236) the elastic energy U is given by

$$\begin{aligned} U(\vec{u}) \;=\; & 2\,\pi \iint\limits_{\Omega} \frac{r\,E}{2\,(1+\nu)\,(1-2\,\nu)} \Big((1-\nu)\,((\frac{\partial\,u_r}{\partial r})^2 + (\frac{\partial\,u_z}{\partial z})^2 + \frac{1}{r^2}\,u_r^2) + \\ & +2\,\nu\,((\frac{\partial\,u_r}{\partial r})(\frac{\partial\,u_z}{\partial z}) + \frac{1}{r}\,u_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z})) \Big)\,dA + \\ & +2\,\pi \iint\limits_{\Omega} \frac{r\,E}{1+\nu}\,\frac{1}{4}\,(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r})^2\,dA\,. \end{aligned}$$

Since the displacements u_r and u_z are available use the function `FEMIntegrate()`. It is best[66] to evaluate the displacements and their partial derivatives at the Gauss points by using `FEMEvaluateGP()` and then integrate. For this example the result is $F = 2402.1$ N, which is close to the above integration at half height of the normal stress σ_z.

WasherSpring.m

```
[urGP,ur_rz] = FEMEvaluateGP(Mesh,ur);  [uzGP,uz_rz] = FEMEvaluateGP(Mesh,uz);
rGP = Mesh.GP(:,1);
w = E/(2*(1+nu)*(1-2*nu))*rGP.*((1-nu)*(ur_rz(:,1).^2+uz_rz(:,2).^2+(urGP./rGP).^2)...
    + 2*nu*(ur_rz(:,1).*uz_rz(:,2) +1./rGP.*urGP.*(ur_rz(:,1)+uz_rz(:,2))))...
    +E/(4*(1+nu))*rGP.*(ur_rz(:,2)+uz_rz(:,1)).^2;
U_elast = 2*pi*FEMIntegrate(Mesh,w);
Force_energy = 2*U_elast/Offset
```

9.48.4 Comparison of linear, quadratic and cubic elements

The above results were generated with a mesh consisting of 1294 triangle and piecewise cubic functions. It is easy to recompute, using the same number of triangles, but linear or quadratic functions. Find the results in Table 24.

- The FEM algorithm is minimizing the energy U of the system amongst the functions to be used. The space of piecewise linear functions is a strict subspace of the piecewise quadratic functions. Thus the minimal energy will be smaller when using elements of order 2 than with elements of order 1. As a consequence linear element will overestimate the resulting force $F = \frac{2\,U}{d}$.
- The space of piecewise quadratic functions is a strict subspace of the piecewise cubic functions. Thus the minimal energy will be smaller when using elements of order 3 than with elements of order 2. The force F evaluated with elements of order 3 will be the smallest. This is confirmed in Table 24.
- Using finer meshes will lead to smaller minimal energies U and thus smaller forces F.
- The estimates in Section 5.4 on page 126 lead to factors of 0.5, 2 or 4.5 for the ratio of number of nodes divided by the number of triangles for linear, quadratic or cubic elements. This is confirmed in Table 24.

9.48.5 Effect of different boundary conditions

The above setup can be modified by changing the boundary conditions at the inner or outer edge.

- In the original setup both sides are free to move in radial direction.
- The second setup prevents the outer side to move in radial direction.

[66] The newer function `EvaluateEnergyDensityAxi()` simplifies the code significantly.

element type	linear	quadratic	cubic
number of nodes	696	2685	5968
elastic energy U	0.12726	0.12118	0.12010
force $F = \frac{2U}{d}$	2545.2	2423.3	2402.1

Table 24: Comparison of different elements for the washer fastener example

- The third setup prevents both sides to move in radial direction.

This is implemented by switches at in the first section of the code. The parameter `Contour` contains the values of the flags indicating the boundary conditions on the vertical segments.

```
if 1 %% free sides
  Contour = [Ri H+D -22; Ri H -22;Ro-D 0 -21; Ro 0 -22; Ro D -22;Ri+D H+D -21];
elseif 0 %% clamped on the outside
  Contour = [Ri H+D -22; Ri H -22;Ro-D 0 -21; Ro 0 -12; Ro D -22;Ri+D H+D -21];
else  %% clamped on both sides
  Contour = [Ri H+D -12; Ri H -22;Ro-D 0 -21; Ro 0 -12; Ro D -22;Ri+D H+D -21];
endif
```

The results computed by the energy argument are

$$\begin{aligned} F &= 2402.1\,\text{N} && \text{with both sides free} \\ F &= 2880.4\,\text{N} && \text{with inner side free and outer side fixed} \\ F &= 3809.1\,\text{N} && \text{with both sides fixed} \end{aligned}$$

The additional constraints lead to a stiffer system, as expected.

9.49 A water dam

A water dam is deformed by its own weight and the water it should hold back. The code `WaterDam.m` below is a naive description of such a situation and generates the results in Figure 249.

- The code allows to select different shapes of the dam and different water levels. Modify the value of `Hwater` .
- By setting the density of the dam material to zero generate the stresses caused by the water only, i.e. use `rho = 0*2.4e3;` .
- Different shapes of the dam can be examined by selecting values of `CASE` .

WaterDam.m

```
global H Hwater BaseLeft
H = 30; Base = 0.7*H; Crest = 0.2*H; Hwater = 0.9*H; BaseLeft = H*0.2;
E = 20e9; nu = 0.2;
CASE = 3
switch CASE
case 1 %% no crack
  xy = [0,0,-11;Base,0,-22;Crest,H,-22;0,H,-33];
  x = [xy(:,1);xy(1,1)]; y = [xy(:,2);xy(1,2)];
case 2 %% with crack
  h = 0.1; depth = 1;
  xy = [0,0,-11;Base,0,-22;Crest,H,-22;0,H,-33;0,H/2+h,-22;depth,H/2,-22;0,H/2-h,-33];
  x = [xy(:,1);xy(1,1)]; y = [xy(:,2);xy(1,2)];
case 3  %% with foot
  BaseLeft = H*0.2;
  xy = [-BaseLeft,0,-11;Base,0,-22;Crest,H,-22;0,H,-33;0,BaseLeft,-33];
  x = [xy(:,1);xy(1,1)]; y = [xy(:,2);xy(1,2)];
case 4  %% with slope on both sides
  BaseLeft = H*0.2;
  xy = [-BaseLeft,0,-11;Base,0,-22;Crest,H,-22;0,H,-33];
```

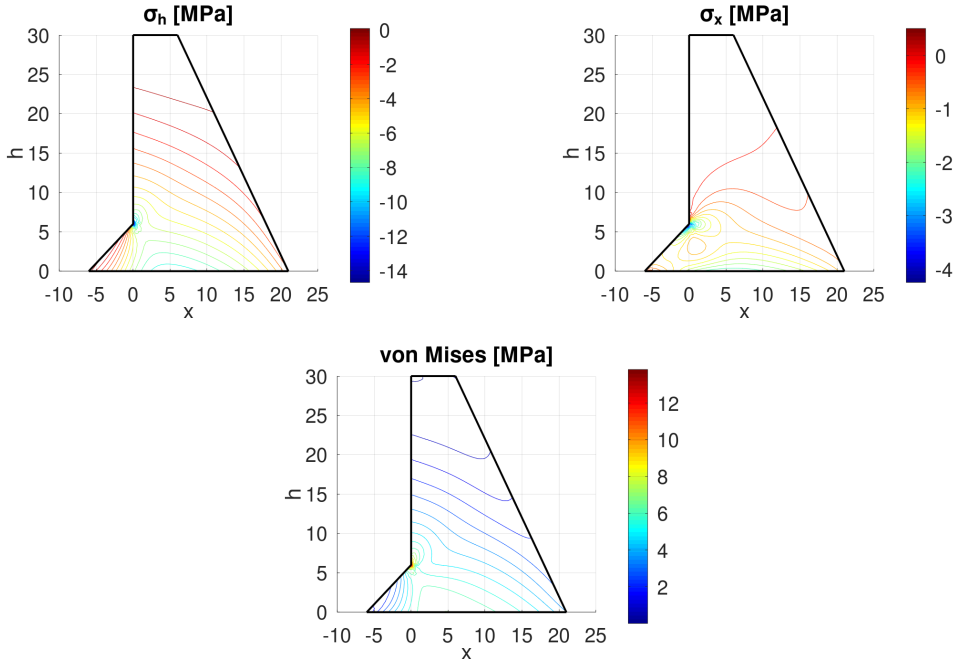


Figure 249: The normal stresses and the von Mises stress in a water dam

```
  x = [xy(:,1);xy(1,1)]; y = [xy(:,2);xy(1,2)];
endswitch

FEMmesh = CreateMeshTriangle('Dam',xy,1);
FEMmesh = MeshUpgrade(FEMmesh,'cubic');
figure(1); FEMtrimesh(FEMmesh); xlabel('x'); ylabel('h')

function res = f_dam(xy,dummy)
   global H BaseLeft
   rho = 2.4e3;
   res = -9.81*rho*(H-xy(:,2));
endfunction

switch CASE  %% different surface pressures
case {1,2}
 function res = px(xy,dummy)
   global Hwater
   res = +9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*(xy(:,2)<Hwater);
 endfunction
 function res = ph(xy,dummy)
   global Hwater
   res = +0*9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*(xy(:,2)<Hwater);
 endfunction

case 3
 function res = px(xy,dummy)
   global Hwater BaseLeft
   res = +9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*...
              ((xy(:,2)<Hwater).*(xy(:,2)>BaseLeft)+1/sqrt(2)*(xy(:,2)<=BaseLeft));
 endfunction
 function res = ph(xy,dummy)
   global Hwater BaseLeft
   res = +9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*(xy(:,2)<=BaseLeft)/sqrt(2);
 endfunction
```

```
  case 4
   function res = px(xy,dummy)
     global Hwater BaseLeft H
     alpha = atan(BaseLeft/H);
     res = +9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*...
                 ((xy(:,2)<Hwater).*(xy(:,2)>BaseLeft)+cos(alpha)*(xy(:,2)<=BaseLeft));
   endfunction
   function res = ph(xy,dummy)
     global Hwater BaseLeft H
     alpha = atan(BaseLeft/H);
     res = +9.81e3*(Hwater-xy(:,2)).*(xy(:,1)<eps).*(xy(:,2)<=BaseLeft)*sin(alpha);
   endfunction
  endswitch

  [u1,u2] = PlaneStrain(FEMmesh,E,nu,{0,'f_dam'},{0,0},{'px','ph'});
  [sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(FEMmesh,u1,u2,E,nu);
  vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy,sigma_z);

  figure(2);  FEMtrimesh(FEMmesh,u1);
              xlabel('x'); ylabel('h'); zlabel('u_x'); view([-120,20])
  figure(3);  FEMtrimesh(FEMmesh,u2);
              xlabel('x'); ylabel('h'); zlabel('u_h'); view([-120,20])
  figure(11); FEMtrimesh(FEMmesh,sigma_y*1e-6);
              xlabel('x'); ylabel('h'); zlabel('\sigma_h [MPa]'); view([60,20])
  figure(21); clf; FEMtricontour(FEMmesh,sigma_y*1e-6);
              xlabel('x'); ylabel('h'); colorbar();
              hold on; plot(x,y,'k'); title('\sigma_h [MPa]')
  figure(12); FEMtrimesh(FEMmesh,sigma_x*1e-6);
              xlabel('x'); ylabel('h'); zlabel('\sigma_x [MPa]'); view([60,20])
  figure(22); clf; FEMtricontour(FEMmesh,sigma_x*1e-6);
              xlabel('x'); ylabel('h'); colorbar()
              hold on; plot(x,y,'k'); title('\sigma_x [MPa]')
  figure(13); FEMtrimesh(FEMmesh,vonMises*1e-6);
              xlabel('x'); ylabel('h'); zlabel('von Mises [MPa]'); view([60,20])
  figure(23); clf; FEMtricontour(FEMmesh,vonMises*1e-6);
              xlabel('x'); ylabel('h'); colorbar()
              hold on; plot(x,y,'k'); title('von Mises [MPa]')
  Max_vonMises = max(vonMises)/1e6
```

9.50 A tuning fork

A tuning fork is used to generate a sound signal with a given, stable frequency. The sound is generated by an eigen mode of the fork. On the web page https://www.acs.psu.edu/drussell/Demos/TuningFork/fork-modes.html find a typical tuning fork and the most important eigen modes. The code `TuningFork.m` below allows to generate and analyze a few of the planar modes, leading to Figure 250.

- On the second line of the code you may change the dimensions of the tuning fork.
- With the value of `CASE` switch between different boundary conditions:

`CASE = 1` only the lower edge is fixed.

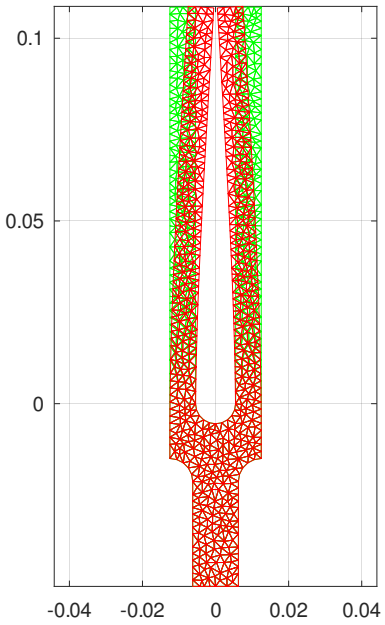
`CASE = 2` the lower edge and the two sides at the bottom are fixed.

- With the value of `MODE` select for which mode the deformation is analyzed and visualized.

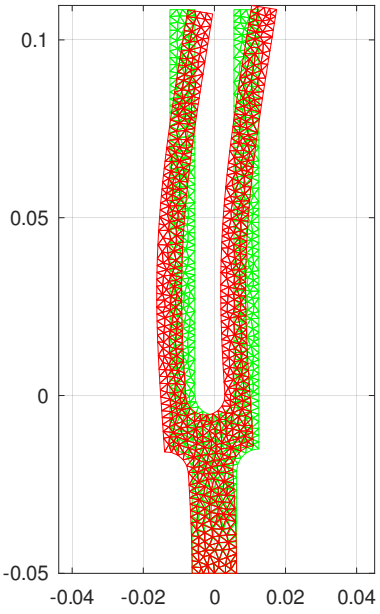
TuningFork.m

```
  angle = linspace(0,pi,11); halfangle = linspace(0,pi/2,6);
  R1 = 5.5; R2 = 6.25;  H1 = 108.5; H2 = H1+15;

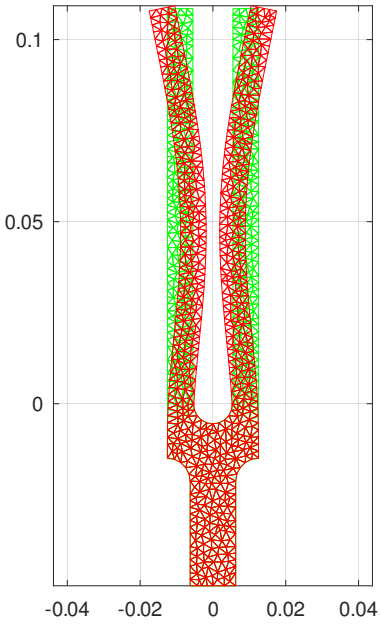
  x = [R1*cos(fliplr(angle)),R1,R1+7,R1+7-R2*sin(halfangle),R1+7-R2,-R1-7+R2,...
       -R1-7+R2*sin(fliplr(halfangle)),-R1-7,-R1]/1000;
  y = [-R1*sin(angle),H1,H1,H1-H2-R2+R2*cos(halfangle),-50,-50,...
```



(a) second mode $f = 426$ Hz



(b) third mode $f = 1603$ Hz



(c) fourth mode $f = 2613$ Hz

Figure 250: Three eigenmodes of a tuning fork

```
     H1-H2-R2+R2*cos(fliplr(halfangle)),H1,H1]/1000;
%%figure(1); plot(x,y); axis equal

CASE = 1;
IND = find(y==-50/1000,1);
switch CASE
case 1  %% only lower edge fixed
  xy = [x',y',-22*ones(length(x),1)]; xy(IND,3)=-11;
case 2  %% lower edge and sides fixed
  xy = [x',y',-22*ones(length(x),1)]; xy([IND-1:IND+1],3)=-11;
endswitch

FEMmesh = CreateMeshTriangle('Fork',xy,3e-6);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

E = 200e9; nu = 0.21; rho = 7.9e3;  %% use SI units
[la,u1all,u2all] = PlaneStressEig(FEMmesh,E,nu,rho,6);
Frequencies = sqrt(la')/(2*pi)

Mode = 2;
u1 = u1all(:,Mode); u2 = u2all(:,Mode);
MaxDisp = max(max(abs(u1)),max(abs(u2)));  %% scale the values
u1 = 0.005*u1/MaxDisp; u2 = 0.005*u2/MaxDisp;

figure(11); FEMtrimesh(FEMmesh,u1); xlabel('x'); ylabel('y'); zlabel('u_1')
figure(12); FEMtrimesh(FEMmesh,u2); xlabel('x'); ylabel('y'); zlabel('u_2')

[sigma_x,sigma_y,tau_xy] = EvaluateStress(FEMmesh,u1,u2,E,nu);
figure(21); FEMtrimesh(FEMmesh,sigma_x/1e6); xlabel('x'); ylabel('y'); zlabel('\sigma_x')
figure(22); FEMtrimesh(FEMmesh,sigma_y/1e6); xlabel('x'); ylabel('y'); zlabel('\sigma_y')
figure(23); FEMtrimesh(FEMmesh,tau_xy/1e6); xlabel('x'); ylabel('y'); zlabel('\tau_{xy}')
figure(30); ShowDeformation(FEMmesh,u1,u2,1); axis equal
-->
Frequencies =  321.78    425.64   1602.83   2612.85   4077.76   7000.23
```

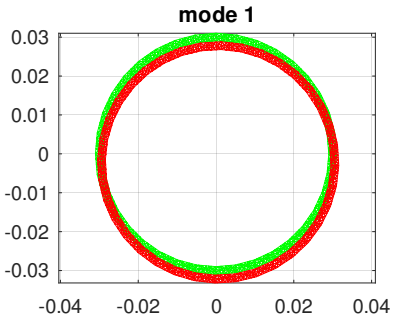
9.51 Vibrations of a ring

The planar eigen modes of a vibrationg ring can be evaluated with the help of `PlaneStressEig()`. The code in `RingVibration.m` evaluates the first twenty frequencies and the corresponding modes. Twelve modes are shown in Figure 251.

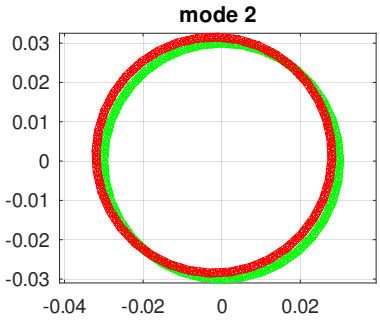
- The first 3 frequencies are (approximately) zero. They correspond to translations in horizontal and vertical direction and a pure rotation. Since no constraints are applied the ring is free to move, without deformation. See also Section 5.18 for the consequence of missing boundary constraints.
- Modes 4 and above always show up in pairs with the same frequency and rotated shape of the eigen modes. The 2 through 6 maximal and minimal deformations are clearly visible.
- For the computations in `RingVibration.m` quadratic elements are used. This avoids the effect of shear locking and increases the accuracy. Since only (approximately) 3 layer of elements are used, the effect of shear–locking is essential. If linear elements are used the resulting frequencies are considerably higher and the differences in the pairs is larger.
- The code in `RingVibration.m` uses the plane stress assumption, i.e. no forces in the z–direction. If the ring is supposed to be a section of a long circular tube, which can not move in z–direction, then use the plan strain assumption, i.e. the command `PlaneStrainEig()`. The frequencies will be slightly higher, since the setup is stiffer, $E^\star > E$ and $\nu^\star > \nu$.

