FEMoctave, Finite Element Algorithms in Octave

Andreas Stahel, Bern University of Applied Sciences

Version 2.0.14 of 19th May 2023

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 lecture notes. 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 plane stress and plane strain problems are examined. This leads to the *Octave* package **FEMoctave**.
  - Provide examples on how to solve steady state and dynamic heat equations, the wave equation and 2D elasticity equations, all part of FEMoctave.
- 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 it by calling
    `pkg install https://github.com/AndreasStahel/FEMoctave/archive/v2.0.10.tar`
  - 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. The source code of `triangle` is not included. Find source code and documentation at 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* [Stah22] by Springer.
- 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 use some of the presentations for a class *Numerical Methods* for biomedical engineers at the University of Bern. There the main ideas of FEM are spelled out. Find the lecture notes for this class on my web site at https://andreasstahel.github.io/Notes/NumMethods.pdf.

The structure of this document is as follows:

1 **Introduction**: a self reference.

2 **The Problems to be Examined**: for each type of problem 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 FEM are presented.

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

7 **Elasticity**: the mathematical aspects of an FEM algorithm to solve plane stress and plane strain problems are presented.

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

2 The Problems to be Examined

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 given in Table 1 on page 10. The instruction on how to use the commands are given in Section 4. Some typical examples are worked out in Section 8.

2.1 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 disjoints section 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 a known functions g_2 and g_3.

In the example shown in Figure 1 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

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

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

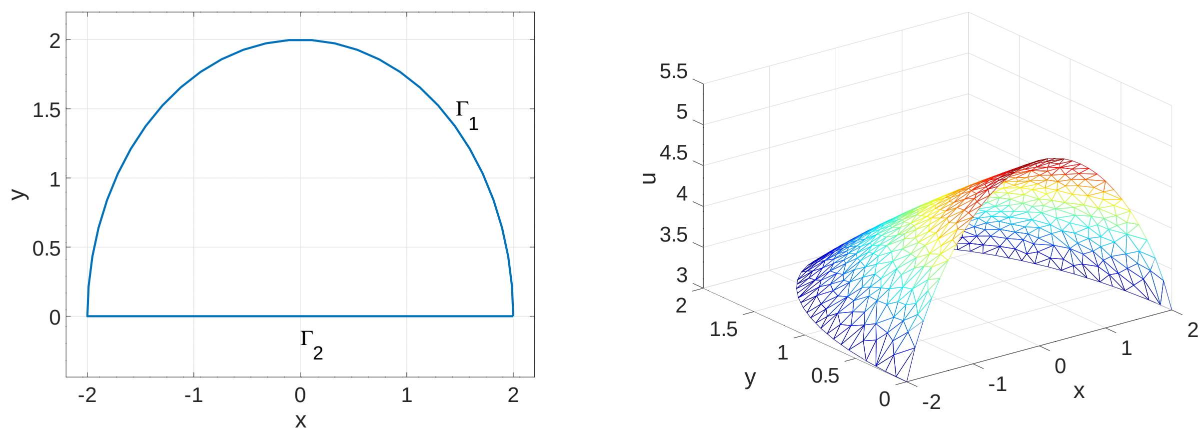


Figure 1: A semi–disk as domain in $\mathbb{R}^2$ and a solution of a BVP

2.2 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)

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

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 3.1.4.

2.3 The symmetric elliptic problem

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

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

The resulting matrix $\mathbf{A}$ will be symmetric and if $a>0$, $b_0\geq 0$ and $\Gamma_1\neq O$ 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 the functional F below among all functions u vanishing on Γ_1.

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

Examples of this type are given in Sections 3.1.1, 3.1.2, 3.1.3, 8.4 and 8.13.

2.4 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.

$$\begin{array}{rcll} -\nabla\cdot(a\,\nabla u)+b_0\,u & = & \lambda\,f\,u & \text{for} \quad (x,y)\in\Omega \\ u & = & 0 & \text{for} \quad (x,y)\in\Gamma_1 \\ a\,\frac{\partial u}{\partial n} & = & g_3\,u & \text{for} \quad (x,y)\in\Gamma_2 \end{array} \tag{3}$$

An example of this type is given in Section 3.2.

2.5 The general parabolic problem

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

$$\begin{array}{rcll} \rho\,\frac{\partial}{\partial t}\,u-\nabla\cdot(a\,\nabla u-u\,\vec{b})+b_0\,u & = & f & \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\ u & = & g_1 & \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\ \vec{n}\cdot(a\,\nabla u-u\,\vec{b}) & = & g_2+g_3\,u & \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\ u & = & u_0 & \text{on} \quad \Omega \quad \text{at} \quad t=0 \end{array} \tag{4}$$

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 8.10.4.

2.6 The symmetric parabolic problem

Consider the symmetric situation of (4) to find the symmetric parabolic problem below.

$$\begin{array}{rcll} \rho\,\frac{\partial}{\partial t}\,u-\nabla\cdot(a\,\nabla u)+b_0\,u & = & f & \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\ u & = & g_1 & \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\ a\,\frac{\partial\nabla u}{\partial n} & = & g_2+g_3\,u & \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\ u & = & u_0 & \text{on} \quad \Omega \quad \text{at} \quad t=0\} \end{array} \tag{5}$$

If $u(x,y)$ and λ are solutions of the eigenvalue problem (3) with $f=g_1=g_2=0$, then the dynamic problem (5) is solved by $e^{-\lambda t}\,u(x,y)$. See also Section 6.8.2.

2.7 The hyperbolic problem

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

$$\begin{array}{rcll} \rho\,\frac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u & = & f & \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\ u & = & g_1 & \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\ \vec{n}\cdot(a\,\nabla u + u\,\vec{b}) & = & g_2 + g_3\,u & \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\ u & = & u_0 & \text{on} \quad \Omega \quad \text{at} \quad t=0 \\ \frac{\partial}{\partial t}\,u & = & v_0 & \text{on} \quad \Omega \quad \text{at} \quad t=0 \end{array} \tag{6}$$

Examples are shown in Sections 3.4, 8.2 and 8.11. The effect of eigenvalues is described in Section 6.8.4.