RingVibration.m

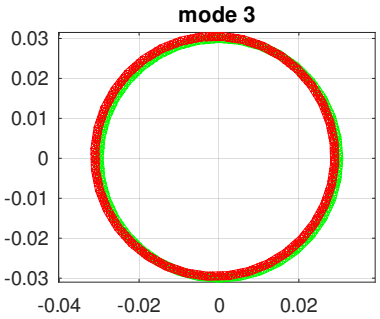
```
R = 0.03; D = 0.002;
Nring = 36; angles = linspace(0,2*pi,Nring+1)'; angles = angles(1:end-1);
Ring = [(R+D/2)*cos(angles),(R+D/2)*sin(angles),-22*ones(size(angles))];
```



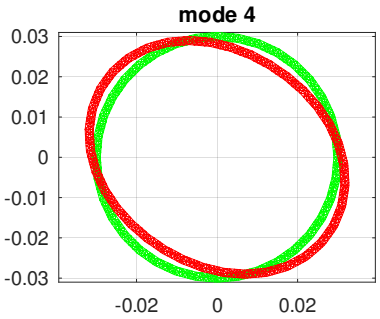
(a) $f = 0$ Hz



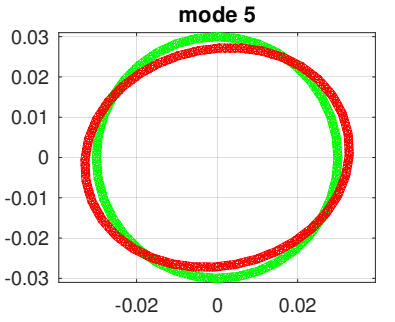
(b) $f = 0$ Hz



(c) $f = 0$ Hz



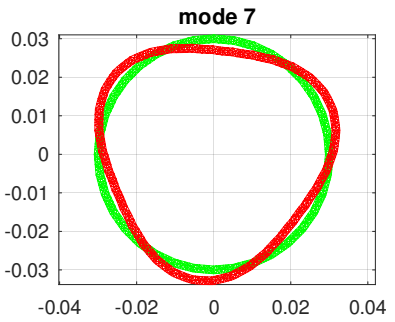
(d) $f = 1'368$ Hz



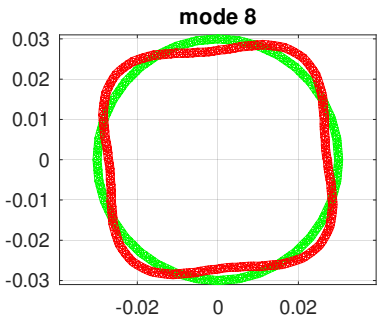
(e) $f = 1'368$ Hz



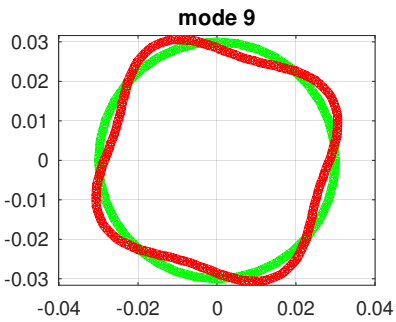
(f) $f = 3'856$ Hz



(g) $f = 3'856$ Hz



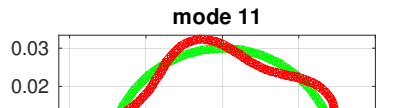
(h) $f = 7'357$ Hz



(i) $f = 7'387$ Hz



(j) $f = 11'825$ Hz





```
Hole.name   = 'Hole';
Hole.border = [(R-D/2)*cos(angles),(R-D/2)*sin(angles),-22*ones(size(angles))];
Hole.point  = [0,0.01];

FEMmesh = CreateMeshTriangle('Ring',Ring,5e-7,Hole);
FEMmesh = MeshUpgrade(FEMmesh,'quadratic')

E = 200e9; nu = 0.25; rho = 8e3; %% steel
Nmodes = 20;
[lambda,u1_all,u2_all] = PlaneStressEig(FEMmesh,E,nu,rho,Nmodes);
frequencies = sqrt(abs(lambda'))/(2*pi)

for Mode = 1:Nmodes
  u1 = u1_all(:,Mode); u2 = u2_all(:,Mode); scale = 3e-3/(max(abs([u1;u2])));
  figure(10+Mode); ShowDeformation(FEMmesh,u1,u2,scale); axis equal;
                   title(sprintf('mode %i',Mode))
endfor
-->
Frequencies =   3.7599e-3   6.9476e-3   8.9547e-3   1.3678e+3   1.3678e+3   3.8556e+3
                3.8556e+3   7.3570e+3   7.3571e+3   1.1825e+4   1.1825e+4   1.7219e+4
```

Figure 251 might indicate that the deformation of the ring is strictly in radial direction, but this is not the case. For each of the modes there is a movement u_r in radial direction, combined with a movement u_ϕ in angular direction. At a point $R\,(\cos(\phi), \sin(\phi))$ on the center line of the ring determine the radial and angular defection by

$$u_r(\phi) = \cos(\phi)\,u_1(\phi) + \sin(\phi)\,u_2(\phi) \quad \text{and} \quad u_\phi(\phi) = -\sin(\phi)\,u_1(\phi) + \cos(\phi)\,u_2(\phi)\ .$$

There is a strong interaction between the radial and angular movements[67]. Evaluate these radial and angular deformations along the center line $(x, y) = R\ \cos(\phi), \sin(\phi))$ and use a linear regression of the form

$$u_r(\phi) = c_1 + c_2\ \cos(n\,\phi) + c_3\ \sin(n\,\phi) \quad \text{and} \quad u_\phi(\phi) = d_1 + d_2\ \cos(n\,\phi) + d_3\ \sin(n\,\phi)\ .$$

In the code `RingVibrationMode.m` below $n = 3$ is used for mode 7, since there are 3 outer arcs visible in Figure 251. The resulting numbers and Figure 252 confirm that the deformations are strictly trigonometric, FEM data and best fit can not be distinguished.

- For `Mode = 7` the numerical results show that the phase difference between the radial and angular movement is given by $-90°$. A point on the center line of the ring moves according to

$$\begin{pmatrix} u_r(t,\phi) \\ u_\phi(t,\phi) \end{pmatrix} = A\ \cos(\sqrt{\lambda}\,t) \begin{pmatrix} \phi_r(\phi) \\ u_\phi(\phi) \end{pmatrix}\ ,$$

  i.e. oscillations along a straight line. For this mode at an angle ϕ_0 of a point halfway between the maximal amplitude for u_r and zero amplitude (i.e. an offset of $15° = \frac{360°}{3\cdot 8}$ of a maximal radial deflection in Figures 251 or 252) find

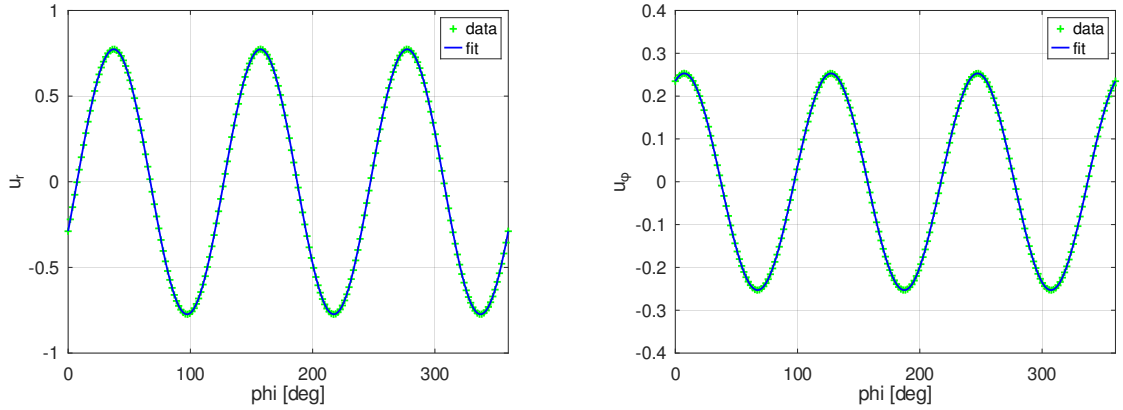
$$\begin{pmatrix} u_r(t,\phi_0) \\ u_\phi(t,\phi_0) \end{pmatrix} \approx \frac{A}{\sqrt{2}}\ \cos(\sqrt{\lambda}\,t) \begin{pmatrix} 0.77 \\ 0.25 \end{pmatrix}\ .$$

  A similar result is correct for (almost) all modes, with different values for n.

- Modes 1, 2 and 3 are of the form $c_1\ \cos(\phi) + c_2\ \sin(\phi)$ and correspond to pure translations and rotations.

- Mode 16 is special: it a purely radial displacement, i.e. $u_\phi \approx 0$. Modes 19 and 20 are superpositions of purely radial displacement and $c_1\ \cos(\phi)+c_2\ \sin(\phi)$. These special modes are the reason for the `if...elseif...endif` construction in the first lines of `RingVibrationMode.m`.

[67] For this reason one can not use the results for transversal displacements of a straight slender beam with periodic boundary conditions to determine the frequencies and modes of the vibration ring. Find results on the ring in [Blev79, p. 205] and the derivation in [Soed04].



(a) radial displacement u_r (b) angular displacement u_ϕ

Figure 252: Radial displacement u_r and angular displacement u_ϕ for mode 7. Shown are the FEM data and the best fit of the form $c_1 + c_2\ \cos(n\,\phi) + c_3\ \sin(n\,\phi)$ with $n = 3$.

RingVibrationMode.m

```
%% this code requires that RingVibration was run first
Mode = 7
if     Mode<16   n = floor(Mode/2)
elseif Mode<19   n = floor((Mode-1)/2)
else             n = floor((Mode - 17)/2)  endif
phi = linspace(0,2*pi,200)'; x = R*cos(phi); y = R*sin(phi);
u1_r = FEMgriddata(FEMmesh,u1_all(:,Mode),x,y);
u2_r = FEMgriddata(FEMmesh,u2_all(:,Mode),x,y);
ur = cos(phi).*u1_r + sin(phi).*u2_r;
uphi = -sin(phi).*u1_r + cos(phi).*u2_r;

M = [ones(size(phi)),cos(n*phi),sin(n*phi)];  p = LinearRegression(M,ur);
dr = p(1); Amp_r = sqrt(p(2)^2+p(3)^2); phase_r = atan2(p(2),p(3))/pi*180;
disp(sprintf('Results for u_r:   dr   = %g, amplitude = %g, phase_r   = %g',dr,Amp_r,phase_r))
ur_fit = M*p;
figure(101); plot(phi*180/pi,ur,'+g',phi/pi*180,ur_fit,'b'); xlim([0,360])
             xlabel('phi [deg]'); ylabel('u_r'); legend('data','fit');

p = LinearRegression(M,uphi);
dphi = p(1); Amp_phi = sqrt(p(2)^2+p(3)^2);  phase_phi = atan2(p(2),p(3))/pi*180;
disp(sprintf('Results for u_phi: dphi = %g, amplitude = %g, phase_phi =% g',...
                                 dphi,Amp_phi,phase_phi))
disp(sprintf('Ratio of amplitudes = %g, phaseDiff = %g',Amp_phi/Amp_r,phase_r-phase_phi))
uphi_fit = M*p;
figure(102); plot(phi*180/pi,uphi,'+g',phi/pi*180,uphi_fit,'b');  xlim([0,360])
             xlabel('phi [deg]'); ylabel('u_\phi'); legend('data','fit');
-->
Mode = 7
n = 3
Results for u_r:   dr   = -5.61863e-07, amplitude = 0.773988, phase_r   =  158.09
Results for u_phi: dphi = -3.14111e-07, amplitude = 0.253133, phase_phi = -111.91
Ratio of amplitudes = 0.327049, phaseDiff = 270
```

9.52 Hertz contact of a rigid cylinder with an elastic half space

There is a very nice web page about `OpenHertz` at foadsf.github.io/OpenHertz/. It allows to examine different Hertz contact setups.

In this section the contact of a rigid cylinder with a half space is examined and compared to result for the Hertz contact theory.

9.52.1 The model and the algorithm

A rigid, horizontal cylinder with radius R is pushed downwards by D into an elastic half space. The goal is to determine the resulting deformation and stresses. This contact problem is a nonlinear problem and thus requires some additional work if analyzed with the help of `FEMoctave`. The domain of contact is not know a–priori and thus has to be determined by an appropriate iteration algorithm.

symbol		units
R	radius of cylinder	mm
D	penetration depth	mm
x	horizontal coordinate	mm
y	vertical coordinate	mm
a	half width of the contact area	mm
P	total pressure per length	N/mm
p	local pressure	N/mm^2
E	Young's modulus of elasticity	N/mm^2
ν	Poisson's ratio	
W	width of the computational domain	mm
H	height of the computational domain	mm

Table 25: Parameters for the contact of a cylinder with a half space

The algorithm is based on the observation, that the normal stress σ_y or the vertical displacement u_2 are known along the upper edge of the half space at $y = 0$. In the contact zone $-a \le x \le +a$ the displacement of the plane is given by the equation of the displaced cylinder.

$$\begin{aligned} R^2 &= x^2 + (y - R + D)^2 \\ y &= R - D - \sqrt{R^2 - x^2} \approx R - D - R + \frac{x^2}{2\,R} = -D + \frac{x^2}{2\,R} \end{aligned}$$

As consequence use the boundary conditions

$$\begin{cases} u_2(x) = R - D - \sqrt{R^2 - x^2} & \text{for} \quad |x| \le a \\ \sigma_y = 0 & \text{for} \quad |x| \ge a \\ \sigma_x = 0 & \text{for} \quad |x| \le W \end{cases}$$

along the upper edge $y = 0$. Due to the obvious symmetry only half of the physical domain has to be used for the computational model, leading to the computational domain $0 \le x \le W$ and $-H \le y \le 0$. On the other sections of the boundary use

$$\begin{cases} u_1 = 0 \quad , \quad \sigma_y = 0 & \text{along} \quad x = 0 \\ u_1 = 0 \quad , \quad \sigma_y = 0 & \text{along} \quad x = W \\ u_1 = u_2 = 0 & \text{along} \quad y = -H \end{cases}$$

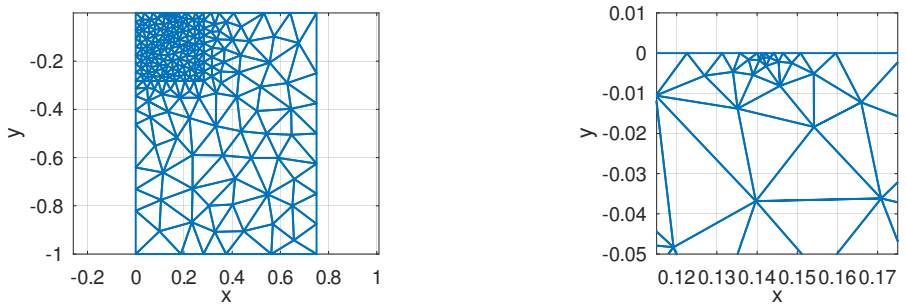
In the domain a plane strain model is used with the material parameters E and ν.

- To start the iteration the intersection of the cylinder with the plane $y = 0$ is used.

  $$y = 0 = R - D - \sqrt{R^2 - x^2} \quad \Longrightarrow \quad a_0^2 = x^2 = R^2 - (R - D)^2 = 2\,R\,D - D^2 = D\,(2\,R - D)$$

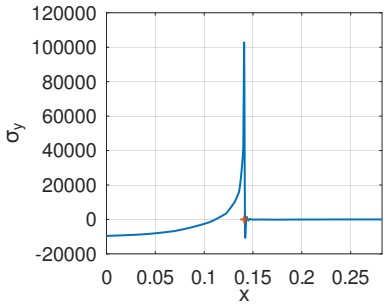
  The initial value of a_0 will be updated to a by each iteration.

- A triangular mesh is generated with smaller triangles in the corner $[0 \le x \le 2\,a_0] \times [-2\,a_0 \le y \le 0]$. By adding points close to $[a, 0]$ an even finer mesh is used in the critical area of the contact point at $x = +a$. Find the original mesh and an enlarged critical area in Figure 253.



(a) the full domain

(b) zoom into critical area



(c) σ_y for the first attempt

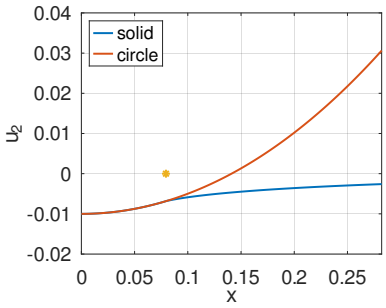
Figure 253: The mesh for the cylinder contact problem

- For each iteration solve the plain strain problem with the help of the command `PlaneStrain()`. Then evaluate the normal stress $\sigma_y(x)$ along the upper edge $y = 0$ and determine the first zero at $x = a$ of $\sigma_y(x)$. For $x > a$ the normal stress σ_y is positive and thus the cylinder and the half plane will not be in contact, see Figure 253(c).
- The above computations are repeated and the changing values of a have to be observed. After 5 iterations the value of a changes only slightly and a good approximation of the domain of contact $-a \le x \le +a$ is determined. The displacement u_2 along the upper edge at $y = 0$ and the circle are shown in Figure 254(a).
- By integrating the normal stress σ_y along different heights y verify that the vertical pressure P (units N/mm) is independent on the height y.
- Compare the obtained results for the contact (half) width a and the pressure P with the theoretical results from Section 9.52.3 below. The obviously missing factor 2 for the pressure is caused by the symmetry, i.e. only half the setup is used for the FEM computations.

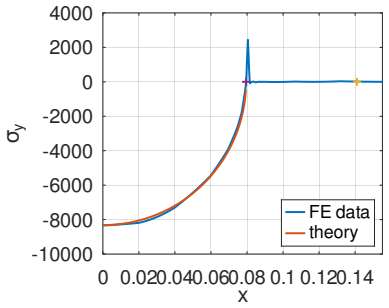
To start define the necessary auxiliary functions and generate the initial mesh. A mesh with cubic elements leads to a smoother solution.

HertzCylinder.m

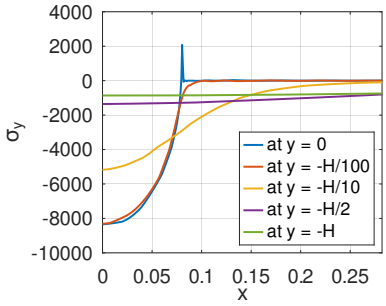
```
PHASE = 1  %% 1: setup and computation
           %% 2: visualization of the result
           %% 3: parametric study for small penetration depth D
           %% 4: parametric study for large penetration depth D
switch PHASE
case 1  %% PHASE 1
clear *
pkg load femoctave
E = 200e3; nu = 0.24 ;%% N/mm^2  parameter for steel
global R D
R = 1; D = 0.01;          %% radius of cylinder and indentation depth
W = 0.75; H = 1;          %% width and height of the computational domain
a0 = sqrt(D)*sqrt(2*R-D)  %% first estimate of contact point
```



(a) u_y along upper edge



(b) σ_y along upper edge, theory and FEM



(c) final σ_y at different levels

Figure 254: The vertical displacement u_2 along the upper edge and the normal stress σ_y at different levels

```
Area = 0.1^2; MeshType = 'cubic';  %% definition of the mesh
Seg1.name = 'Segment';
Seg1.border = [0,-2*a0,0;2*a0,-2*a0,0;2*a0,0,0];
Point1.name = 'MeshSize';   Point1.where = [+a0,-a0];  Point1.area = Area/20;
%% finer mesh arounf origin
Point2.name = 'MeshSize'; Point2.where = [+2*a0,-2*a0]; Point2.area = Area;

dd = 0.001;   %% very fine mesh at contact point
Mesh = CreateMeshTriangle('Flat',[0,0,-21;a0-dd,0,-21;a0,0,-22;a0+dd,0,-22;...
                        W,0,-12;W,-H,-11;0,-H,-12], Area,Seg1,Point1,Point2);

function res = disp_y(xy)   %% displacement in y-direction as function of x
  global R D
  a0 = sqrt(D)*sqrt(2*R-D);
  res = (R-D-sqrt(R^2 - xy(:,1).^2));
  res = res.*(xy(:,1)<=a0).*(xy(:,2)>-eps);
endfunction

function res = FindFirstPositive(x,sigma_y);%% find the first zero of the normal stress
  ind = find(sigma_y>0,1);
  x0 = x(ind-1); dx = x(ind)-x0;
  y0 = sigma_y(ind-1); y1 = sigma_y(ind);
  res = x0 -y0*dx/(y1-y0);
endfunction

figure(1); FEMtrimesh(Mesh); axis equal; xlabel('x'); ylabel('y')
           xlim(0.145+[-0.03,0.03]);ylim([-0.05,0.01])
Mesh = MeshUpgrade(Mesh,MeshType);
```

Then solve the first plane strain problem and start the iteration.

HertzCylinder.m

```
[u1,u2] = PlaneStrain(Mesh,E,nu,{0,0},{0,'disp_y'},{0,0});

[sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(Mesh,u1,u2,E,nu);
x_edge = linspace(0,2*a0,1000)';
sigma_y_edge = FEMgriddata(Mesh,sigma_y,x_edge,0*x_edge);
figure(30); plot(x_edge,sigma_y_edge,a0,0,'+');
            xlabel('x'); ylabel('\sigma_y'); xlim([0,max(x_edge)])
a = FindFirstPositive(x_edge,sigma_y_edge)

for jj = 1:5   %% use 5 iterations and observe the value of a
  Mesh = CreateMeshTriangle('Flat',
         [0,0,-21;a-dd,0,-21;a,0,-22;a+dd,0,-22;W,0,-12;W,-H,-11;0,-H,-12],...
          Area,Seg1,Point1,Point2);
  Mesh = MeshUpgrade(Mesh,MeshType);
  [u1,u2] = PlaneStrain(Mesh,E,nu,{0,0},{0,'disp_y'},{0,0});
  [sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(Mesh,u1,u2,E,nu);
  sigma_y_edge = FEMgriddata(Mesh,sigma_y,x_edge,0*x_edge);
  a = FindFirstPositive(x_edge,sigma_y_edge)
endfor

%% display the normal stress as function of x along the upper edge
ind = find(x_edge<a);
sigma_y_fit = mean(sigma_y_edge(ind))*4/pi*sqrt(1-x_edge(ind).^2/a^2);
figure(31); plot(x_edge,sigma_y_edge,x_edge(ind),sigma_y_fit,a0,0,'+',a,0,'+');
            xlabel('x'); ylabel('\sigma_y'); legend('FE data','theory',...
            'location','southeast'); xlim([0,1.1*a0])
VonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy,sigma_z);

%% display the displacement of the upper edge
y_edge = FEMgriddata(Mesh,u2,x_edge,0*x_edge);
y_circle = R-D-sqrt(R^2-x_edge.^2);
figure(20); plot(x_edge,y_edge,x_edge,y_circle,a,0,'*'); xlabel('x'); ylabel('u_2');
            legend('solid','circle', 'location','northwest'); xlim([0,max(x_edge)])

x_edge_low = linspace(0,W,1000)';
sigma_y_edge_low  = FEMgriddata(Mesh,sigma_y,x_edge_low,-H*ones(size(x_edge_low)));
sigma_y_edge_top  = FEMgriddata(Mesh,sigma_y,x_edge_low,zeros(size(x_edge_low)));
sigma_y_edge_half = FEMgriddata(Mesh,sigma_y,x_edge_low,-H/2*ones(size(x_edge_low)));
sigma_y_edge_10   = FEMgriddata(Mesh,sigma_y,x_edge_low,-H/10*ones(size(x_edge_low)));
sigma_y_edge_100  = FEMgriddata(Mesh,sigma_y,x_edge_low,-H/100*ones(size(x_edge_low)));
figure(90); plot(x_edge_low,sigma_y_edge_top,x_edge_low,sigma_y_edge_100,...
                 x_edge_low,sigma_y_edge_10,x_edge_low,sigma_y_edge_half,...
                 x_edge_low,sigma_y_edge_low);
            xlabel('x'); ylabel('\sigma_y'); xlim([0,max(x_edge)])
            legend('at y = 0','at y = -H/100','at y = -H/10','at y = -H/2','at y = -H',...
            'location','southeast')

PressureUpperEdgeLocal = trapz(x_edge,sigma_y_edge)
PressureUpperEdge = trapz(x_edge_low,sigma_y_edge_top)
Pressure100Edge = trapz(x_edge_low,sigma_y_edge_100)
Pressure10Edge = trapz(x_edge_low,sigma_y_edge_10)
PressureHalfEdge  = trapz(x_edge_low,sigma_y_edge_half)
PressureLowerEdge = trapz(x_edge_low,sigma_y_edge_low)

Estar = E/(1-nu^2);
P = -2*PressureLowerEdge;
a = sqrt(4*P*R/(pi*Estar))
-->
PHASE = 1
a0 = 0.1411
a = 0.1115
a = 0.095575
a = 0.087056
```

```
a = 0.082939
a = 0.080529
a = 0.079477
PressureUpperEdgeLocal = -516.16
PressureUpperEdge = -515.46
Pressure100Edge = -512.26
Pressure10Edge = -514.30
PressureHalfEdge = -513.27
PressureLowerEdge = -514.44
a = 0.078567
```

9.52.2 Evaluation and visual results

The above results have to be visualized.

- A grid of the domain $[0, 3\,a] \times [-3\,a, 0]$ with the sizable effects is generated, the deformation is given by u_1 and u_2.
- On this grid u_1, u_2 and the total displacement $\sqrt{u_1^2 + u_2^2}$ are evaluated and visualized, leading to Figure 255.
- The normal stresses σ_y and σ_x and the von Mises stress are evaluated on the same grid and visualized, leading to Figure 256.

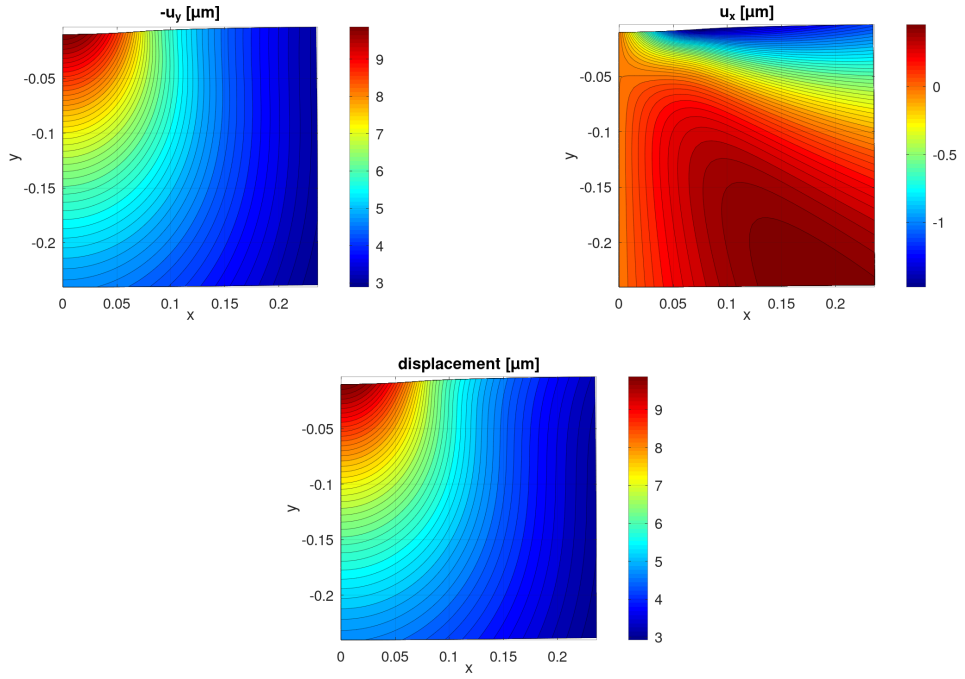


Figure 255: Contact with cylinder: contours for the displacements u_2 and u_1 and $\sqrt{u_1^2 + u_2^2}$

HertzCylinder.m

```
case 2  %% PHASE 2
[x,y] = meshgrid(linspace(0,3*a,51),linspace(-3*a,0,51));
u1g = FEMgriddata(Mesh,u1,x,y); u2g = FEMgriddata(Mesh,u2,x,y);
x_g = x + u1g;  y_g = y + u2g;              %% construct the deformed grid
sigma_yg = FEMgriddata(Mesh,sigma_y,x,y);   %% evaluate sigma_y on the  grid
figure(101); mesh(x_g,y_g,sigma_yg); xlabel('x'); ylabel('y');zlabel('\sigma_y')
figure(111); contourf(x_g,y_g,sigma_yg/1e3,linspace(min(sigma_yg(:)),0,51)/1e3);
             xlabel('x'); ylabel('y'); title('\sigma_y [kPa]'); colorbar
figure(121); contourf(x_g,y_g,-u2g*1e3,51); xlabel('x'); ylabel('y');
```

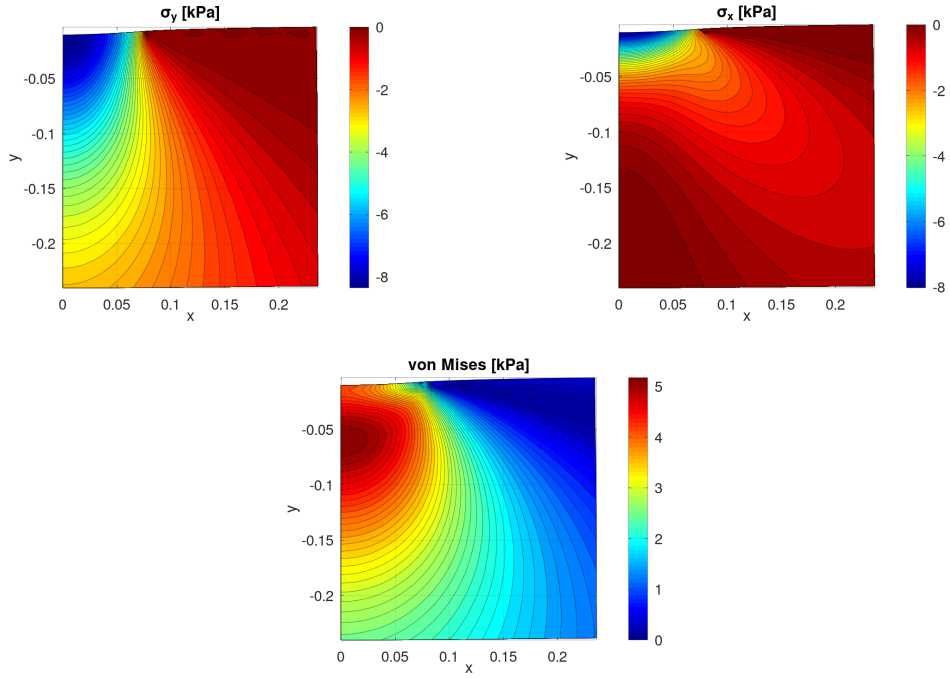


Figure 256: Contact with cylinder: contours for the two normal stresses and the von Mises stress

```
            title('-u_y [\mum]'); colorbar
sigma_xg = FEMgriddata(Mesh,sigma_x,x,y);
figure(102); mesh(x_g,y_g,sigma_xg); xlabel('x'); ylabel('y');zlabel('\sigma_x')
figure(112); contourf(x_g,y_g,sigma_xg/1e3,linspace(min(sigma_yg(:)),0,51)/1e3);
             xlabel('x'); ylabel('y'); title('\sigma_x [kPa]'); colorbar
figure(122); contourf(x_g,y_g,u1g*1e3,51); xlabel('x'); ylabel('y');
             title('u_x [\mum]'); colorbar

VonMises_g = FEMgriddata(Mesh,VonMises,x,y);
figure(103); mesh(x,y,VonMises_g); xlabel('x'); ylabel('y');zlabel('von Mises')
figure(113); contourf(x_g,y_g,VonMises_g/1e3,linspace(0,max(VonMises_g(:)),51)/1e3);
             xlabel('x'); ylabel('y'); title('von Mises [kPa]'); colorbar
figure(123); contourf(x_g,y_g,sqrt(u1g.^2+u2g.^2)*1e3,51);
             xlabel('x'); ylabel('y'); title('displacement [\mum]'); colorbar
DOriginal = D; aOriginal = a;
```

9.52.3 The analytical solution based on the Hertz theory

The results in this section are found in [Barb18], which is based on [John87a]. The goal is to present the results of the Hertz contact for a line load by a cylinder on a half space. The notation is adapted to match the above sections.

- Use [Barb18, p. 14]. Apply a point load P at the origin to a half space $y < 0$. The shearing stresses are $\tau_{xy} = \tau_{yz} = 0$. Solve $\Delta\phi = 0$ for the potential ϕ and then along the surface $y = 0$ find

$$\sigma_y = -\frac{\partial\,\phi}{\partial y} \quad \text{and} \quad u_2 = -\frac{2\,(1-\nu^2)\,P}{E}\,\frac{\partial\,\phi}{\partial y} = -\frac{2\,P}{E^\star}\,\frac{\partial\,\phi}{\partial y}\,.$$

  With $r^2 = x^2 + y^2 + z^2$ obtain the fundamental solution

$$\phi(R,y) = -\frac{P}{2\,\pi}\,\ln(r+y) = -\frac{P}{2\,\pi}\,\ln(\sqrt{x^2+y^2+z^2}+y)\,.$$

  This leads to the displacement at the surface $y = 0$

$$u_2(x,0,z) = -\frac{P}{\pi\,E^\star\,\sqrt{x^2+z^2}}\,.$$

- For a line load p along $x = 0$ for $-b < z < b$ obtain

$$u_2(x,0,0) = -\frac{p}{\pi\, E^\star} \int_{z=-b}^{+b} \frac{1}{\sqrt{x^2+z^2}}\, dz = \ldots = -\frac{p}{\pi\, E^\star}\, \ln|z+\sqrt{z^2+z^2}|\,\Big|_{z=-b}^{+b}\,.$$

This expression does not converge as $b \to +\infty$. This might be the reason why I did not find an analytic formula for the displacement D as function of the applied load!

- In the area of contact $-a < x < a$ the slope of the displacement has to coincide with the slope of the circle. This fact can be used to extract information on the Hertz solution. For the circle obtain

$$\frac{\partial}{\partial x}\, u_2(x) = \frac{d}{\partial x}\left(-D + \frac{x^2}{2\,R}\right) = \frac{x}{R}\,.$$

For a point load p evaluate the slope of the upper edge.

$$u_2(x,0,z) = -\frac{p}{\pi\, E^\star}\, \frac{1}{\sqrt{x^2+z^2}} \qquad \Longrightarrow \qquad \frac{\partial}{\partial x}\, u_2(x,0,z) = -\frac{p}{\pi\, E^\star}\, \frac{x}{\sqrt{x^2+z^2}^3}$$

For a constant load density p for $-b < z < +b$ conclude

$$\begin{aligned}
\frac{\partial}{\partial x}\, u_2(x,0,z) &= -\frac{p}{\pi\, E^\star} \int_{z=-b}^{+b} \frac{x}{\sqrt{x^2+z^2}^3}\, dz \\
&= -\frac{p}{\pi\, E^\star}\, \frac{z}{x\,\sqrt{x^2+z^2}}\,\Big|_{z=-b}^{+b} = -\frac{p}{\pi\, E^\star}\, \frac{2\,b}{x\,\sqrt{x^2+b^2}} \to -\frac{p}{\pi\, E^\star}\, \frac{2}{x} \quad \text{as} \quad b \to +\infty
\end{aligned}$$

Now evaluate the slope generated by the applied pressure density $p(x)$ for $-a < x < +a$ and set it equal to the slope of the circle.

$$\frac{x}{R} = \frac{2}{\pi\, E^\star} \int_{x=-a}^{+a} \frac{p(s)}{x-s}\, ds$$

A solution of this singular integral equation is given in the Appendix C of [Barb18].

$$\begin{aligned}
P &= \int_{-a}^{+a} p(x)\, dx \\
p(x) &= \frac{1}{\pi\,\sqrt{a^2-x^2}} \left(P - \frac{E^\star}{2} \int_{-a}^{+a} \frac{\xi\,\sqrt{a^2-\xi^2}}{R\,(x-\xi)}\, d\xi\right)
\end{aligned}$$

- At $x = +a$ (or $-a$) use $p(a) = 0$ to conclude

$$\begin{aligned}
0 &= P - \frac{E^\star}{2} \int_{-a}^{+a} \frac{\xi\,\sqrt{a^2-\xi^2}}{R\,(\pm a-\xi)}\, d\xi \\
P &= \frac{E^\star}{2} \int_{-a}^{+a} \frac{\xi\,\sqrt{a^2-\xi^2}}{R\,(+a-\xi)}\, d\xi = \frac{E^\star}{2} \int_{-a}^{+a} \frac{\xi\,\sqrt{(a-\xi)(a+\xi)}}{R\,(+a-\xi)}\, d\xi \\
&= \frac{E^\star}{2} \int_{-a}^{+a} \frac{\xi\,\sqrt{a+\xi}}{R\,\sqrt{a-\xi}}\, d\xi = \frac{E^\star}{2}\, \frac{\pi\, a^2}{2\,R} = \frac{E}{4\,(1-\nu^2)}\, \frac{\pi\, a^2}{R}
\end{aligned}$$

This equation can be solved for the half width a of the contact area.

$$a^2 = \frac{4\,P\,R\,(1-\nu^2)}{\pi\, E} \quad \text{and} \quad \frac{P}{a^2} = \frac{\pi\, E}{4\,R\,(1-\nu^2)} \tag{142}$$

These results are confirmed by the above FEM values.