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
`IBVP2D()`	solve a parabolic IBVP	4.6
`IBVP2Dsym()`	solve a symmetric parabolic IBVP	4.6
`I2BVP2D()`	solve a hyperbolic IBVP	4.7
`PlaneStress()`	solve a plane stress problem	4.8
`PlaneStrain()`	solve a plane strain problem	4.8

Table 1: Commands to solve PDEs and IBVPs

2.8 Plane Elasticity

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

2.8.1 Description of strain

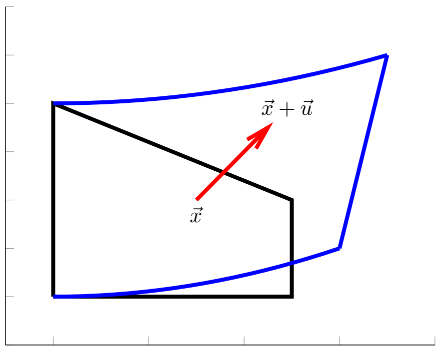
The first goal is to determine the displacement function $\vec{u} = (u_1, u_2)$. It describes the displacement of arbitrary points $(x, y) \in \Omega \subset \mathbb{R}^2$. Based on $\vec{u}(x, y)$ the infinitesimal strain tensor is given by

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

It contains the essential information of how a small section of the large solid is deformed, see Figure 3. 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 geometric interpretations in Table 2.



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

Figure 2: Deformation of an elastic solid

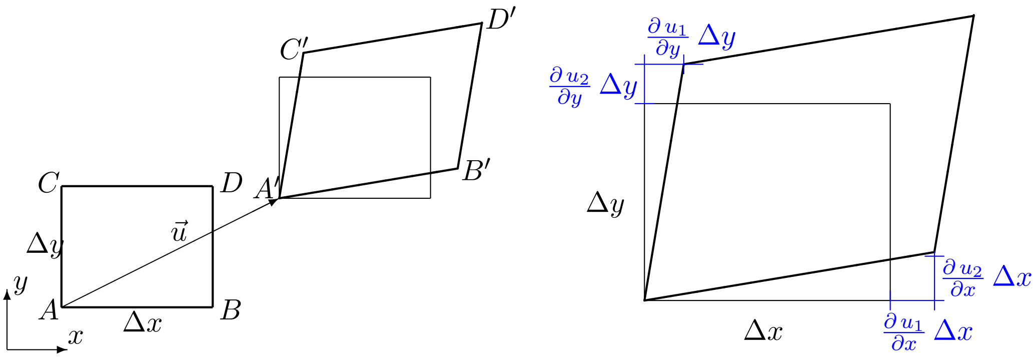


Figure 3: Definition of strain: rectangle before and after deformation

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

2.8.2 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. Find a graphical interpretation of the 6 strains in space $\mathbb{R}^3$ in Figure 4 and a description in Table 3.

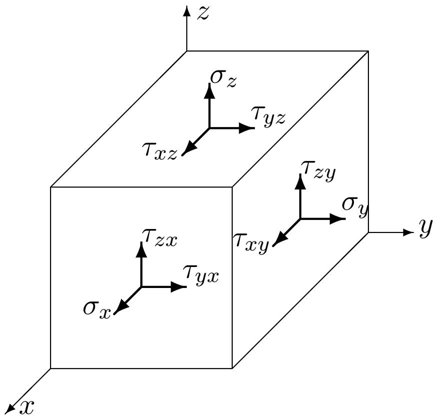


Figure 4: Components of stress in space

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

Table 3: Description of normal and tangential stress in space

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

$$\begin{array}{rcll} \vec{u} & = & \vec{g}_D & \text{on Dirichlet boundary } \Gamma_1 \text{, i.e. prescibed displacement} \\ \text{force density} & = & \vec{g}_N & \text{on Neumann boundary } \Gamma_2 \text{, i.e. prescibed force density} \\ \text{force density} & = & \vec{0} & \text{on free boundary } \Gamma_3 \end{array} \tag{7}$$

The conditions can be set for each component, find the codes in Table 5, 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 basic 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].

$$\begin{aligned}
\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} &= \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} , \\
\begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} &= \frac{1+\nu}{E} \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}
\end{aligned} \tag{8}$$

or by inverting the matrix

$$\begin{aligned}
\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} &= \frac{E}{(1+\nu)\,(1-2\,\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} , \\
\begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} &= \frac{E}{1+\nu} \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} .
\end{aligned} \tag{9}$$

This leads to an elastic energy density of

$$W = \frac{1}{2} \langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle + \langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \rangle \tag{10}$$

$$= \frac{1}{2} \frac{E}{(1+\nu)\,(1-2\,\nu)} \langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle + \tag{11}$$

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

2.8.3 The plane stress problem

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

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

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

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

This leads to a simpler version of Hooke's law

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \quad \text{and} \quad \begin{array}{rcl} \varepsilon_{zz} & = & \frac{-\nu}{1-\nu}\,(\varepsilon_{xx}+\varepsilon_{yy}) \\ \varepsilon_{xz} & = & 0 \\ \varepsilon_{yz} & = & 0 \end{array} \quad . \tag{12}$$

The energy density given by equation (10) simplifies to

$$\begin{aligned} W_{stress} &= \frac{1}{2}\,\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle + \langle \begin{pmatrix} \tau_{xy} \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \rangle = \frac{1}{2}\,\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 2\,\tau_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \\ &= \frac{E}{2\,(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \\ &= \frac{E}{2\,(1-\nu^2)}\,\left(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\,\nu\,\varepsilon_{xx}\,\varepsilon_{yy} + 2\,(1-\nu)\,\varepsilon_{xy}^2\right) \ . \end{aligned} \tag{13}$$

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]

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \qquad (14) \\ &= \iint\limits_\Omega \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle\, dA - \\ &\quad - \iint\limits_\Omega \vec{f}\cdot\vec{u}\, dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\, ds \ . \end{aligned}$$

Using the Bernoulli principle this energy has to be minimized. It is this minimization problem that is solved, subject to the boundary conditions (7).

2.8.4 The plane strain problem

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

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

This leads to a simpler version of Hooke's law

$$\begin{aligned} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} &= \frac{E}{(1+\nu)\,(1-2\,\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\,\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \ . \\ \sigma_z &= \frac{E\,\nu\,(\varepsilon_{xx}+\varepsilon_{yy})}{(1+\nu)\,(1-2\,\nu)} \quad , \quad \tau_{xz} = \tau_{yz} = 0 \end{aligned} \tag{15}$$

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

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

$$\nu^\star = \frac{\nu}{1-\nu} > \nu \quad\text{and}\quad E^\star = \frac{E}{1-\nu^2} > E\;. \tag{16}$$

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 (12) 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

$$\begin{aligned}
W_{strain} &= \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\langle\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & 2\,(1-2\,\nu)\end{bmatrix}\cdot\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\,,\,\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle\\
&= \frac{E\,(1-\nu)}{2\,(1+\nu)\,(1-2\,\nu)}\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\frac{\nu}{1-\nu}\,\varepsilon_{xx}\varepsilon_{yy}+2\,\frac{1-2\,\nu}{1-\nu}\,\varepsilon_{xy}^2\right)\\
&= \frac{1}{2}\,\frac{E^\star}{1-(\nu^\star)^2}\,\langle\begin{bmatrix}1 & \nu^\star & 0\\ \nu^\star & 1 & 0\\ 0 & 0 & 2\,(1-\nu^\star)\end{bmatrix}\cdot\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\,,\,\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle\\
&= \frac{E^\star}{2\,(1-(\nu^\star)^2)}\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\nu^\star\,\varepsilon_{xx}\varepsilon_{yy}+2\,(1-\nu^\star)\,\varepsilon_{xy}^2\right)
\end{aligned} \tag{17}$$

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 (14).

2.9 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[4] the strains in the plane $\varphi = 0$.

$$\begin{aligned}
\varepsilon_{xx} &= \frac{\partial\,u_1}{\partial x} = \cos^2\varphi\,\frac{\partial\,u_r}{\partial r} + \frac{\sin^2\varphi}{r}\,u_r = \frac{\partial\,u_r}{\partial r} \\
\varepsilon_{yy} &= \frac{\partial\,u_2}{\partial y} = \sin^2\varphi\,\frac{\partial\,u_r}{\partial r} + \frac{\cos^2\varphi}{r}\,u_r = \frac{1}{r}\,u_r \\
\varepsilon_{zz} &= \frac{\partial\,u_z}{\partial z} \\
2\,\varepsilon_{xy} &= \frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x} = \cos\varphi\,\sin\varphi\,\frac{\partial\,u_r}{\partial r} - \frac{\cos\varphi\,\sin\varphi}{r}\,u_r + \cos\varphi\,\sin\varphi\,\frac{\partial\,u_r}{\partial r} - \frac{\sin\varphi\,\cos\varphi}{r}\,u_r = 0 \\
2\,\varepsilon_{xz} &= \frac{\partial\,u_1}{\partial z} + \frac{\partial\,u_3}{\partial x} = \cos\varphi\,\frac{\partial\,u_r}{\partial z} + \cos\varphi\,\frac{\partial\,u_z}{\partial r} - \frac{\sin\varphi}{r}\,\frac{\partial\,u_z}{\partial\varphi} = \frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r} \\
2\,\varepsilon_{yz} &= \frac{\partial\,u_2}{\partial z} + \frac{\partial\,u_3}{\partial y} = \sin\varphi\,\frac{\partial\,u_r}{\partial z} + \sin\varphi\,\frac{\partial\,u_z}{\partial r} + \frac{\cos\varphi}{r}\,\frac{\partial\,u_z}{\partial\varphi} = 0
\end{aligned}$$

This leads to

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

Observe that the angular strain $\varepsilon_{\varphi\varphi}$ is given by the displacement $\varepsilon_{\varphi\varphi} = \frac{1}{r}\,u_r$. The energy density (10) in the rz–plane at angle $\varphi = 0$ is given by

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

[4]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

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

$$\begin{aligned} &= \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,(\varepsilon_{rr}^2+\varepsilon_{zz}^2+\frac{1}{r^2}\,u_r^2)+2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz}+\frac{1}{r}\,u_r\,(\varepsilon_{rr}+\varepsilon_{zz}))\right)+ \\ &\quad +\frac{E}{1+\nu}\,\varepsilon_{rz}^2\,. \end{aligned}$$

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 (14).

$$\begin{aligned} U(\vec{u}) &= U_{elast}+U_{Vol}+U_{Surf} \\ &= \iint\limits_{\Omega} \frac{2\,\pi\,r\,E}{2\,(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,(\varepsilon_{rr}^2+\varepsilon_{zz}^2+\frac{1}{r^2}\,u_r^2)+2\,\nu\,(\varepsilon_{rr}\varepsilon_{zz}+\frac{1}{r}\,u_r\,(\varepsilon_{rr}+\varepsilon_{zz}))\right)\,dA+ \\ &\quad +\iint\limits_{\Omega}\frac{2\,\pi\,r\,E}{1+\nu}\,\varepsilon_{rz}^2\,dA-\iint\limits_{\Omega}2\,\pi\,r\,\vec{f}\cdot\vec{u}\,dA-\int_{\Gamma_2}2\,\pi\,r\,\vec{g}_N\cdot\vec{u}\,ds\,. \end{aligned} \tag{18}$$

Some of the contributions are similar to the elastic energy for plane stress problems (13) or (14), 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 (7).

3 Illustrative Examples

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

1. Generate the mesh to be used for the problem. With this step the type of element can be selected,i.e. linear, quadratic or cubic.

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 equation

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

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

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 `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.

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 5. The above code is using linear elements. To use quadratic or cubic elements uncomment one of the lines with `MeshUpgrade()`.

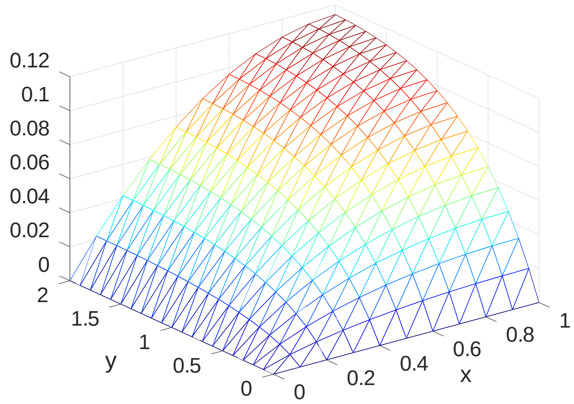


Figure 5: Solution of $-\Delta u = 0.25$ on a rectangle

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 6.

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);
```

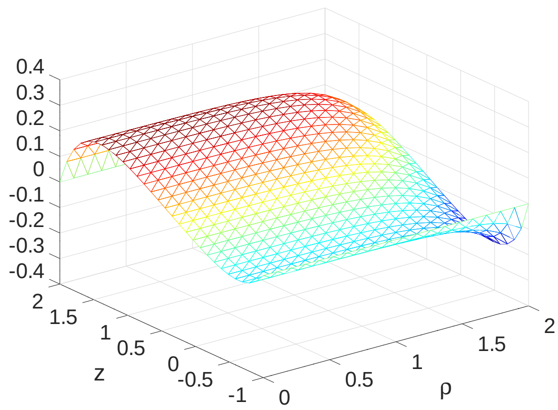


Figure 6: Solution of the Laplace equation in cylindrical coordinates

```
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

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

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 7.