The above analytical computaion also leads to

$$p(x) = \frac{E^\star}{2\,R}\,\sqrt{a^2-x^2} = \frac{E}{2\,R\,(1-\nu^2)}\,\sqrt{a^2-x^2} = \frac{2\,P}{\pi\, a^2}\,\sqrt{a^2-x^2} = \frac{2\,P}{\pi\, a}\,\sqrt{1-\frac{x^2}{a^2}}\,.$$

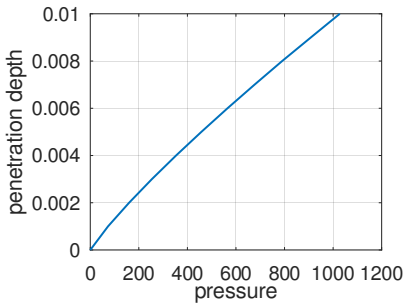
Figure 254(b) illustrates this result with the help of the FEM simulation. Use the formula for $p(x)$ to conclude

$$\begin{aligned}
p_{max} &= p(0) = \frac{E\, a}{2\,R\,(1-\nu^2)} = \frac{2\,P}{\pi\, a} \\
p_{mean} &= \frac{P}{2\,a} = \frac{\pi\, E\, a}{8\,R\,(1-\nu^2)} = \frac{\pi}{4}\, p_{max}\,.
\end{aligned}$$

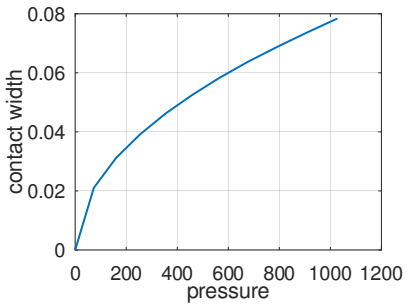
9.52.4 Parameter studies for different penetration depths

Starting with the above numerical results the penetration depth D can be diminished step by step and the new contact width a and pressure P be evaluated with a loop over different values of D. Instead of a fixed number of iterations a simple termination criterion is used, stop if a does not change too much any more. Find the results in Figure 257. Surprisingly the penetration depth is not too far from a linear function of the applied pressure.

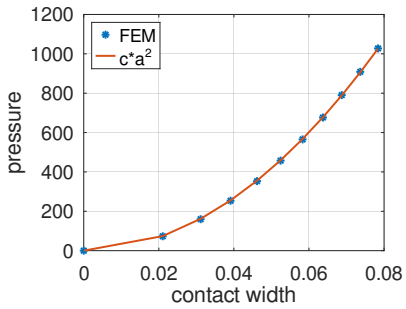
Based on equation (142) expected that $P = c\,a^2$ for a constant $c = \frac{\pi E}{4\,R\,(1-\nu^2)}$. This is confirmed with the help of a linear regression, leading to Figure 257(c).



(a) depth as function of pressure



(b) contact width as function of pressure



(c) pressure as function of contact width

Figure 257: Contact with cylinder: results of pressure and contact width for small penetration depth

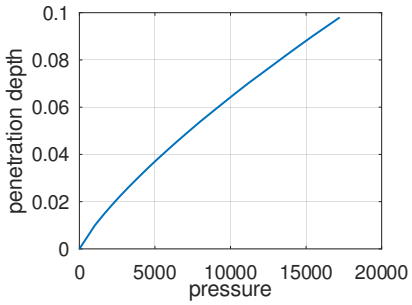
HertzCylinder.m

```
case 3  %% Phase 3: parametric study for small D
a = aOriginal;
D_List = DOriginal*[1:-0.1:0.1]';
a_List = zeros(size(D_List)); Pressure_List = a_List;
a_List(1) = aOriginal; Pressure_List(1) = -2*PressureLowerEdge;
for ii = 1:length(D_List)
  D = D_List(ii);
  disp(sprintf('working with penetration depth D = %g',D))
  jj = 0;
  do
    jj++;
    Mesh = CreateMeshTriangle('Flat',
           [0,0,-21;a-dd,0,-21;a,0,-22;a+dd,0,-22;W,0,-12;W,-H,-11;0,-H,-12],...
            Area,Seg1,Point1,Point2);
    Mesh = MeshUpgrade(Mesh,MeshType);
    [u1,u2] = PlaneStrain(Mesh,E,nu,{0,0},{0,'disp_y'},{0,0});
    [sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(Mesh,u1,u2,E,nu);
    sigma_y_edge = FEMgriddata(Mesh,sigma_y,x_edge,0*x_edge);
    a_old = a;
    a = FindFirstPositive(x_edge,sigma_y_edge);
    disp(sprintf('value a = %g, last change = %g', a , a-a_old))
  until or(abs(a-a_old)< 5e-5,jj>20)
  a_List(ii) = a;
```

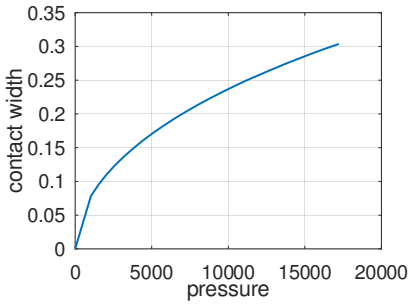
```
  sigma_y_edge_low  = FEMgriddata(Mesh,sigma_y,x_edge_low,-H*ones(size(x_edge_low)));
  PressureLowerEdge = trapz(x_edge_low,sigma_y_edge_low);
  Pressure_List(ii) = -2 * PressureLowerEdge;
endfor
D_List = [D_List;0]; Pressure_List = [Pressure_List;0];  a_List = [a_List;0];
figure(201); plot(Pressure_List,D_List); xlabel('pressure'); ylabel('penetration depth')
figure(202); plot(Pressure_List,a_List); xlabel('pressure'); ylabel('contact width')

p = LinearRegression(a_List.^2,Pressure_List);
figure(203); plot(a_List,Pressure_List,'*',a_List,p*a_List.^2)
             ylabel('pressure'); xlabel('contact width')
             legend('FEM','c*a^2','location','northwest')
```

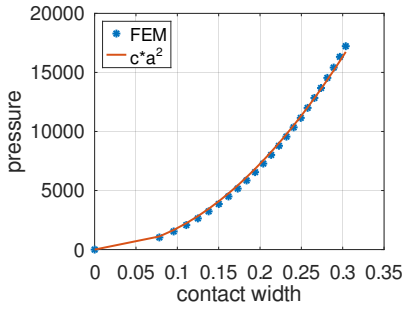
Instead of using smaller penetration depth, larger ones can be examined too. The tools do not change. The results in Figure 258 show no surprises.



(a) depth as function of pressure



(b) contact width as function of pressure



(c) pressure as function of contact width

Figure 258: Contact with cylinder: results of pressure and contact width for large penetration depth

HertzCylinder.m

```
case 4  %% Phase 4: parametric study for large D
a = aOriginal;
D_List = DOriginal*[1:0.4:10]';
a_List = zeros(size(D_List)); Pressure_List = a_List;

for ii = 1:length(D_List)
  D = D_List(ii);
  disp(sprintf('working with penetration depth D = %g',D))
  x_edge = linspace(0,1.3*a,1000');
  jj = 0;
  do
  jj++;
  Mesh = CreateMeshTriangle('Flat',
           [0,0,-21;a-dd,0,-21;a,0,-22;a+dd,0,-22;W,0,-12;W,-H,-11;0,-H,-12],...
            Area,Seg1,Point1,Point2);
```

```
    Mesh = MeshUpgrade(Mesh,MeshType);
    [u1,u2] = PlaneStrain(Mesh,E,nu,{0,0},{0,'disp_y'},{0,0});
    [sigma_x,sigma_y,tau_xy,sigma_z] = EvaluateStress(Mesh,u1,u2,E,nu);
    sigma_y_edge = FEMgriddata(Mesh,sigma_y,x_edge,0*x_edge);
    a_old = a;
    a = FindFirstPositive(x_edge,sigma_y_edge);
    disp(sprintf('iteration %i, value a = %g, last change = %g', jj, a, a-a_old))
  until or(abs(a-a_old)< 5e-5,jj>20)
  a_List(ii) = a;
  sigma_y_edge_low  = FEMgriddata(Mesh,sigma_y,x_edge_low,-H*ones(size(x_edge_low)));
  PressureLowerEdge = trapz(x_edge_low,sigma_y_edge_low);
  Pressure_List(ii) = -2 * PressureLowerEdge;
endfor

D_List = [0;D_List]; Pressure_List = [0;Pressure_List];  a_List = [0;a_List];
figure(301); plot(Pressure_List,D_List); xlabel('pressure'); ylabel('penetration depth')
figure(302); plot(Pressure_List,a_List); xlabel('pressure'); ylabel('contact width')

p = LinearRegression(a_List.^2,Pressure_List);
figure(303); plot(a_List,Pressure_List,'*',a_List,p*a_List.^2)
             ylabel('pressure'); xlabel('contact width')
             legend('FEM','c*a^2','location','northwest')
```

9.53 Hertz contact of a rigid sphere with an elastic half space

The above situation of a cylinder pressing into a half space can be modified to model the situation of a sphere pressing into a half space. The algorithm does not change substantially. Find the adaptation in Table 26. The Hertz based

cylinder	$\longleftrightarrow$	sphere
x, y	$\longleftrightarrow$	r, z
`PlaneStress()`	$\longleftrightarrow$	`AxiStress()`
`EvaluateStress()`	$\longleftrightarrow$	`EvaluateStressAxi()`
`EvaluateVonMises()`	$\longleftrightarrow$	`EvaluateVonMisesAxi()`

Table 26: Adaptations for the contact problem of a plane with with a cylinder or a sphere

formulas change when pressing with a sphere with radius a. Find in [John87a, p. 93]

$$
\begin{aligned}
a &= \left(\frac{3\,P\,R}{4\,E^\star}\right)^{1/3} \quad \text{and} \quad P = \frac{4\,E^\star}{3\,R}\,a^3 \\
D &= \left(\frac{3\,P}{4\,E^\star}\right)^{2/3}\frac{1}{R^{1/3}} = \left(\frac{3\,P\,R}{4\,E^\star}\right)^{2/3}\frac{1}{R} = \frac{a^2}{R} = \left(\frac{3}{4\,E^\star}\right)^{2/3}\frac{1}{R^{1/3}}\,P^{2/3} \\
P &= \frac{4\,E^\star}{3}\,R^{1/2}\,D^{3/2} \\
p(r) &= p_{max}\sqrt{1-\frac{r^2}{a^2}} \quad \text{for} \quad 0 \le r \le a \quad , \quad p_{max} = \frac{3}{2}\,p_{mean} = \frac{3\,P}{2\,\pi\,a^2}
\end{aligned}
$$

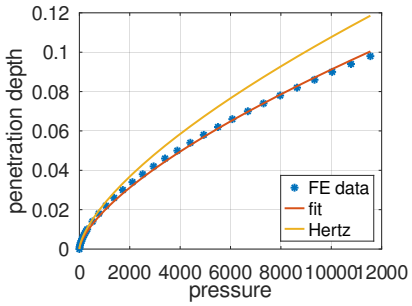
The above code `HertzCylinder.m` is adapted to `HertzSphere.m`, not shown in these notes, but included in the distribution of `FEMoctave`.

- In Figure 259 the relations between the penetration depth, the resulting pressure and the radius of the contact are are shown, together with the theoretical results based on the Hertz theory. In addition a linear regression was used to determine that $0.8469 \cdot D_{Hertz}$ is the best match to the overall FEM data. Observe that for small penetration depth D the Hertz result

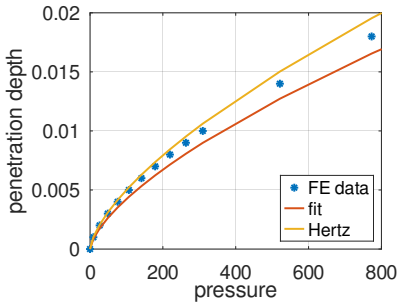
$$
P = \frac{4\,E^\star\,\sqrt{R}}{3}\,D_{Hertz}^{3/2} \quad \text{or} \quad D_{Hertz} = \left(\frac{3\,P}{4\,E^\star}\right)^{2/3}\frac{1}{R^{1/3}}
$$

is close to the result generated by `FEMoctave`, but for large values of D there is a systematic difference. For the same penetration depth the required pressure by the Hertz approach is smaller than for the FEM simulation. Thus for the same pressure Hertz will arrive at a larger penetration depth. For large values of D the fitted curve is a better match.

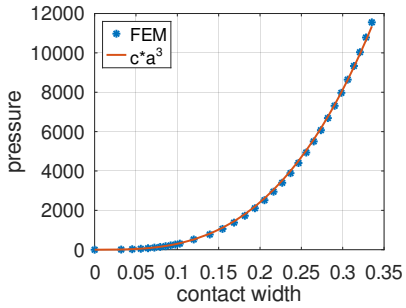
- Figure 259(c) confirms that the total pressure P is proportional to a^3, the radius of the area of contact.
- In Figure 260 find the displacement in radial and z direction. In Figure 261 the normal stresses and the von Mises stress are shown.



(a) depth as function of pressure



(b) zoom for small pressure



(c) pressure as function of contact width

Figure 259: Contact with sphere: results of pressure and contact width for large penetration depth

For a given penetration depth D we can use the Hertz results in [John87a] and determine the surface pressure

$$\sigma_z(r,0) = \begin{cases} -\frac{2\,E^\star}{\pi\,a^2} R^{1/3} D^{3/2} \sqrt{1-\frac{r^2}{a^2}} & \text{for } r \le a \\ 0 & \text{for } r \ge a \end{cases}$$

and then use `FEMoctave` to determine the resulting deformations and stresses caused by the sphere indenting the half space. The other boundary conditions used by `FEMoctave` are

$$\begin{cases} \sigma_r = 0 & & & \text{along} \quad z = 0 \\ u_r = 0 & , & \sigma_y = 0 & \text{along} \quad r = W \\ u_r = 0 & , & u_z = 0 & \text{along} \quad z = -H \\ u_r = 0 & , & \sigma_y = 0 & \text{along} \quad r = 0 \end{cases}$$

The code `HertzSphereDirect.m` shown below generates Figure 262.

- Figure 262(a) shows the resulting displacement and the expected circular displacement by Hertz's theory. The two coincide quite well, as expected. Hertz predicts a slightly larger penetration depth, which is consistent with the observations in Figure 259.
- Figure 262(b) shows the normal stress σ_z in vertical direction. The result coincides with Figure 261.

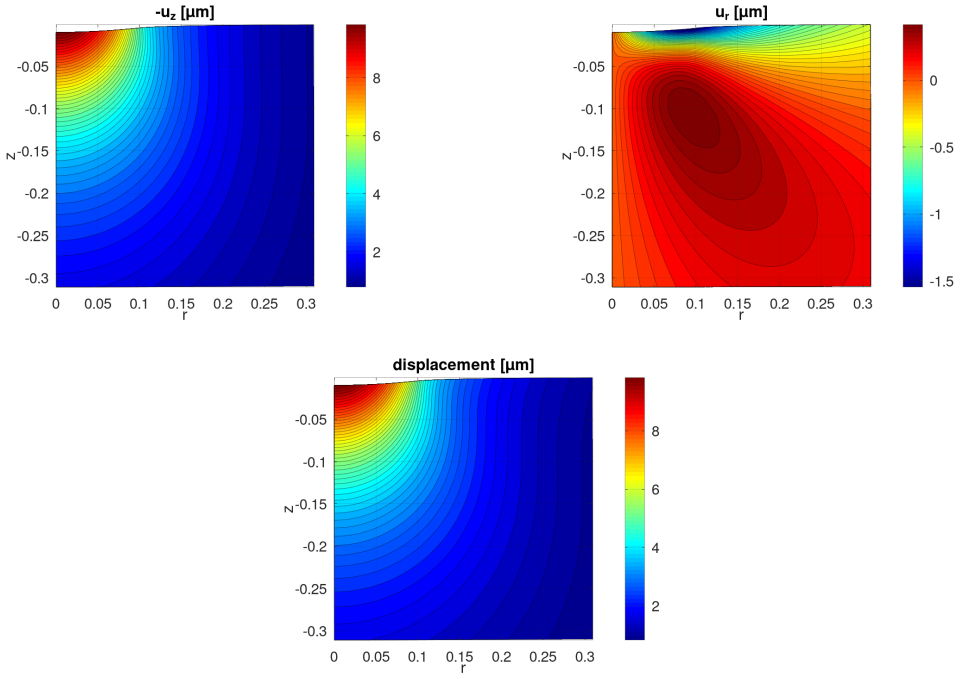


Figure 260: Contact with sphere: contours for the displacements u_z and u_r and $\sqrt{u_r^2 + u_z^2}$

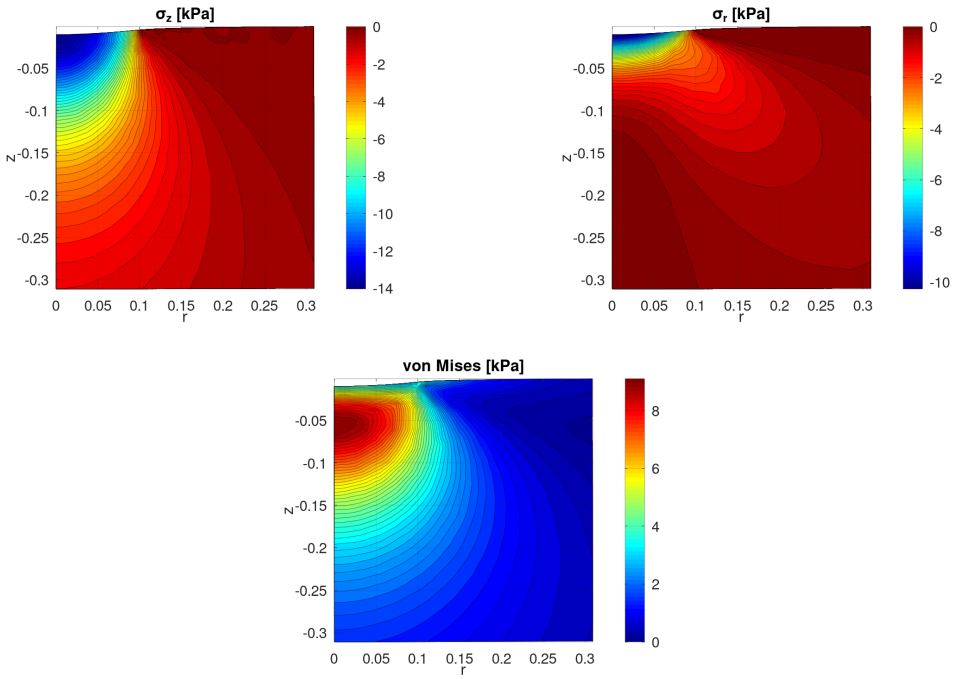
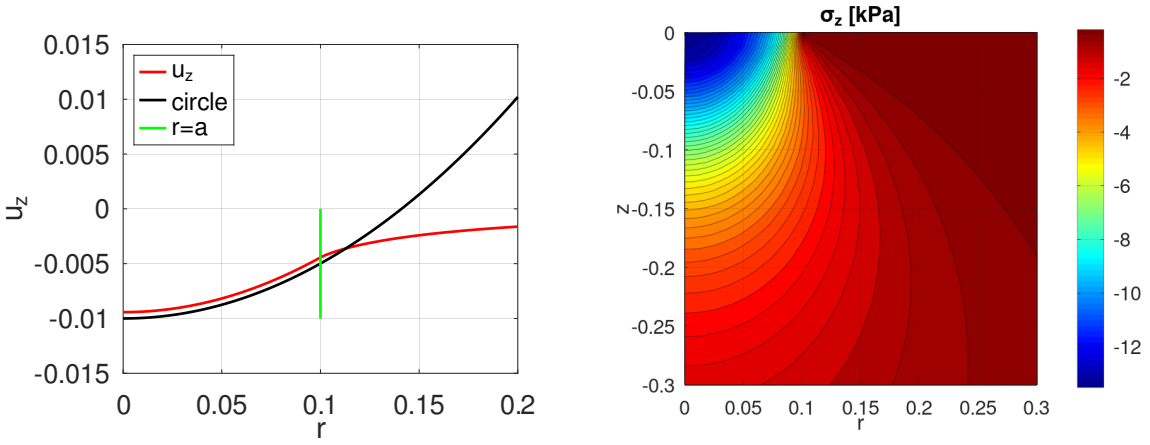


Figure 261: Contact with sphere: contours for the two normal stresses and the von Mises stress

- In Figure 263 find the von Mises stress for this setup. Maximal value at $r = 0$, $z \approx -0.046$ for a radius $a = 0.1$ of the area of contact. This is consistent with the information in [Gold01, p.80], where the maximal stress is predicted at $z \approx -0.48 \cdot a$.