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); FEMtricontour(FEMmesh,u);
           xlabel('x'); ylabel('y');
```

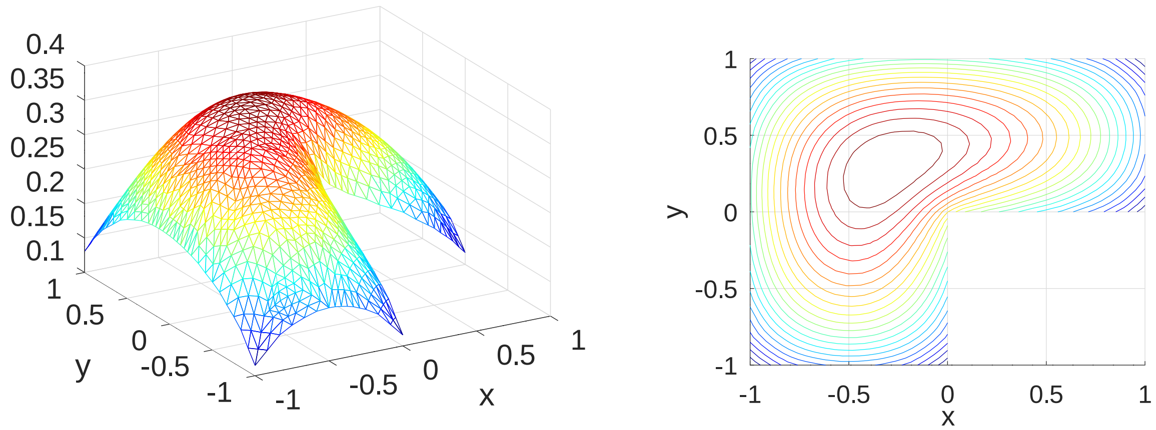


Figure 7: Solution of a diffusion problem on a L-shaped domain

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 8.

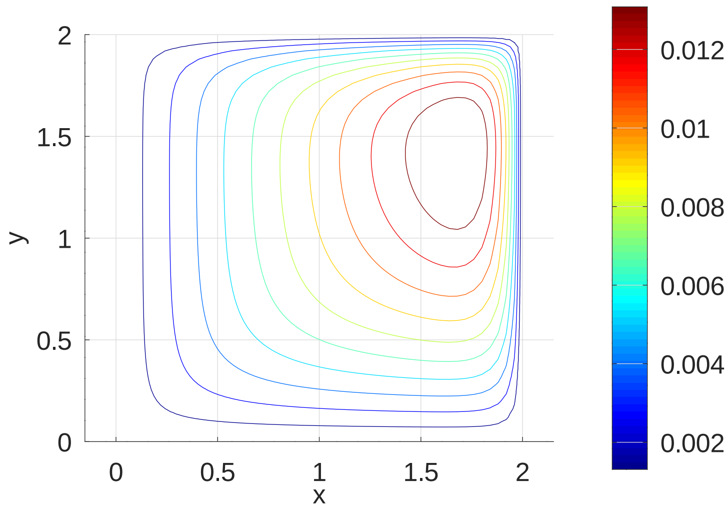


Figure 8: Solution of a diffusion convection problem

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

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 u has to vanish on the boundary. The goal is to compute the first four 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 9.
- 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.

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');
```

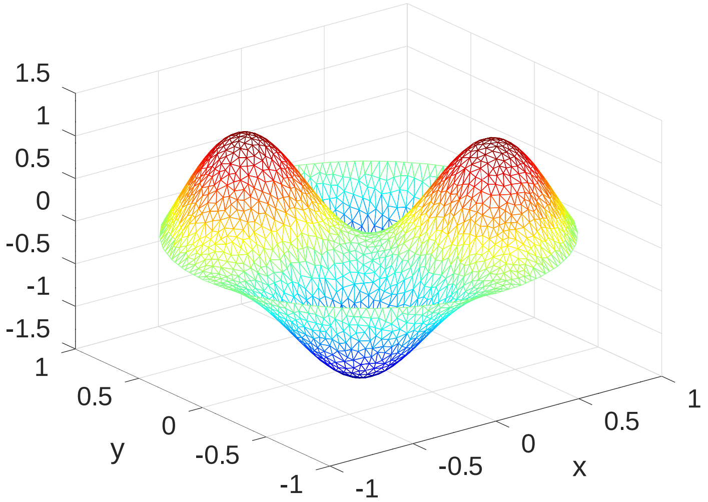


Figure 9: The fourth eigenfunction of $\Delta u = \lambda\, u$ on a disc

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.

Octave

```
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
```

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 10. 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 8 in Section 3.1.4.

HeatDynamic.m

```
%% generate the mesh
FEMmesh = CreateMeshRect(linspace(0,2,31),linspace(0,2,31),-1,-1,-1,-1);
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);
[u_dyn,t] = IBVP2D(FEMmesh,m,a,b0,bx,by,f,gD,gN1,gN2,u0,t0,tend,steps);
%% 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.elem,x,y,u_dyn(:,t_ii),linspace(0,0.99*u_max,11))
  xlabel('x'); ylabel('y'); caxis([0,u_max]); drawnow();
  pause(1)
endfor
```

3.4 Solving hyperbolic problems, wave equations

As an example solve the wave equation

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

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 11. This initial hump is traveling towards the boundary of the circle with speed 1, where it is reflected. More examples are shown in Sections 8.2 and 8.12.