(a) the displacement $u_z(r, 0)$ (b) the normal pressure σ_z

Figure 262: Contact with a sphere for given normal pressure $\sigma_z(r, 0)$

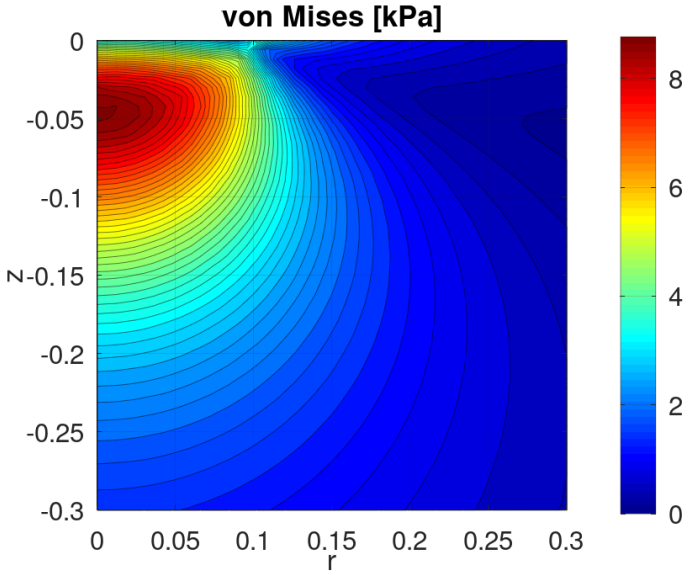


Figure 263: The von Mises stress for a contact with a sphere for given normal pressure $\sigma_z(r, 0)$

HertzSphereDirect.m

```
%% 1 GPa = 10^9 N/m^2 = 10^3 MPa = 10^3 N/mm^2
E = 200e3; nu = 0.24 ;%% N/mm^2
%% parameters for steel: E = 200 GPa = 200e3 MPa, nu = 0.24, yield strength = 300 MPa
%% parameters for gold:  E =  79 GPa =  79e3 MPa, nu = 0.42, yield strength = 200 MPa
R = 1; D = 0.01;    %% radius of sphere and indentation depth
W = 0.75; H = 1;    %% width and height of the computational domain

global a P
Estar = E/(1-nu^2);
P = 4*Estar/3*sqrt(R)*D^(3/2)
```

```
a = (3*P*R/(4*Estar))^(1/3)
D = (3*P/(4*Estar))^(2/3)/R^(1/3);

function res = Load(rz)
  global a P
  r = rz(:,1);
  res = -3*P/(2*pi*a^2)*sqrt(1-r.^2/a^2).*(r<a);
endfunction

N = 51;
if 0  %% mesh by rectangle
  Mesh = CreateMeshRect(linspace(0,W,N),linspace(-H,0,N),-11,-23,-12,-12);
else  %% mesh by triangle
  area = 0.5*W*H/N^2;
  dd = 0.001;
  Mesh = CreateMeshTriangle('Flat',[0,0,-23;a-dd,0,-23;a,0,-22;a+dd,0,-22;...
                                    W,0,-12;W,-H,-11;0,-H,-12],area);
endif
Mesh = MeshUpgrade(Mesh,'quadratic');
%%Mesh = MeshUpgrade(Mesh,'cubic');
[ur,uz] = AxiStress(Mesh,E,nu,{0,0},{0,0},{0,'Load'});

figure(1); FEMtrimesh(Mesh,ur); xlabel('r'); ylabel('z'); zlabel('u_r')
figure(2); FEMtrimesh(Mesh,uz); xlabel('r'); ylabel('z'); zlabel('u_z')
r = linspace(0,2*a,1000)';
uz_edge = FEMgriddata(Mesh,uz,r,zeros(size(r)));
circle = R-D-sqrt(R^2-r.^2);
figure(11); plot(r,uz_edge,'r',r,circle,'k',[a,a],[-D,0],'g')
            xlabel('r'); ylabel('u_z'); xlim([0,2*a]);
            legend('u_z','circle','r=a','location','northwest')
DifferenceUzAtOrigin = circle(1)-uz_edge(1)

[sigma_r,sigma_y,sigma_z,tau_xz] = EvaluateStressAxi(Mesh,ur,uz,E,nu);
VonMises = EvaluateVonMisesAxi(sigma_r,sigma_y,sigma_z,tau_xz);

NN = 51;
[rr,zz] = meshgrid(linspace(0,3*a,NN),linspace(-3*a,0,NN));
sigma_zg   = FEMgriddata(Mesh,sigma_z, rr,zz);
sigma_rg   = FEMgriddata(Mesh,sigma_r, rr,zz);
VonMises_g = FEMgriddata(Mesh,VonMises,rr,zz);
figure(31); contourf(rr,zz,sigma_rg/1e3,51); xlabel('r'); ylabel('z');
            title('\sigma_r [kPa]'); axis equal; colorbar
figure(32); contourf(rr,zz,sigma_zg/1e3,51); xlabel('r'); ylabel('z');
            title('\sigma_z [kPa]'); axis equal; colorbar
figure(33); contourf(rr,zz,VonMises_g/1e3,51); xlabel('r'); ylabel('z');
            title('von Mises [kPa]'); axis equal; colorbar
```

9.54 Elastic waves in solids

In this section three initial boundary value problems for dynamic elasticity are examined. The effect of a localized initial displacement is visualized. Elastic waves are generated by each of the examples.

9.54.1 A cylindrical elastic wave

On a square domain $(x, y) \in [-5, +5] \times [-5, +5]$ an initial horizontal displacement is prescribed by

$$u_1(x,y) = \begin{cases} 0.1 \cdot \cos^2(4\,r) & \text{for} \quad 0 \le r \le \frac{\pi}{8} \approx 0.4 \\ 0 & \text{for} \quad \frac{\pi}{8} \le r \end{cases} ,$$

where $r = \sqrt{x^2 + y^2}$. The vertical displacements u_2 and the initial velocities are set to zero. On all of the boundaries zeros displacements are enforced. Then the solutions are generated at times $t = 1.0, 2.0, 3.0$ and 4.0. The code below leads to Figures 264 and 265.

- The maximal speed of the elastic waves is $c = \sqrt{\frac{E}{\rho}} = 1$ and for $r = \sqrt{x^2+y^2} > \frac{\pi}{8} \approx 0.4$ the initial displacements are zero. Thus the displacements for $r > c\,t + 0.4$ vanish. Expect the highest amplitudes at $r = c\,t$.
- In the right columns in Figures 264 and 265 the scale if changing for the different times. Thus use the values indicated on the axes to evaluate the speed of the waves. The scaling in the left columns remains constant.
- The initial displacement is strictly horizontal and longitudinal waves move faster than transversal waves. This leads to the elliptical shape of the horizontal displacement u_1 for small times, i.e. Figure 264(b). The transversal speed $c_{trans} = \frac{c}{\sqrt{2}} \approx 0.7\,c$ is confirmed by the figure.
- Initially the vertical displacements u_2 are considerably smaller than the horizontal displacements u_1.

ElasticWaveCylinder.m

```
E = 1; nu = 0; rho = 1; f = {0,0}; gD = {0,0}; gN = {0,0}; L = 10; H = L;
function res = u0Func(xy)
  r = sqrt(xy(:,1).^2+xy(:,2).^2);
  res = 0.1*cos(4*r).^2.*(4*r<pi/2);
endfunction
u0 = {'u0Func',0}; v0 = {0,0}; t0 = 0; tend = 4; steps = [4,100];

N = 100; %% the graphs in the notes are generated with N = 200, but the CPU time ...
Mesh = CreateMeshRect(linspace(-L/2,+L/2,N+1),linspace(-H/2,+H/2,N+1),-11,-11,-11,-11);
Mesh = MeshUpgrade(Mesh,'quadratic');
[u1_all,u2_all,t] = PlaneStressDynamic(Mesh,E,nu,rho,f,gD,gN,u0,v0,t0,tend,steps);

Amp = 0.02; Levels = Amp*[-1:0.1:1]; Levels(11)=[]; %% drop Levels = 0
for jj = 2:length(t)
  u1 = u1_all(:,jj); u2 = u2_all(:,jj);
  figure(20+jj); FEMtrimesh(Mesh,u1); zlim(Amp*[-1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_1');
  figure(30+jj); FEMtrimesh(Mesh,u2); zlim(Amp*[-1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_2');
  figure(40+jj); clf; FEMtricontour(Mesh,u1,Levels); axis equal;
                 xlabel('x'); ylabel('y'); title('u_1');
  figure(50+jj); clf; FEMtricontour(Mesh,u2,Levels); axis equal
                 xlabel('x'); ylabel('y'); title('u_2')
endfor
```

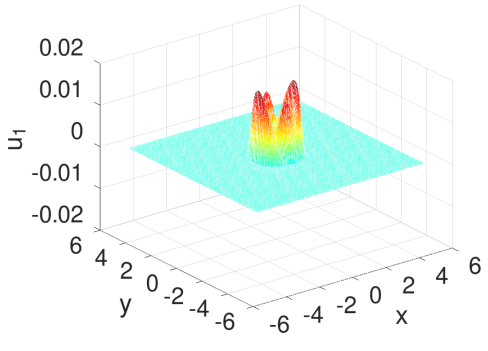
9.54.2 A planar elastic wave in a canal

On a domain snown on the right in Figure 266 an initial horizontal displacement is prescribed by

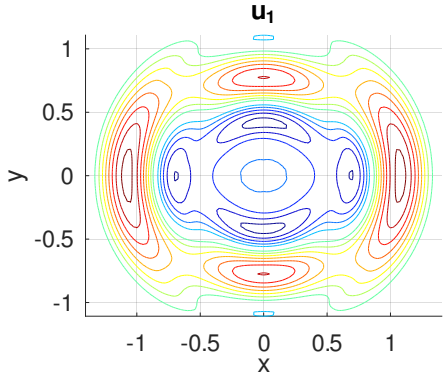
$$u_1(x,y) = \begin{cases} 0.1 \cdot \cos^2(8\,(x+1)) & \text{for} \quad 0 \le (x+1) \le \frac{\pi}{16} \approx 0.2 \\ 0 & \text{for} \quad \frac{\pi}{16} \le (x+1) \end{cases} .$$

The vertical displacements u_2 and the initial velocities are set to zero. On all of the boundaries are force free. Only on the edge on the right zeros displacements are enforced. Then the solutions are generated at times $t = 1.0, 2.0, 3.0$ and 4.0. The code below leads to Figures 266 and 267.

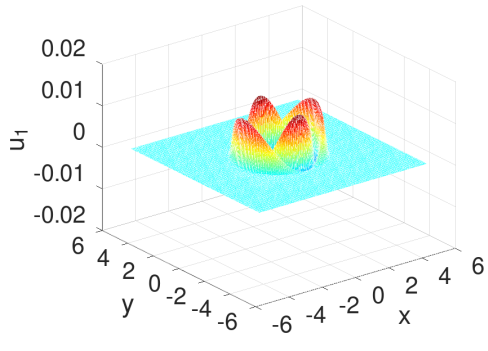
- With the parameters $E = 1$, $\nu = 0$ and $\rho = 1$ the solution is a single pulse moving form $x \approx -1$ to the right with the speed $c = 1$ for the longitudinal elastic wave. This would be the case if the canal had a constant width. This observation is confirmed by Figure 266(b) at time $t = 1$. Up to this time the width of the canal was constant. Only at to upper edge the first, minor effect of the changing width is visible.
- At time $t = 2$ in Figure 266(d) the longitudinal pulse is at $x \approx 1$ with its shape and size mostly intact. The upper part is clearly modified by the narrower canal. In the section $x < 1$ the transversal elastic wave generated by the upper wall is clearly visible.
- At time $t = 3$ in Figure 266(f) the longitudinal pulse is at $x \approx 2$ with its shape and size mostly intact. The transvesal wave moved down with speed $c_{trans} = \frac{c}{\sqrt{2}} \approx 0.7$.



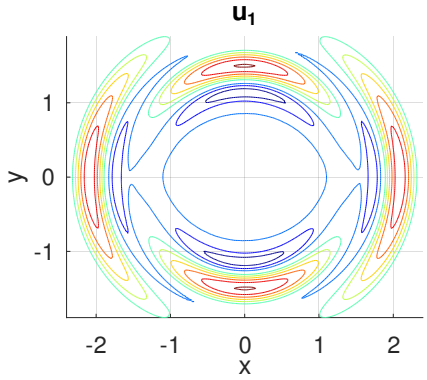
(a) graph of u_1 at $t = 1.0$



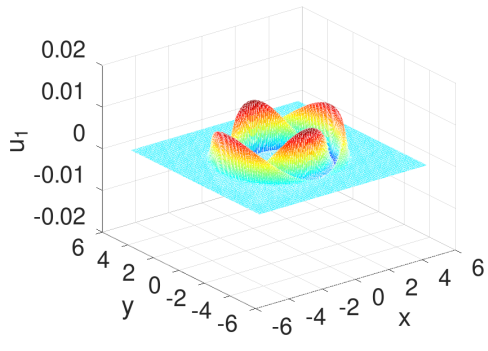
(b) contours of u_1 at $t = 1.0$



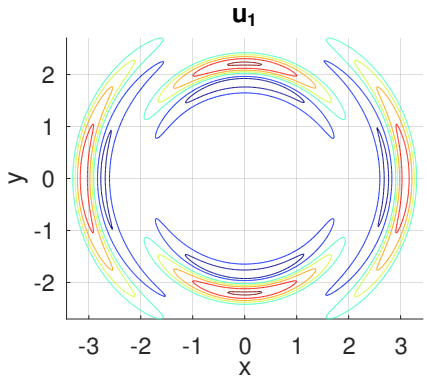
(c) graph of u_1 at $t = 2.0$



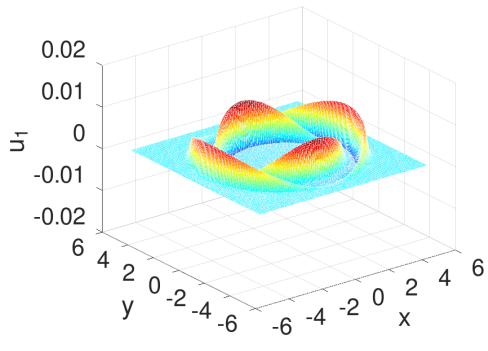
(d) contours of u_1 at $t = 2.0$



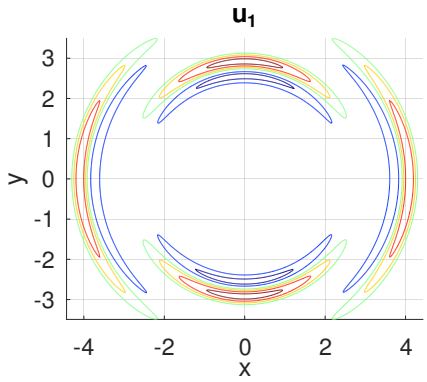
(e) graph of u_1 at $t = 3.0$



(f) contours of u_1 at $t = 3.0$

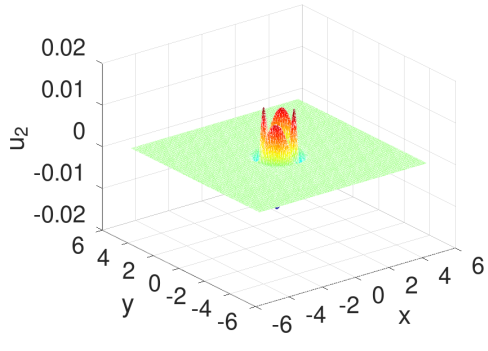


(g) graph of u_1 at $t = 4.0$

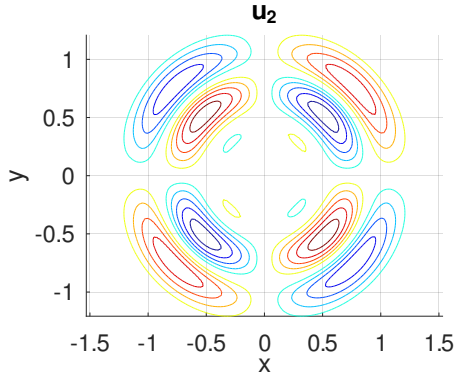


(h) contours of u_1 at $t = 4.0$

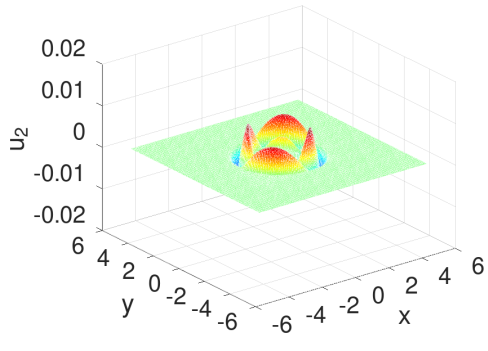
Figure 264: The horizontal displacement u_1 of an elastic cylindrical wave