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);
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;
```

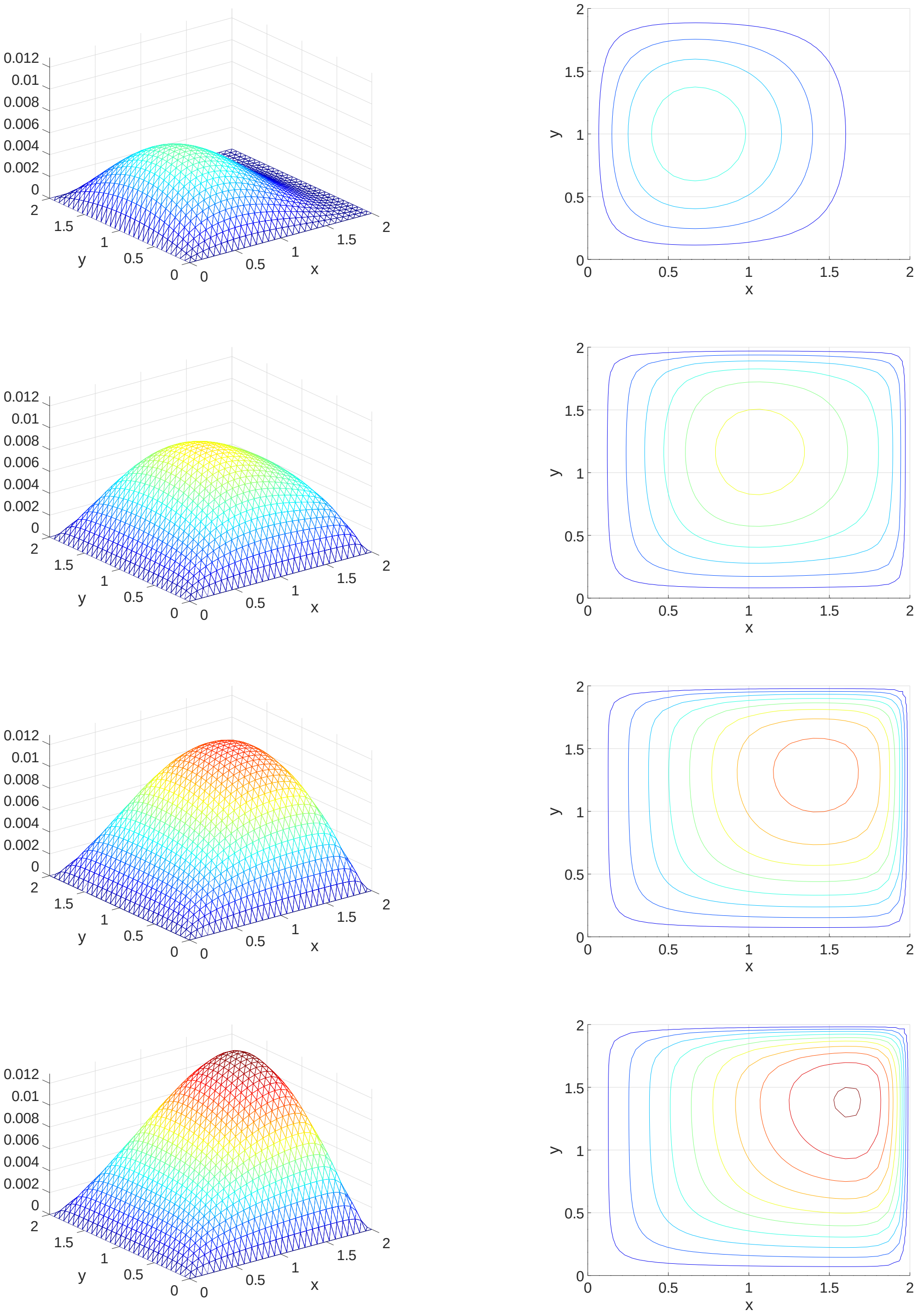


Figure 10: Solution of a dynamic heat equation

```
%% 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.
```

3.5 Plane elasticity

In this section a typical plane stress situation is examined and the related commands illustrated. This is followed by a similar plane strain situation.

3.5.1 A plane stress example

On a trapezoidal domain visible in Figure 12(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 $-0.05 \leq x \leq +0.05$.
- 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.
- 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 12.

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])
```

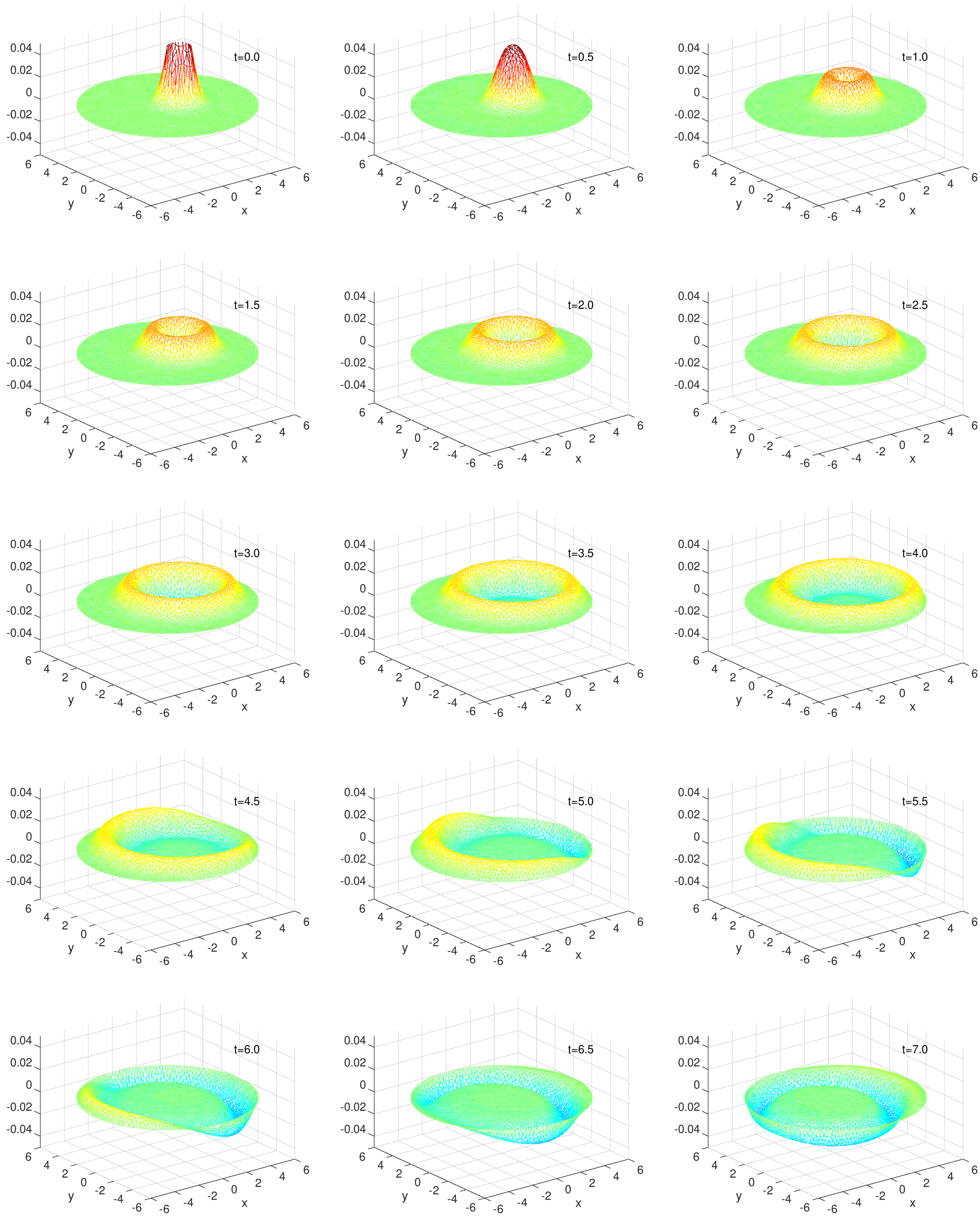
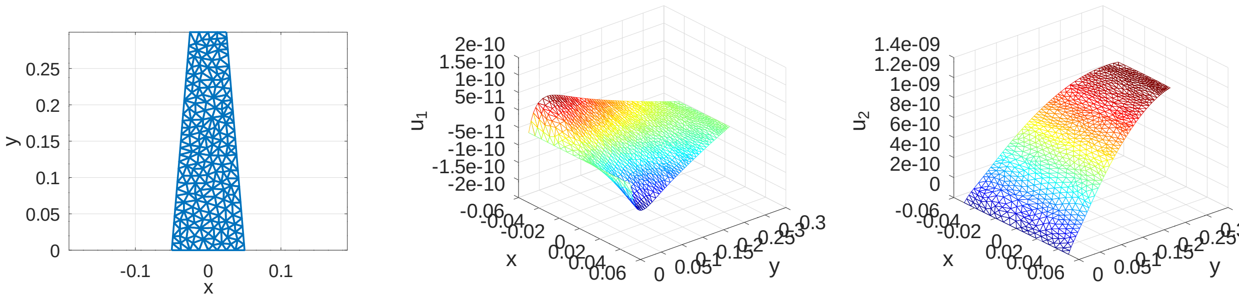


Figure 11: Solution of a wave equation



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

Figure 12: The computational domain and the two displacement functions u_1 and u_2

```
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 13. The Saint–Venant's principle at the lower edge $y = 0$ is clearly visible.

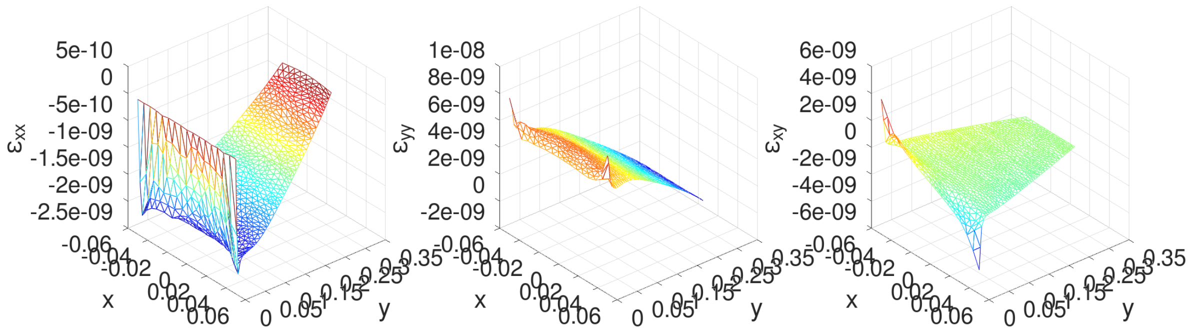


Figure 13: The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy}

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 14. The Saint–Venant's principle at the lower edge $y = 0$ is again clearly visible.

PlaneStressExample.m
```
[sigma_x,sigma_y,tau_xy] = EvaluateStress(FEMmesh,u1,u2,E,nu);
figure(5);
subplot(1,3,1); FEMtrimesh(FEMmesh,sigma_x)
```

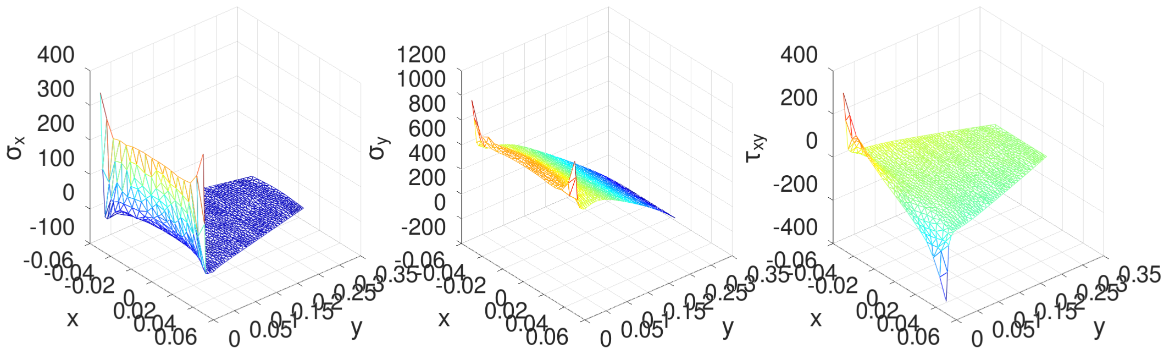
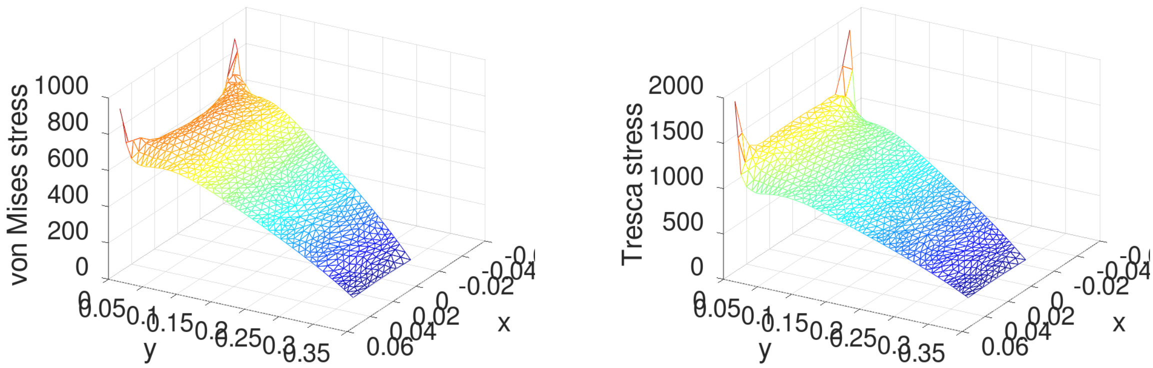


Figure 14: The normal stresses σ_x and σ_y and the shearing stress τ_{xy}

```
                 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 15. 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 15: The von Mises and Tresca stress