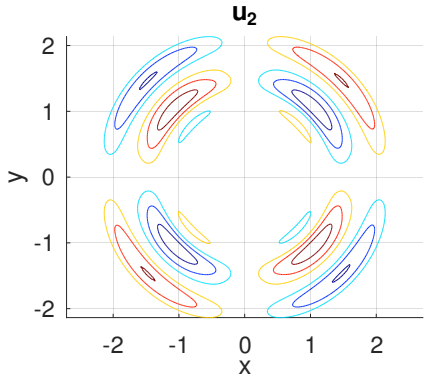
(a) graph of u_2 at $t = 1.0$



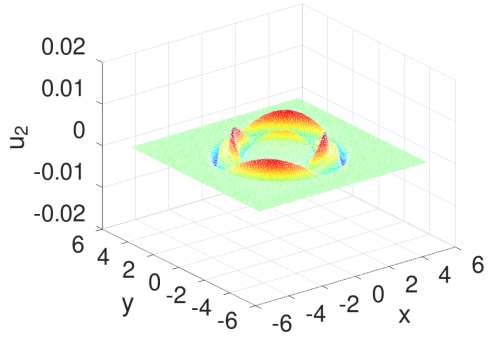
(b) contours of u_2 at $t = 1.0$



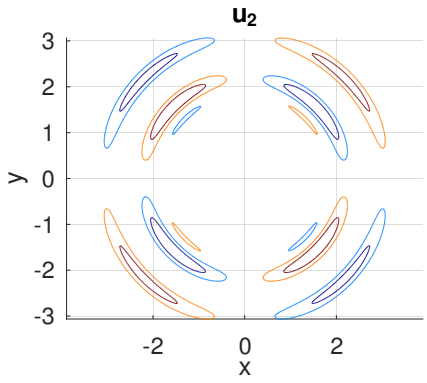
(c) graph of u_2 at $t = 2.0$



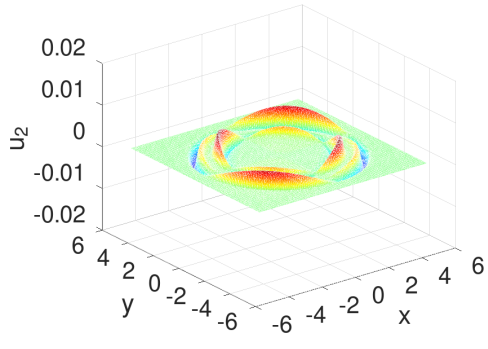
(d) contours of u_2 at $t = 2.0$



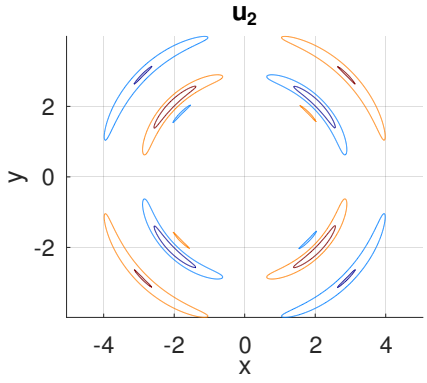
(e) graph of u_2 at $t = 3.0$



(f) contours of u_2 at $t = 3.0$



(g) graph of u_2 at $t = 4.0$



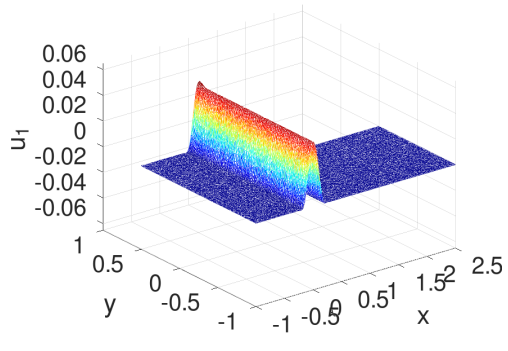
(h) contours of u_2 at $t = 4.0$

Figure 265: The vertical displacement u_2 of an elastic cylindrical wave

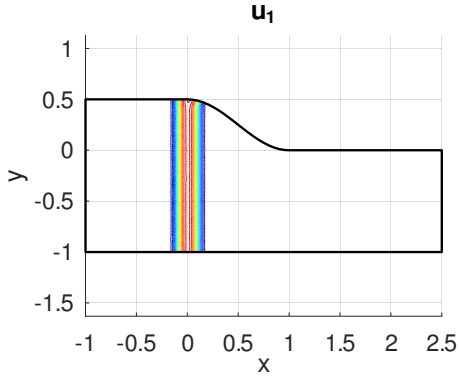
- At time $t = 4$ in Figure 266(h) the longitudinal pulse is at $x \approx 2$. It was reflected at the hard wall at $x = 2.5$ and is moving back with negative amplitude.
- In Figure 267 the vertical displacement u_2 is shown. At time $t = 1$ no result is shown, since $u_2 \approx 0$.

WaveCanal.m

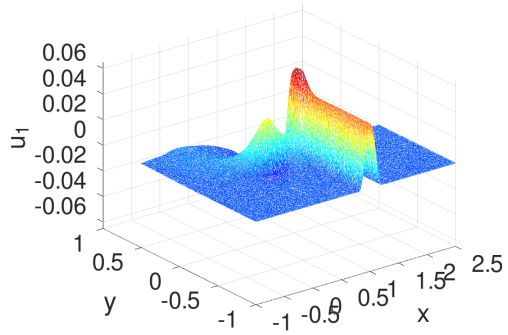
```
x = linspace(0,1,21)'; y = cos(pi/2*x).^2; bc = -22*ones(size(x));
MeshData = [-1,0.5,-22;x,0.5*y,bc;2.5,0,-11;2.5,-1,-22;-1,-1,-22];
x = MeshData(:,1); y = MeshData(:,2);
Mesh = CreateMeshTriangle('Canal',MeshData,0.001);
Mesh = MeshUpgrade(Mesh,'quadratic');
E = 1; nu = 0; rho = 1; f = {0,0}; gD = {0,0}; gN = {0,0};
function res = u0Func(xy)
  res = 0.1*cos(8*(xy(:,1)+1)).^2.*(8*(xy(:,1)+1)<pi/2);
endfunction
u0 = {'u0Func',0}; v0 = {0,0}; t0 = 0; tend = 4; steps = [4,200];
[u1_all,u2_all,t] = PlaneStressDynamic(Mesh,E,nu,rho,f,gD,gN,u0,v0,t0,tend,steps);
Amp = 0.065; Levels = Amp*[-1:0.1/2:1]; Levels(21)=[]; %% drop Levels = 0
for jj = 2:length(t)
  u1 = u1_all(:,jj); u2 = u2_all(:,jj);
  figure(20+jj); FEMtrimesh(Mesh,u1); zlim(Amp*[-1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_1');
  figure(30+jj); FEMtrimesh(Mesh,u2); zlim(Amp*[-1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_2');
  figure(40+jj); clf; FEMtricontour(Mesh,u1,Levels); axis equal; title('u_1')
                 hold on; plot([x;x(1)],[y;y(1)],'k'); xlabel('x'); ylabel('y')
   figure(50+jj); clf; FEMtricontour(Mesh,u2,Levels); axis equal; title('u_2')
                  hold on; plot([x;x(1)],[y;y(1)],'k'); xlabel('x'); ylabel('y')
endfor
```



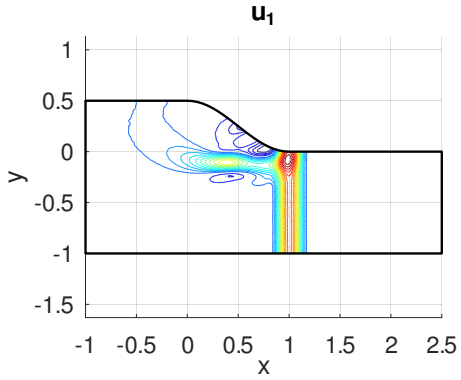
(a) graph of u_1 at $t = 1.0$



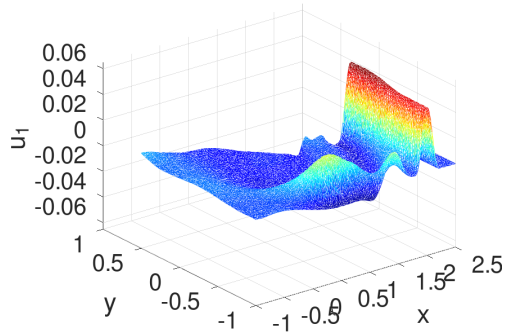
(b) contours of u_1 at $t = 1.0$



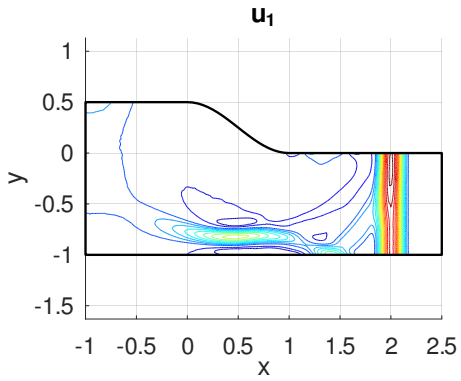
(c) graph of u_1 at $t = 2.0$



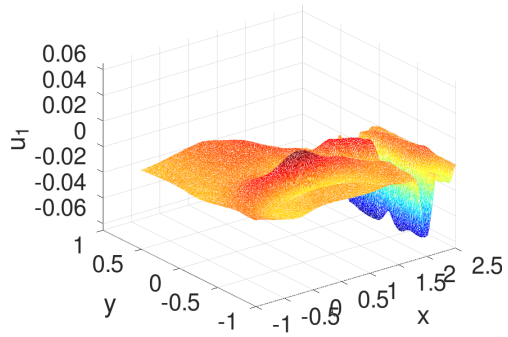
(d) contours of u_1 at $t = 2.0$



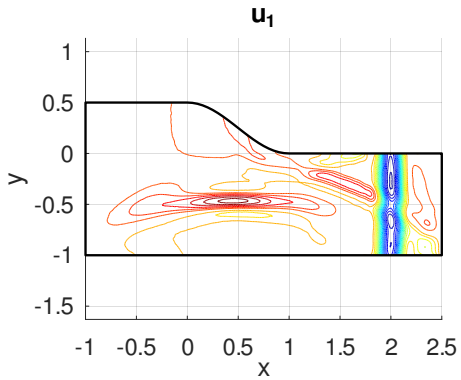
(e) graph of u_1 at $t = 3.0$



(f) contours of u_1 at $t = 3.0$

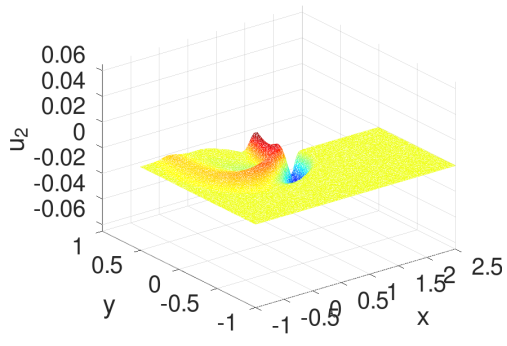


(g) graph of u_1 at $t = 4.0$

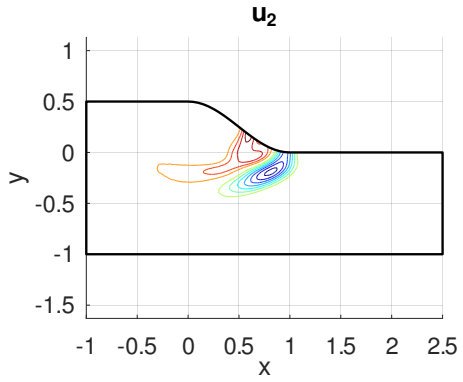


(h) contours of u_1 at $t = 4.0$

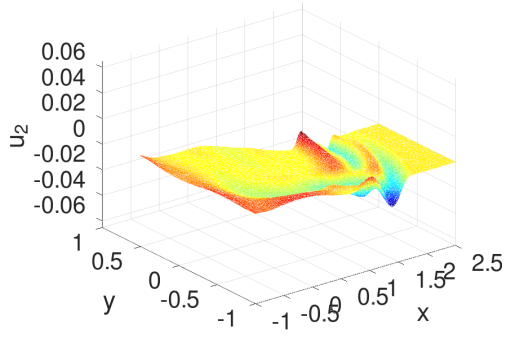
Figure 266: The horizontal displacement u_1 of an elastic cylindrical wave



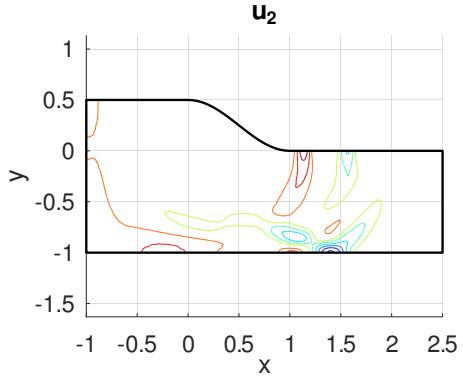
(a) graph of u_2 at $t = 2.0$



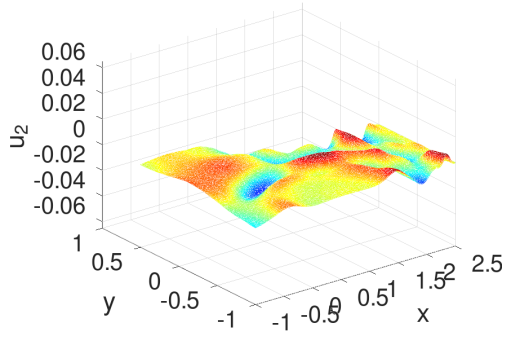
(b) contours of u_2 at $t = 2.0$



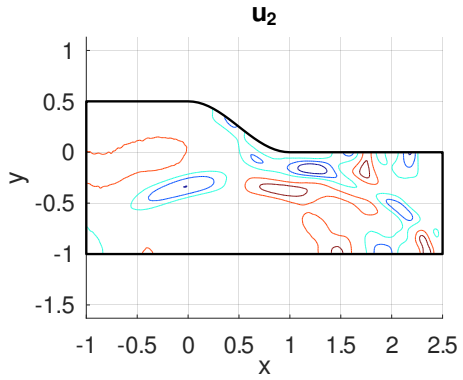
(c) graph of u_2 at $t = 3.0$



(d) contours of u_2 at $t = 3.0$



(e) graph of u_2 at $t = 4.0$



(f) contours of u_2 at $t = 4.0$

Figure 267: The vertical displacement u_2 of an elastic cylindrical wave

9.54.3 A planar wave moving around a turn

On the curved domain shown in Figure 268 force free boundary contions are used on all section, but the top right part. There zero displacements are imposed. An elastic wave is started horizontally in the lower left section by using the initial displacement

$$u_1(x,y) = \begin{cases} 0.1 \cdot \cos^2(4\,x) & \text{for} \quad 0 \le x \le \frac{\pi}{8} \approx 0.4 \\ 0 & \text{for} \quad \frac{\pi}{8} \le x \end{cases} ,$$

$u_2(x,y) = 0$ and zero initial velocity. Due to the rather theoretical material parameters $E = 1$, $\nu = 0$ and $\rho = 1$ the wave speed for longitudinal elastic waves is given by $c = \sqrt{\frac{E}{\rho}} = 1$. The single pulse of the initial form $\frac{0.1}{2}\ \cos^2(4\,x)$ is expected to move around the corner, leading to nonzero vertical displacements u_2. The code below leads to Figure 269, confirming the results. Caused by refelctions at the boundary of the curved domain the initial pulse will form a sizable tail.

- Observe that the values of u_1 decrease as time t advances and the values of u_2 increase.
- To examine different wave speeds, change the modulus of elasticity E, since the speed is given by $c = \sqrt{\frac{E}{\rho}}$.
- To examine the considerably smoother results on a horizontal strip, memove the bend by uncommenting the line with the command `MeshDeform()` in the code below. Observe the pure pulse moving to the right, no tail is forming and u_2 remains zero.
- To examine transversal waves change the initial displacement to $u_2(x,y)$ with the modified line

```
u0 = {0,'u0Func'}; v0 = {0,0};
```

For transversal waves the speed is given by $c_{trans} = \sqrt{\frac{G}{\rho}}$ with the shearing modulus $G = \frac{E}{2\,(1+\nu)}$. Thus for $\nu = 0$ find $c_{trans} = \frac{1}{\sqrt{7}}\,c \approx 0.7\,c$. The results in Figure 269 will change. Now u_2 will be large to start out.

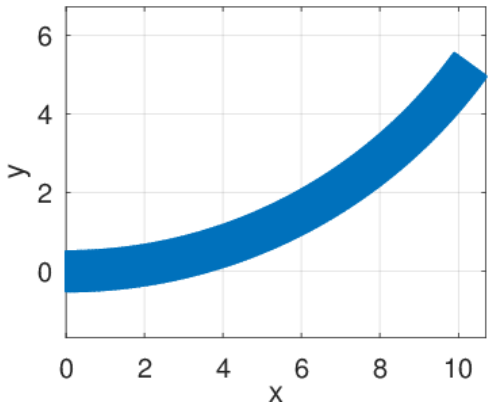


Figure 268: The domain for an elastic wave moving on a curve

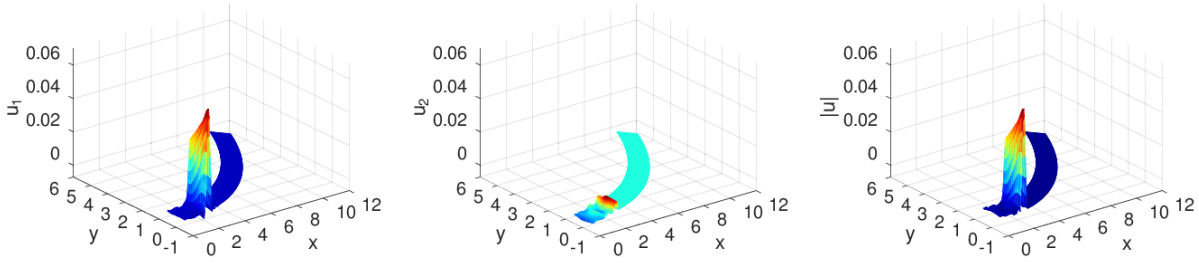
ElasticWaveCorner.m

```
%%% speed sqrt(E/rho) for longitudinal waves
E = 1; nu = 0; rho = 1; L = 12; H = 1; f = {0,0}; gD = {0,0}; gN = {0,0};
function res = u0Func(xy)
  res = 0.1*cos(4*xy(:,1)).^2.*(4*xy(:,1)<pi/2);
endfunction
u0 = {'u0Func',0}; v0 = {0,0}; t0 = 0; tend = 10; steps = [5,200];

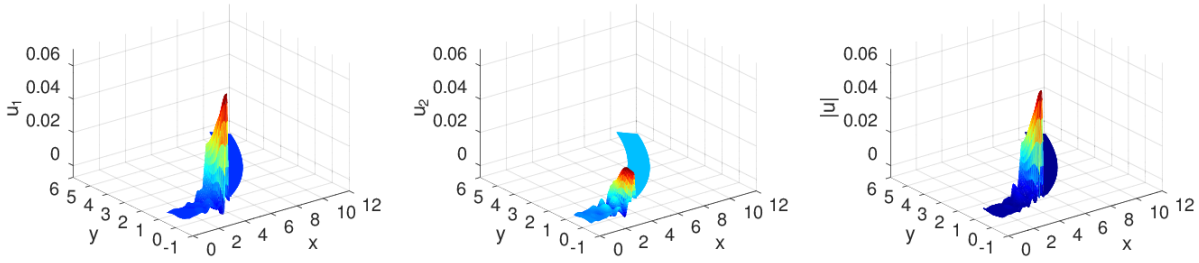
Mesh = CreateMeshRect(linspace(0,L,121),linspace(-H/2,+H/2,11),-22,-22,-22,-11);
function res = Deform(xy)
  alpha = pi/8;  R = 5/alpha;
```

```
  x = xy(:,1); y = xy(:,2) + R;  angles = x/R; r = y;
  res = [r.*sin(angles),R-r.*cos(angles)];
endfunction
Mesh = MeshDeform(Mesh,'Deform');
Mesh = MeshUpgrade(Mesh,'quadratic');
[u1_all,u2_all,t] = PlaneStressDynamic(Mesh,E,nu,rho,f,gD,gN,u0,v0,t0,tend,steps);

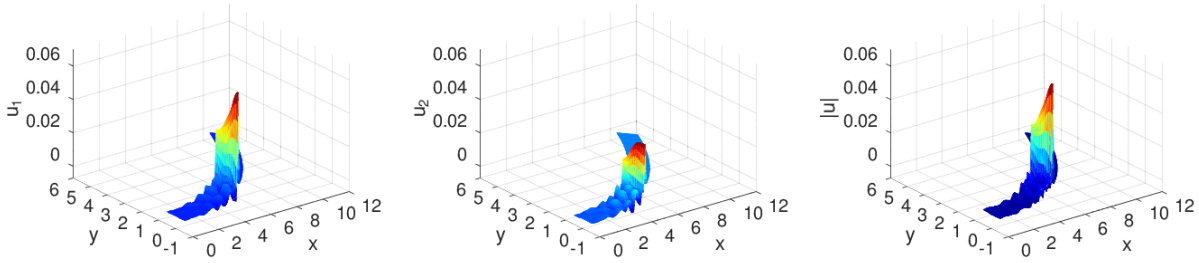
Amp = 0.07;
for jj = 2:length(t)
  u1 = u1_all(:,jj); u2 = u2_all(:,jj);
  disp(sprintf('at time t=%i, max(u1) = %g, max(u2) = %g, max(u) = %g',...
                    t(jj),max(u1),max(u2),max(sqrt(u1.^2+u2.^2))))
  figure(20+jj); FEMtrimesh(Mesh,u1); zlim(Amp*[-0.1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_1');
  figure(30+jj); FEMtrimesh(Mesh,u2); zlim(Amp*[-0.1,1])
                 xlabel('x'); ylabel('y'); zlabel('u_2');
  figure(40+jj); FEMtrimesh(Mesh,sqrt(u1.^2+u2.^2)); zlim(Amp*[-0.1,1])
                 xlabel('x'); ylabel('y'); zlabel('|u|');
endfor
-->
at time t=2,  max(u1) = 0.0610161, max(u2) = 0.00706702, max(u) = 0.0612235
at time t=4,  max(u1) = 0.0633773, max(u2) = 0.0178088,  max(u) = 0.0653784
at time t=6,  max(u1) = 0.0561453, max(u2) = 0.0253832,  max(u) = 0.0613949
at time t=8,  max(u1) = 0.0469787, max(u2) = 0.0311124,  max(u) = 0.0563221
at time t=10, max(u1) = 0.0409416, max(u2) = 0.0369974,  max(u) = 0.0551817
```