PlaneStressExample.m

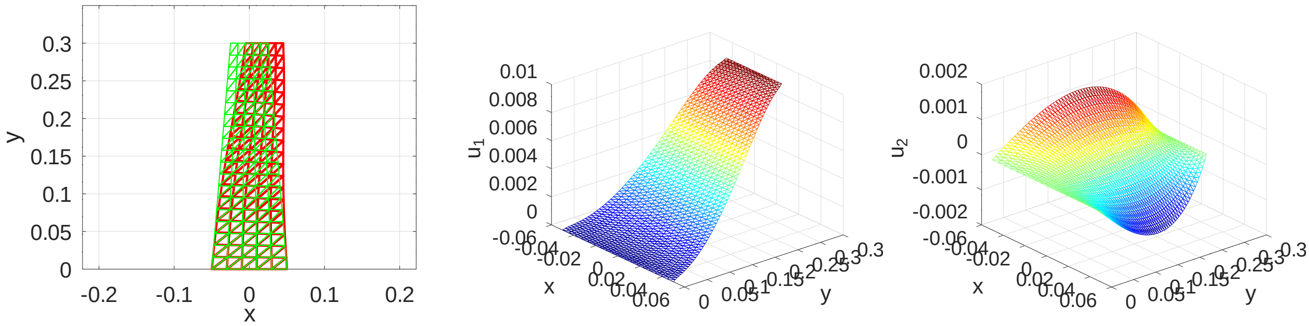
```
vonMises = EvaluateVonMises(sigma_x,sigma_y,tau_xy);
figure(6); FEMtrimesh(FEMmesh,vonMises)
           xlabel('x'); ylabel('y'); zlabel('von Mises stress'); view([120,30])
Tresca = EvaluateTresca(sigma_x,sigma_y,tau_xy);
figure(7); FEMtrimesh(FEMmesh,Tresca)
           xlabel('x'); ylabel('y');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

3.5.2 A plane strain example

On the trapezoidal domain visible in Figure 12(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 16. A corser mesh on the same domain is generated and then used to display the original and deformed domain. Find the result in Figure 16(a) with the original domain in green and the deformed domain in red.



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

Figure 16: The computational domain and the two displacement functions u_1 and u_2

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));
factor = 2;
figure(11); trimesh(CMesh.elem,CMesh.nodes(:,1)+factor*u1i,...
                    CMesh.nodes(:,2)+factor*u2i,'color','red','linewidth',2);hold on ;
            trimesh(CMesh.elem,CMesh.nodes(:,1),...
                    CMesh.nodes(:,2),'color','green','linewidth',1);
            hold off; 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 17.

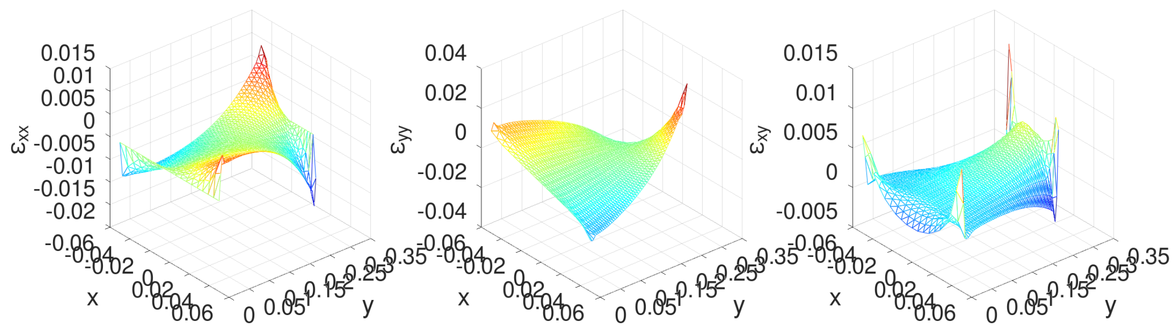


Figure 17: The normal strains ε_{xx}, ε_{yy} and the shearing strain ε_{xy}

PlaneStrainExample.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 18. 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.

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)
```

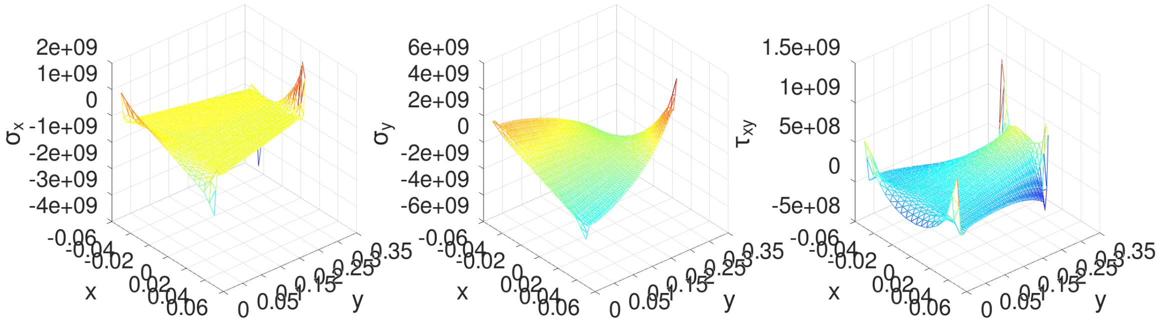
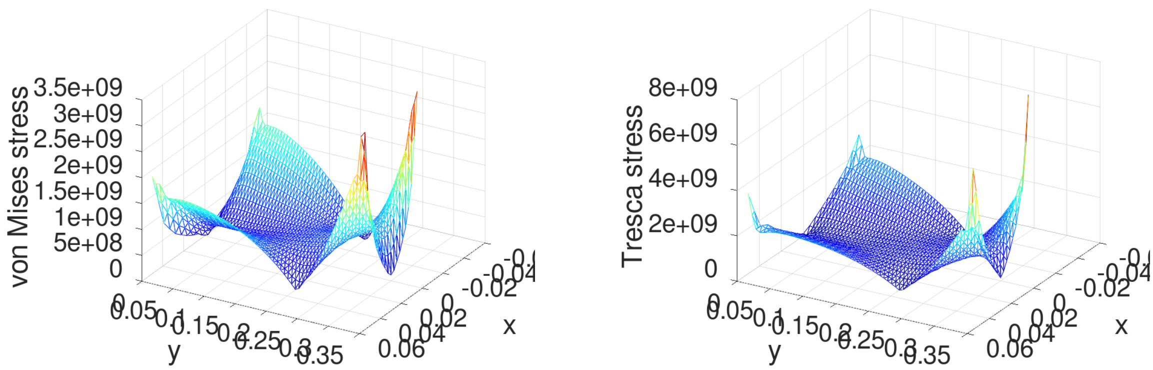


Figure 18: The normal stresses σ_x and σ_y and the shearing stress τ_{xy}

```
                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 15. 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.



(a) von Mises stress (b) Tresca stress

Figure 19: The von Mises and Tresca stress

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);
```