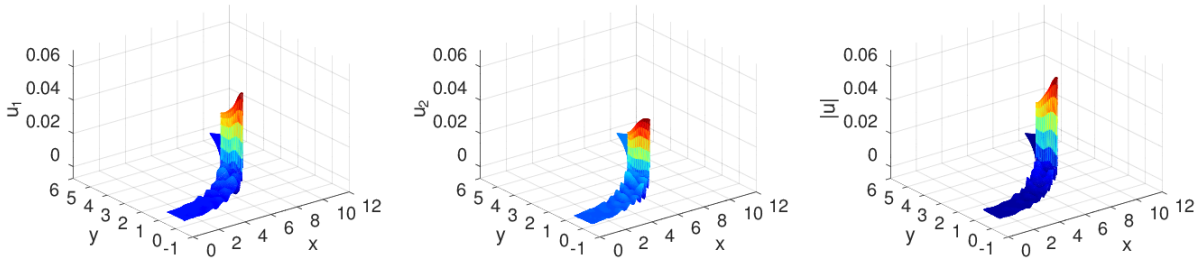
(b) $t = 2.0$



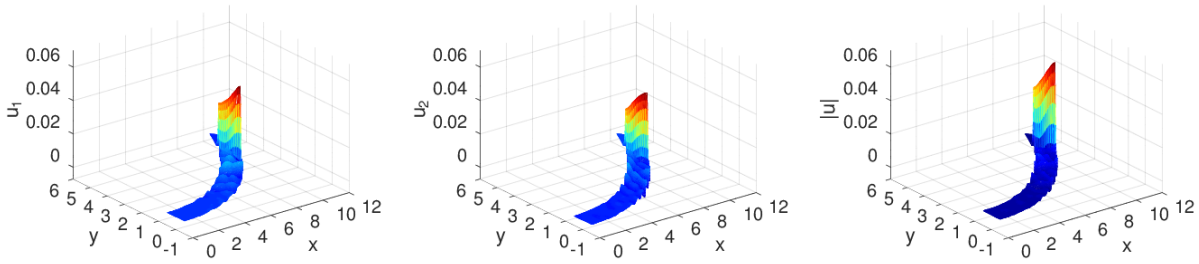
(e) $t = 4.0$



(h) $t = 6.0$



(k) $t = 8.0$



(n) $t = 10.0$

Figure 269: An elastic wave moving in a turn at times 2, 4 , 6, 8 and 10. The left column shows the horizontal displacement u_1, the middle column the vertical displacement u_2 and the column on the right $|u| = \sqrt{u_1^2 + u_2^2}$.

Bibliography

[AgarHodiRega19] R. P. Agarwal, S. Hodis, and D. O'Regan. *500 Examples and Problems of Applied Differential Equations*. Springer, 2019.

[AtkiHan09] K. Atkinson and W. Han. *Theoretical Numerical Analysis*. Number 39 in Texts in Applied Mathematics. Springer, 2009.

[AxelBark84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Values Problems*. Academic Press, 1984.

[Barb18] J. Barber. *Contact Mechanics*. Springer, 2018.

[Blev79] R. Blevins. *Formulas for Natural Frequency and Mode Shape*. Van Nostrand Reinhold, 1979.

[Butc03] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, second edition, 2003.

[Demm97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[Gold01] W. Goldsmith. *Impact*. Dover Civil and Mechanical Engineering. Dover Publications, 2001.

[GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[GrifSchr04] D. J. Griffiths and D. F. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, 3rd edition, 2004. available.

[IzadSuayNoei21] M. Izadi, S. Yüzbasi, and S. Noeiaghdam. Approximating Solutions of Non-Linear Troesch's Problem via an Efficient Quasi-Linearization Bessel Approach. *Mathematics*, 9(16), 2021.

[JacoSchm02] J. Jacobsen and K. Schmitt. The Liouville–Bratu–Gelfand problem for radial operators. *Journal of Differential Equations*, 184:283–298, 2002. PDF on disk.

[John87a] K. Johnson. *Contact Mechanics*. Cambridge University Press, 1987. see PDF.

[Kell92] H. B. Keller. *Numerical Methods for Two–Point Boundary Value Problems*. Dover, 1992.

[KhatMirc17] A. M. Khatkhate and M. P. Singh. A parametric approximation for the radius of curvature of a bimetallic strip. *International Journal of Engineering Research & Technology*, 6(6), June 2017.

[KubiHlav08] M. Kubicek and V. Hlavacek. *Numerical Solution of Nonlinear Boundary Value Problems with Applications*. Dover books on engineering. Dover, 2008.

[Mohs14] A. Mohsen. A simple solution for the bratu problem. *Computer and Mathematics with Applications*, 67(1):26–33, 2014. PDF on disk.

[MuelSilt12] J. Mueller and S. Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications*. Computational Science and Engineering. Society for Industrial and Applied Mathematics, 2012.

[MuelSilt20] J. Mueller and S. Siltanen. The D-bar method for electrical impedance tomography–demystified. *Inverse Problems*, 36:28, 2020.

[Prze68] J. Przemieniecki. *Theory of Matrix Structural Analysis*. McGraw–Hill, 1968. Republished by Dover in 1985.

[Sege77] L. A. Segel. *Mathematics Applied to Continuum Mechanics*. MacMillan Publishing Company, New York, 1977. republished by Dover 1987.

[Seyd00] R. Seydel. *Einführung in die numerische Berechnung von Finanz–Derivaten*. Springer, 2000.

[Seyd11] R. Seydel. *Tools for Computational Finance*. Springer, 5th edition, 2011.

[ShamDym95] I. Shames and C. Dym. *Energy and Finite Element Methods in Structural Mechanics*. New Age International Publishers Limited, 1995.

[www:triangle] J. R. Shewchuk. https://www.cs.cmu.edu/~quake/triangle.html.

[Soed04] W. Soedel. *Vibrations of Shells and Plates*. Dekker Mechanical Engineering. Taylor & Francis, 3rd edition, 2004.

[Sout73] R. W. Soutas-Little. *Elasticity*. Prentice–Hall, 1973.

[VarFEM] A. Stahel. Calculus of Variations and Finite Elements. Lecture Notes used at BFH–TI, 2000.

[Stah08] A. Stahel. Numerical Methods. lecture notes, BFH-TI, 2008.

[Stah22] A. Stahel. *Octave and MATLAB for Engineering Applications*. Springer Fachmedien Wiesbaden, Wiesbaden, first edition, 2022.

[Stew13] I. Stewart. *Seventeen Equations that Changed the World*. Profile Books Limited, 2013.

[TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.

[Tref75] J. S. Trefil. *Introduction to the Physics of Fluids & Solids*. Pergamon Press, 1975. republished by Dover, 2010.

[WaitMitc85] R. Wait and A. Mitchell. *Finite Element Analysis and Applications*. A Wiley-Interscience publication. Wiley, 1985.

[Wein74] R. Weinstock. *Calculus of Variations*. McGraw–Hill, New York, 1962. republished by Dover.

[Zien13] O. Zienkiewicz, R. Taylor, and J. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7 edition, 2013.

List of Figures

1 The selection tree for two dimensional problems . . . 10
2 The selection tree for one dimensional problems . . . 11
3 The selection tree for elasticity problems . . . 12
4 A semi–disk as domain in $\mathbb{R}^2$ and a solution of a BVP . . . 14
5 Deformation of an elastic solid . . . 18
6 Definition of strain: a small rectangle before and after deformation . . . 18
7 Components of stress in space . . . 19
8 Solution of $-\Delta u = 0.25$ on a rectangle . . . 26
9 Solution of the Laplace equation in cylindrical coordinates . . . 27
10 Solution of a diffusion problem on a L-shaped domain . . . 28
11 Solution of a diffusion convection problem . . . 29
12 The temperature u for a steady state heat equation with direction dependent conduction coefficients . . . 30
13 The flow lines and the energy flux through the boundary . . . 31
14 The solution of a nonlinear 2D boundary value problem . . . 32
15 The fourth eigenfunction of $\Delta u = \lambda\, u$ on a disc . . . 32
16 The first four radial eigenmodes of the Laplace operator on a disk of radius 1 . . . 34
17 An eigen function of a complex eigenvalue problem . . . 35
18 Solution of a dynamic heat equation . . . 37
19 Solution of a wave equation . . . 39
20 The final solution and its contour lines for a semilinear wave equation . . . 40
21 The solution for a simple 1D boundary value problem and the sparsity of the matrix $\mathbf{A}$ . . . 41
22 The solution of a steady state heat problem and the heat flux across spheres of radius r . . . 41
23 The solution of dynamic heating of a ball, u as function of r or t . . . 42
24 The solution of dynamic heating of a ball, surface and contours . . . 43
25 The amplitude of a vibrating string, u as function of x and t, the surface . . . 44
26 The amplitude of a vibrating string, u as function of x and t, the contour lines . . . 44
27 The solution and its contour for a nonlinear dynamic heat problem . . . 47
28 The solution and its contour for a nonlinear dynamic wave problem . . . 48
29 The final state $u(x, T_{end})$ and the steady state solution $u_{static}(x)$ . . . 48
30 The computational domain and the two displacements u_1 and u_2 for the plain stress example. . . . 49
31 The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy} for the plain stress example. . . . 50

32 The normal stresses σ_x and σ_y and the shearing stress τ_{xy} for the plain stress example. 50
33 The von Mises and Tresca stress for the plain stress example. 51
34 The computational domain and the two displacements u_1 and u_2 for the plain strain example. 52
35 The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy} for the plain strain example. 53
36 The normal stresses σ_x and σ_y and the shearing stress τ_{xy} for the plain strain example. 53
37 The von Mises and Tresca stress for the plain strain example. 54
38 Displacement and strain for a plane stress plate with thermal expansion 54
39 The normal stress σ_x for a plane stress plate with thermal expansion 55
40 Two normal stresses for a plane strain plate with thermal expansion 56
41 The first eigenmode of a bending beam 57
42 The first eigenmode of a bending beam as dynamic problem 58
43 The original and deformed domain and the von Mises stress for an axially symmetric setup 59
44 Stresses for an axially symmetric setup 59
45 A domain with a hole and a finer mesh at the lower edge 66
46 The deformed lever and the bending of the center line 66
47 A domain with a two different mesh sizes 67
48 The same mesh with linear or quadratic elements 69
49 A mesh generated by a Delaunay triangulation and the solution of a BVP 71
50 A function evaluated on a uniform grid 75
51 Convergence results for linear, quadratic and cubic elements 115
52 An linear, equilateral triangle, the Gauss integration points and the element stiffness matrix 117
53 Uniform meshes consisting of equilateral triangles 117
54 An equilateral, quadratic triangle, the Gauss integration points and the element stiffness matrix 118
55 A right triangle, the Gauss integration points and the element stiffness matrix 119
56 Uniform meshes consisting of rectangular triangles 120
57 A right angle triangle, the Gauss integration points and the element stiffness matrix 120
58 The number of nonzero entries in each row 123
59 The mesh and the solution for a BVP 124
60 Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a first order mesh 124
61 Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a second order mesh 125
62 Difference of the approximate values of $\frac{\partial u}{\partial y}$ to the exact values 125
63 Difference of the approximate values of u and $\frac{\partial u}{\partial y}$ to the exact values for cubic elements 126
64 Meshes for linear, quadratic and cubic elements 128
65 Graph and contour lines for the solution of the BVP (44) 129
66 The solution and the first derivative, evaluated at the nodes and by interpolation 132
67 The first derivative, complete graph and zoomed in at $x = 0.5$ 133
68 Differences of the FEM solution $u_{FEM}(x)$ to the exact solution $u_{exact}(x) = \exp(x)$ 133
69 The 2D and 1D solutions of $\Delta u = 1$ 135
70 Difference of the two solutions and the mesh close to $(R, 0)$ 135
71 The solution and difference to the exact solution for a 1D BVP 136
72 The derivative of the numerical solution and difference to the exact solution 137
73 The second derivative of the numerical solution and difference to the exact solution 138
74 Errors for the modified Crank–Nicolson time steppers 139
75 The solution generated by the implicit time stepper 141
76 The solution generated by the Crank–Nicolson time stepper 141
77 The solution at time $t = 2$ generated by the explicit and Runge–Kutta time steppers 141
78 Solutions of the wave equation at the time $t = t_{end}$ 143
79 Original and deformed domain and the Gauss integration points for linear elements 144
80 The strains ε_{xx} and ε_{xy} with two layers in each direction 145
81 The original shape of the a beam and its (exaggerated) deformed shape, using two layers of elements . 149
82 Meshes for linear and quadratic elements with one layer, with the integration points 150
83 The elastic energy density of the bending beam with one or five layers 150
84 The second eigenmode of a bending beam 153
85 The displacements u_x and u_y for a plate with a slit 154
86 The mesh of a plate with a slit 155
87 The von Mises stress for a plate with a slit 156
88 The solution of the boundary value problem with an additional constraint 157

89 The contours for the displacement $u = \sqrt{u_1^2 + u_2^2}$, without and with constraints . . . 158
90 Classical and weak solutions, minimizers and FEM . . . 162
91 A few triangular elements . . . 164
92 Transformation of the standard triangle Ω to a general triangle E . . . 164
93 Gauss integration of order 2 on the standard triangle, using 3 integration points . . . 166
94 Gauss integration of order 5 on the standard triangle, using 7 integration points . . . 166
95 Local and global numbering of nodes . . . 168
96 Basis functions for second order triangular elements . . . 174
97 Transformation of the cubic standard triangle Ω to a general triangle E . . . 183
98 The 10 basis functions for third order triangular elements . . . 185
99 The interpolation from four nodes to three Gauss points on an interval $[-\frac{h}{2}, +\frac{h}{2}]$ . . . 191
100 The Hermite triangle . . . 193
101 The ten functions related to the ten DOF of a Hermite element . . . 195
102 The mesh and the solution of an elliptic problem with variable coefficients . . . 243
103 Traveling waves on a rectangle . . . 245
104 The radial Bessel function as solution of a BVP . . . 246
105 Difference to the exact solution of a BVP . . . 246
106 Difference to the exact solution of a BVP, using quadratic elements and interpolation to a finer grid. . . . 247
107 Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using second order elements . . . 247
108 Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using first order elements . . . 248
109 A solution with singular partial derivatives at the origin . . . 249
110 A solution with singular partial derivatives, graphs of $\frac{\partial u}{\partial x}$ and $\|\nabla u\|$ . . . 250
111 The solution of the BVP to examine the accuracy of Hermite elements . . . 251
112 The Hermite solution of the BVP in a corner . . . 251
113 The Hermite solution of the BVP along the coordinate axis . . . 252
114 A Gauss Bell surface as solution of a BVP . . . 254
115 A Gauss Bell surface, the difference to the exact solution . . . 255
116 The solution of a Liouville equation in $\mathbb{R}^2$ . . . 255
117 The solution of a Liouville equation in $\mathbb{R}$ and the difference to the solution in $\mathbb{R}^2$ . . . 256
118 Solutions for the parameters α and C and three solutions . . . 257
119 The branch of all solutions to the the Bratu equation . . . 258
120 Two solution of Bratu's equation for $C = 1$ . . . 260
121 The branch of solutions for Bratu's equation in 2D . . . 261
122 The velocity profile $u(r)$ for a Poiseuille flow through a pipe $1 < r < 2$ . . . 263
123 The velocity profile $u(x, y)$ for a Poiseuille flow through a pipe $1 < r < 2$ . . . 263
124 The velocity profile $u(x, y)$ for a Poiseuille flow through a pipe with an offset center . . . 264
125 Fluid flow between two plates, the setup . . . 265
126 Velocity field of an ideal fluid . . . 266
127 The domains for a potential flow around a cylinder . . . 267
128 The speed of the flow aound a cylinder . . . 269
129 The vector field and flow lines for a flow around a cylinder . . . 269
130 Velocity field of a ideal fluid in a circular pipe . . . 271
131 The wing profile and the mesh with the rotated wing . . . 273
132 The surface for v^2 and the pressure along wing . . . 274
133 The contour plot for the pressure . . . 275
134 The flow lines around the wing . . . 275
135 A minimal surface . . . 276
136 The capacitance and the section used for the modeling . . . 277
137 A mesh on the domain . . . 278
138 The contour lines of the resulting voltage . . . 279
139 Voltage plot and electric field between the plates of the capacitance . . . 279
140 Torsion of a shaft . . . 280
141 The von Mises stress caused by torsion of a bar with square or rectangular cross section . . . 283
142 The mesh for a dynamic heat problem . . . 284
143 The evolution of the temperature surface at different times . . . 285
144 The temperature surface at different times along $y = 0$ . . . 285
145 The temperature as function of time at the endpoint $(2.5\,,\,0)$ . . . 286
146 The mesh for a dynamic heat problem . . . 287

147 The evolution of the temperature surface at different times . . . 287
148 The temperature surface at different times along $y = 0$ . . . 287
149 The temperature as function of time at the endpoint $(2.5\,, 0)$ . . . 288
150 The domain and the initial temperature . . . 289
151 The temperature at different times . . . 289
152 The temperature at different times along $y = 0$ . . . 290
153 The temperature decay at the center $(0, 0)$ . . . 291
154 The domain for a heat wave propagation . . . 292
155 The propagation of a heat wave . . . 292
156 The steady state solution of a heat problem in a ball . . . 294
157 The errors of the steady state solution of a heat problem in a ball . . . 295
158 The steady statesolution of a heat problem in a ball, solved as 1D problem . . . 295
159 The temperatur $u(r, t)$ inside the cylinder . . . 296
160 The contours of the temperatur $u(r, t)$ inside the cylinder . . . 297
161 The temperatur $u(r, t)$ at the inner ($r = 0$) and outer ($r = R$) edge . . . 297
162 The domain for the wave propagation . . . 298
163 Wave propagation, leading to a Kirchhoff diffraction pattern . . . 299
164 A spherical sound wave at time $t = 1.75$, and the decaying amplitude with the best fitting $\frac{c}{t}$ . . . 300
165 A circular sound wave at time $t = 4$ and the decaying amplitude with the best fitting $\frac{c}{\sqrt{t}}$ . . . 302
166 A spherical wave as function of time t and radius $r = \sqrt{x^2 + y^2 + z^2}$ . . . 303
167 A cylindrical wave as function of time t and radius $r = \sqrt{x^2 + y^2}$ . . . 304
168 The amplitude $u(x, t)$ for a reflected pulse . . . 305
169 Waterfall plot and contour plot for the amplitude $u(x, t)$ for a reflected pulse . . . 305
170 Traveling kink solutions for the Sine–Gordon equation . . . 306
171 A breather solution of the Sine–Gordon equation . . . 307
172 The final state of a breather solution of the Sine–Gordon equation . . . 308
173 The solution of a Helmholtz equation . . . 310
174 The solution $u(x, 0)$ for $-l \leq x \leq +l$ of a Helmholtz equation . . . 311
175 The solution of a Helmholtz equation with variable speed c of the wave . . . 311
176 The solution $u(x, 0)$ for $-l \leq x \leq +l$ of a Helmholtz equation with variable c . . . 312
177 The value V of a European call option as function of time τ and the value of the stock S . . . 314
178 The value V of a European put option as function of time τ and the value of the stock S . . . 315
179 The spherical harmonics for the eigenvalue $\lambda = 12$ . . . 316
180 The movement of a free particle with speed 10 and an initial Gauss curve . . . 319
181 The movement of a particle with speed 10 and an initial Gauss curve with an intermediate size potential 319
182 The movement of a particle with speed 10 and an initial Gauss curve with a stronger size potential . . . 320
183 The time evolution of the Schrödinger equation for a bouncing ball . . . 321
184 The first six eigenfunctions of the harmonic Schrödinger operator . . . 322
185 The PDF for the first six modes of the harmonic Schrödinger operator . . . 323
186 Graphs of a few eigenfunctions of the hydrogen atom . . . 324
187 Graphs and contours of the first three eigenfunction of the hydrogen atom . . . 327
188 Graphs and contours of fourth, fifth and sixth eigenfunction of the hydrogen atom . . . 328
189 Surface and contour plot for the helium ground state . . . 329
190 The conductivity . . . 331
191 Contours of the voltages . . . 332
192 The vector field for the current density $\vec{J}$ and a few streamlines . . . 333
193 Voltage along the boundary . . . 334
194 Differences of the voltage and the reference voltage . . . 334
195 Flux density at inlet and outlet . . . 336
196 A catenary, the FEM solution and the difference to the exact solution . . . 337
197 Four solutions of the brachistochrone problem . . . 339
198 Displacement and strain for a beam with thinner midsection . . . 340
199 Fata Morgana: the ray of light for a point on the ground . . . 341
200 Fata Morgana: how the image of two points appear to the observer . . . 342
201 Keller's boundary value problem . . . 343
202 Two solutions of $-u''(x) = \frac{1}{2}\exp(u(x))$ with $u(-1) = u(+1) = 0$ . . . 345
203 A BVP with multiple nonlinear contributions . . . 346

204 The solution of Fisher's equation . . . 348
205 A dynamic solution of Fisher's equation . . . 350
206 A dynamic solution of Fisher's equation for the radially symmetric setup . . . 351
207 A dynamic solution of Fisher's equation on the plane . . . 352
208 A dynamic solution of Fisher's equation with anisotropic diffusion coefficients . . . 352
209 The connections between Salt Lake City and Zürich . . . 354
210 The solutions for a bending beam . . . 358
211 Time evolution and the final solution for a dynamic bending beam . . . 360
212 A BVP describing mass transfer in a porous catalyst . . . 360
213 A BVP describing mass transfer in a porous catalyst, second setup . . . 361
214 Motion of a string, excited by an initial pulse close to $x = 0$ . . . 363
215 The original and deformed Mesh for a plane stress example . . . 364
216 The displacements for a plane stress example . . . 365
217 The stresses for the Wait–Mitchell plane stress example . . . 365
218 The von Mises stress and the energy density for a plane stress example . . . 366
219 The normal strain in the $45°$ direction for a plane stress example . . . 367
220 The displacements for a heated bimetallic strip . . . 368
221 The normal stress σ_x as surface and along a slice at $x = L/2$ . . . 369
222 The vertical displacements $u_2(x, 0)$ along the axis $y = 0$ . . . 369
223 The mesh of the copper plate with the heated disk . . . 371
224 The displacements of the copper plate with the heated disk . . . 371
225 The normal strains of the copper plate with the heated disk . . . 371
226 Normal stresses of the copper plate with the heated disk along the circle . . . 372
227 Domain with a hole in the plate of thickness 0.5 mm . . . 373
228 Horizontal and vertical displacement on the plate with the hole . . . 374
229 The von Mises stress on the plate with the hole . . . 375
230 The von Mises stress on the plate with the loading inside of the hole . . . 376
231 The normal stress σ_x on the plate with the loading inside of the hole . . . 377
232 The horizontal stress σ_x and the normal stress σ_n along the inner edge of the hole . . . 377
233 One quarter of a section through the pipe . . . 378
234 Von Mises stress and principal stresses . . . 380
235 Domain for the crook with attached weight . . . 385
236 A horizontal slice with σ_y shown and a vertical slice with σ_x shown . . . 387
237 The von Mises stress on the crook, as surface and level curves . . . 388
238 Bending of vertical beam and von Mises Stress at corner . . . 388
239 The deformed wrench and the stress σ_y along upper edge with the applied load . . . 390
240 Surface and contour plot of the von Mises stress in [MPa] . . . 391
241 The upper half of the original and deformed domain for the rotating rubber box . . . 391
242 The displacements u_r and u_z for the rotating rubber box . . . 392
243 The von Mises stress for the rotating rubber box . . . 392
244 The original and deformed domain of the washer . . . 393
245 The radial displacement u_r . . . 394
246 The height displacement u_z . . . 395
247 The normal stress σ_z . . . 395
248 The normal pressures σ_z along upper and lower edge and at half height . . . 396
249 The normal stresses and the von Mises stress in a water dam . . . 399
250 Three eigenmodes of a tuning fork . . . 401
251 The first twelve eigen modes of a vibrating ring . . . 403
252 Radial and angular displacement for mode 7 . . . 405
253 The mesh for the cylinder contact problem . . . 407
254 Vertical displacement and normal stress at different levels . . . 408
255 Contact with cylinder: contours for the displacements u_2 and u_1 and $\sqrt{u_1^2 + u_2^2}$ . . . 410
256 Contact with cylinder: contours for the two normal stresses and the von Mises stress . . . 411
257 Contact with cylinder: results of pressure and contact width for small penetration depth . . . 413
258 Contact with cylinder: results of pressure and contact width for large penetration depth . . . 414
259 Contact with sphere: results of pressure and contact width for large penetration depth . . . 416
260 Contact with sphere: contours for the displacements u_z and u_r and $\sqrt{u_r^2 + u_z^2}$ . . . 417
261 Contact with sphere: contours for the two normal stresses and the von Mises stress . . . 417

262 Contact with a sphere for given normal pressure $\sigma_z(r,0)$ . . . 418
263 The von Mises stress for a contact with a sphere for given normal pressure $\sigma_z(r,0)$ . . . 418
264 The horizontal displacement u_1 of an elastic cylindrical wave . . . 421
265 The vertical displacement u_2 of an elastic cylindrical wave . . . 422
266 The horizontal displacement u_1 of an elastic cylindrical wave . . . 424
267 The vertical displacement u_2 of an elastic cylindrical wave . . . 425
268 The domain for an elastic wave moving on a curve . . . 426
269 An elastic wave moving through a turn . . . 428

List of Tables

1 Commands to solve BVPs, IBVPs and elasticity problems . . . 13
2 Normal and shear strains in space . . . 19
3 Description of normal and tangential stress in space . . . 19
4 Commands to create and modify meshes . . . 60
5 Elements of a mesh structure . . . 61
6 Codes for the boundary conditions . . . 62
7 Commands to display and evaluate solutions . . . 72
8 Commands to solve and examine 1D BVP and IBVP . . . 86
9 Commands to solve and examine plane elasticity problems . . . 101
10 Commands to solve and examine axially symmetric elasticity problems . . . 106
11 Internal commands of `FEMoctave` . . . 109
12 Results for elements of order 1, 2 and 3 . . . 129
13 The minimal energy for different types of elements . . . 130
14 Errors for the modified Crank–Nicolson time steppers . . . 139
15 Elastic energy contributions for shearing . . . 146
16 Different values for the deformation of a bending beam, depending on the size of the grid . . . 151
17 A few properties of triangular elements . . . 163
18 Coordinates of the nodes in the standard quadratic triangle . . . 173
19 Coordinates of the nodes in the standard cubic triangle . . . 183
20 Entries in the matrix **BF2DOF** with the reduced polynomials in (71) . . . 197
21 Properties of the ODE solvers used in `IBVP1D()` . . . 216
22 Computation time and error for generating the Bell surface . . . 253
23 Variables for the Black–Scholes PDE . . . 313
24 Comparison of different elements for the washer fastener example . . . 398
25 Parameters for the contact of a cylinder with a half space . . . 406
26 Adaptations for the contact problem of a plane with with a cylinder or a sphere . . . 415

List of Algorithms

1 The algorithm of BVP1DNL() . . . 93
2 Update the linear mesh to a Hermite mesh . . . 200
3 Convert a Hermite mesh to a linear mesh . . . 201

Index

AxiStress(), 58, 105, 381, 383, 384, 390, 391, 393, 415, 418, 419
AxiStressEquationCubicM(), 112
AxiStressEquationM(), 112, 160
AxiStressEquationQuadM(), 112
axisymmetric, 24

basis function, 173, 183
Bernoulli principle, 21, 24, 25, 224
Bernoulli's law, 272
BF2DOF, 196
BF2uGPmat(), 110
bimetallic strip, 367
Black–Scholes, 313
Bohr radius, 326
bouncing ball, 320
boundary condition, 213
brachistochrone, 338
Bratu equation, 256, 344
BVP, 26
    boundary value problem, 13
    eigenvalue, 15
    elliptic, 13
    symmetric, 14
BVP1D(), 40, 41, 85, 86, 132–134, 136, 262, 295, 340, 342, 343, 354, 357
BVP1Deig(), 33, 34, 91, 92, 322
BVP1DNL(), 45–47, 92, 93, 255, 257, 258, 337, 338, 341, 344–346, 348, 355, 358–362
BVP2D(), 28, 29, 79, 157, 250, 253, 263, 264, 291, 309
BVP2Deig(), 32, 35, 79, 291, 316, 326, 329
BVP2DNL(), 31, 81, 203, 255, 260, 261
BVP2Dsym(), 26–28, 78, 243, 268, 273, 294, 333

catenary, 336
Cholesky, 204
conforming, 74, 124, 131–134, 163
constant strain triangle, 146, 227
convection, 28
convergence, 114
Crank–Nicolson, 89, 94, 138, 140, 204, 216–218
CreateMeshRect(), 26, 27, 29, 35, 51, 56, 62, 69, 74, 119, 146, 151, 154, 244, 245, 250, 259, 268, 300, 301, 316, 326, 329, 368, 378, 379, 386, 418–420, 423, 426
CreateMeshTriangle(), 28, 31, 32, 38, 49, 63, 65, 71, 75, 115, 116, 119, 124, 155, 157, 243, 245, 248, 253, 255, 259, 263–265, 271, 273, 276, 283, 285, 286, 289, 292, 298, 329, 364, 372, 373, 384, 385, 389, 393, 402, 404, 407, 418, 419
CST, 146, 227

Delaunay, 70
Delaunay2Mesh(), 70
DIRK, 217
DOF2BF, 196

eigenvalue, 32, 111, 205, 207
eigs, 111
eigSmall(), 108, 111
EIT, 331
element stiffness matrix, 167
EvaluateEnergyDensity(), 104, 151, 365
EvaluateEnergyDensityAxi(), 108, 384, 397
EvaluatePrincipalStress(), 51, 53, 103, 380
EvaluatePrincipalStressAxi(), 107, 382
EvaluateStess(), 54, 55
EvaluateStrain, 372
EvaluateStrain(), 50, 52, 54, 55, 100, 147, 155, 374
EvaluateStrainAxi(), 106, 108, 382, 384
EvaluateStress, 372
EvaluateStress(), 50, 53, 101, 154, 155, 365, 368, 374, 379, 386, 387, 390, 398, 402, 408
EvaluateStressAxi(), 58, 106, 382, 393, 415, 418, 419
EvaluateTresca(), 51, 53, 104
EvaluateTrescaAxi(), 107, 382
EvaluateVonMises(), 51, 53, 102, 103, 155, 365, 374, 379, 387, 388, 390, 398, 408
EvaluateVonMisesAxi(), 58, 107, 382, 393, 415, 418, 419
explicit, 89, 140, 204, 207, 216

Fata Morgana, 340
fBF2uGP(), 197
fDOF2BF(), 110
FEM1DEvaluateDu(), 85, 87, 132–134, 215, 337, 341, 343, 345, 354
FEM1DGaussPoints(), 85, 88, 343, 353, 354, 357
FEM1DIntegrate(), 88, 337
FEMEquation(), 109, 116
FEMEquation.cc, 108
FEMEquationComplex(), 109
FEMEquationComplex.cc, 108
FEMEquationCubic(), 110
FEMEquationCubic.cc, 108
FEMEquationCubicComplex(), 110
FEMEquationCubicComplex.cc, 108
FEMEquationCubicM(), 110
FEMEquationCubicM.m, 108
FEMEquationHermite(), 110
FEMEquationHermite.m, 108
FEMEquationM(), 109
FEMEquationM.m, 108
FEMEquationQuad(), 110
FEMEquationQuad.cc, 108
FEMEquationQuadComplex(), 110
FEMEquationQuadComplex.cc, 108
FEMEquationQuadM(), 110
FEMEquationQuadM.m, 108
FEMEvaluateGP(), 73, 148, 397
FEMEvaluateGradient(), 72, 249, 268
FEMgriddata(), 30, 66, 74, 144, 146, 246, 250, 252–255, 268, 269, 274, 275, 291, 294, 310, 316, 333, 336, 366, 380, 385–388, 390, 408, 410, 419

FEMIntegrate(), 73, 147, 148, 243, 263, 264, 365, 384, 397
FEMInterpolBoundaryWeight(), 111
FEMInterpolWeight(), 111
FEMSolve(), 108, 110
FEMtricontour(), 28, 29, 31, 35, 70, 71, 265, 271, 274, 280, 333, 366, 374, 388, 390, 393, 420, 423
FEMtrimesh(), 26–28, 31, 32, 35, 38, 49, 70, 71, 124, 154, 157, 243, 244, 248, 249, 255, 265, 268, 271, 274, 280, 292, 298, 316, 333, 365, 374, 390, 393, 398, 402, 418–420, 423, 426
FEMtrisurf(), 71, 276, 283, 364, 388
Fermat's principle, 341
Fisher's equation, 347

Galerkin method, 162
Gauss integration, 74, 165–167, 171, 174, 181, 184, 192, 210, 211, 229, 232
GenerateFEM1D(), 85, 87
GenerateWeight1D(), 85

Hamiltonian, 326
heat equation, 15, 28, 35, 36, 203, 205
helium ground state, 326
Helmholtz, 308, 309, 311
Hermite, 62
Hooke's law, 20
hydrogen, 322

I2BVP1D(), 43, 85, 90, 143, 302–304, 363
I2BVP1DNL(), 85, 95, 221, 307
I2BVP2D(), 38, 84, 244, 298, 300, 301
I2BVP2DNL(), 38, 47, 85
IBVP, 26
  hyperbolic, 16, 84, 85, 206
  parabolic, 15, 16, 82, 83, 203
IBVP1D(), 42, 85, 88, 89, 140, 296, 313, 315, 318, 320
IBVP1DNL(), 46, 85, 94, 138, 349, 350, 359
IBVP2D(), 35, 82, 204, 285, 286, 292
IBVP2DNL(), 36, 83, 138, 351
IBVP2Dsym(), 82, 205, 289, 292
implicit, 89, 140, 204, 216, 217
interpolation, 210

Jaccobi determinant, 165

Liouville equation, 254

MeahDeform(), 268
mesh, refine, 70
MeshAddConstraint(), 68, 157, 158, 367, 368, 370, 372
MeshCubic2Linear, 250
MeshCubic2Linear(), 68, 69
MeshDeform(), 51, 70, 243, 268, 325, 378, 379, 426
meshgrid(), 269
MeshHermite2Linear(), 69, 72
MeshQuad2Linear(), 29, 68, 69, 245
MeshUpgrade(), 29, 49, 51, 56, 63, 64, 68, 154, 155, 243, 245, 250, 253, 268, 273, 294, 298, 333, 364, 379, 385, 389, 402, 404, 407, 418, 419
minimal surface, 276

pendulum, 345
plane strain, 23, 51, 55, 56, 96, 97, 378, 406
plane stress, 21, 49, 54, 96, 97, 363, 373, 386, 402
PlaneStrain(), 51, 54, 55, 97, 379, 398, 405, 408
PlaneStrainDynamic(), 99
PlaneStrainEig(), 98, 402
PlaneStress(), 49, 66, 97, 155, 364, 368, 372, 374, 385, 389
PlaneStressDynamic(), 57, 99, 420, 423, 426
PlaneStressEig(), 56, 98, 152, 154, 160, 400, 402, 404
potential flow, 264, 270, 272
Prandtl stress function, 282
principal stress, 51, 53, 103
PStressEquationCubicM(), 112
PStressEquationM(), 112, 159
PStressEquationQuadM(), 112
PStressEquationWM(), 226
pwquadinterp(), 41, 85, 87, 132–134, 136, 214, 257, 295, 340, 343, 353, 354, 357, 361, 362

quantum mechanics, 316, 318, 322, 326

ReadMeshTriangle(), 63, 64, 68
Ritz method, 162
Runge–Kutta, 89, 140, 204, 217
Rydberg constant, 324

scattering, 308, 311
Schrödinger equation, 318, 321, 322
shear–locking, 144, 146, 151, 152, 385, 402
ShowDeformation(), 51, 71, 100, 151, 379, 391, 393, 402, 404
Sine–Gordon equation, 306, 307
singular problem, 248
solution
  classical, 161, 162
  weak, 162
spherical harmonics, 316
stability
  A–stability, 216
  conditional, 89, 140, 216
  L–stability, 89, 140, 216
  unconditional, 89, 140, 216
stiffness matrix
  element, 163
  global, 163
strain, 18
streamline(), 30, 269, 275, 333
stress
  concentration, 154, 155, 365, 373, 374, 385
  principal, 51, 53, 103, 382
  Tresca, 51, 53, 104, 382
  von Mises, 51, 53, 102, 282, 365, 373, 379, 382, 388, 390, 393
successive substitution, 342, 353
superconvergence, 114, 136, 245, 295

tensor
    infinitesimal strain, 18
thermal expansion, 20, 22, 24, 54, 225, 229, 232, 235, 367, 370, 378
time step
    Crank–Nicolson, 204, 217
    explicit, 216
    implicit, 217
    Runge–Kutta, 217
torsional rigidity, 281
Tresca stress, 51, 53, 104, 382
triangle, 8, 63, 64, 113, 278
tricontour(), 71, 113
tuning fork, 400

water dam, 398
wave equation, 36, 38, 203, 206, 207