3.6 An axially symmetric example

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

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

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

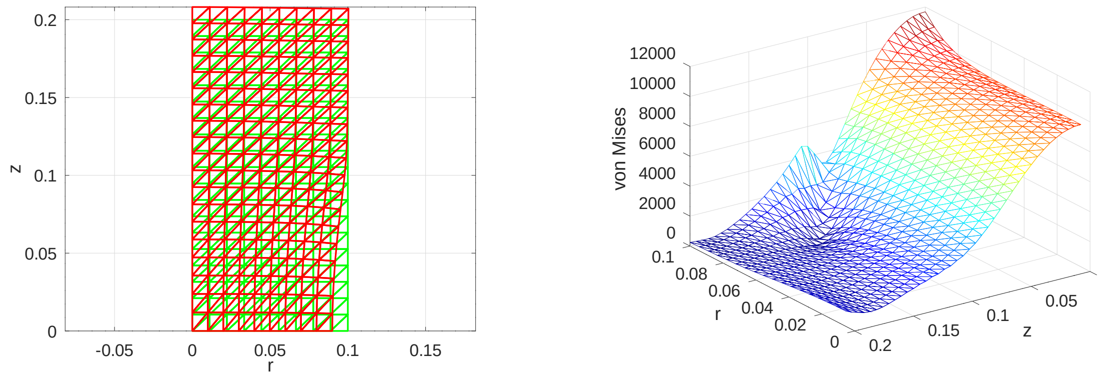
```
R = 0.1;
if 0  %% nonuniform mesh
  Mesh = CreateMeshTriangle('AxiSymm',[0 0 -21; R 0 -32; R 2*R -22; 0 2*R -12],1e-4);
else
  Mesh = CreateMeshRect(linspace(0,R,10),linspace(0,2*R,20),-21,-22,-12,-32);
endif
Mesh = MeshUpgrade(Mesh,'quadratic');

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 20.

AxiSymmetricExample.m

```
factor = 0.1*R/max(sqrt(ur.^2+uz.^2));
figure(1); trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),'color','green'); hold on
           trimesh(Mesh.elem,Mesh.nodes(:,1)+factor*ur,Mesh.nodes(:,2)+factor*uz,...
           'color','red')
           xlabel('r'); ylabel('z'); axis equal; hold off
```

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



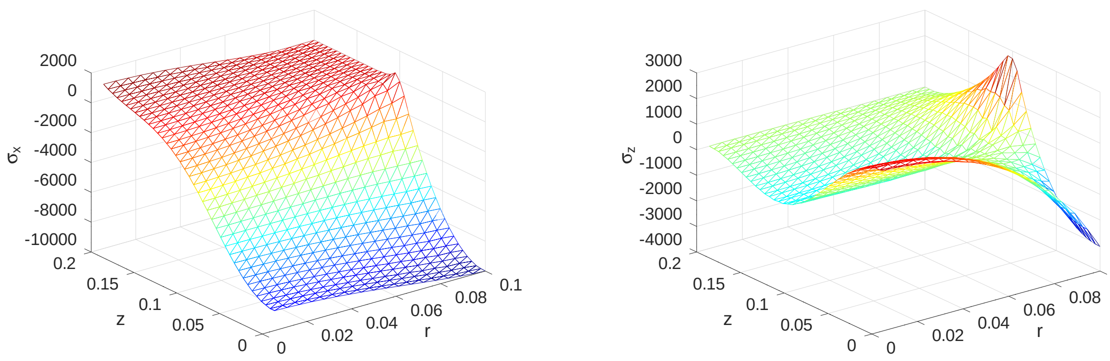
(a) original and deformed domain

(b) von Mises stress

Figure 20: 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) normal stress σ_x

(b) nomal stress σ_z

Figure 21: Stresses for an axially symmetric setup

4 The Commands of FEMoctave

In this section find the documentation for the commands provided by FEMoctave.

4.1 Commands for meshes: creation and modification

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) nodes forming one triangle,
Nb	boundary edges, with 2 (or 3) 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 4. 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`.
- `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`: types 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 5 for the codes.

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.

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, either 0 (free) or 1 (fixed)
`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 4: Elements of a mesh structure

- `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 5 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 $gN2(x, y)$.

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.

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 5: Codes for the boundary conditions

- 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 5.

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 4.1.5) you can upgrade to the same mesh with elements of order 2 or 3.

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 8 starting on page 153.

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. Currently no holes can be generated.

    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 he 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"

 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 or n_n by 2 matrix of integers, mapping nodes to DOF
```

- With `CreateMeshTriangle()` generate meshes with elements of order 1. With the help of the command `MeshUpgrade()` (Section 4.1.5) you can upgrade to the same mesh with elements of order 2 or 3.
- 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 8.8.

With `CreateMeshTriange()` and `ReadMeshTriangle()` one can only generate meshes with elements of order 1. With the help of `MeshUpgrade()` (Section 4.1.5) you can upgrade to the same mesh with elements of order 2 or 3.

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. Besides the mandatory entry `name='Segment'` one additional entry `border` has to be provided. 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 5. 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
```

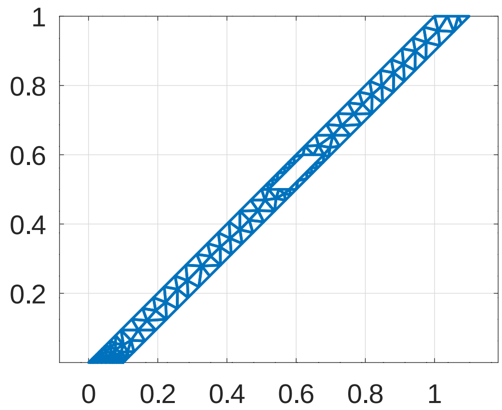
- **Option**: to give more options, not documented yet.

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

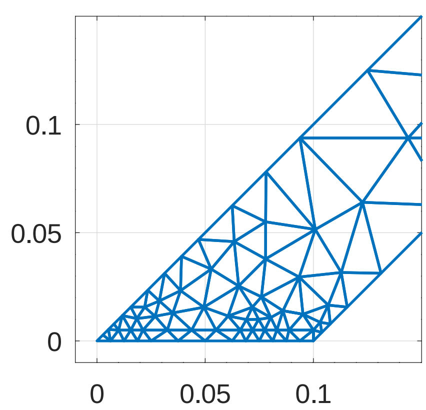
- The lines created by `Segment` shall not interfere with the holes.
- If `Segment` is used to divide the domain into multiple sections, the endpoints of the segments have to be exactly on the borders of the domain, but you can (often) not use the points defining the borders of the domain. A possible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

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 22. 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 22(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



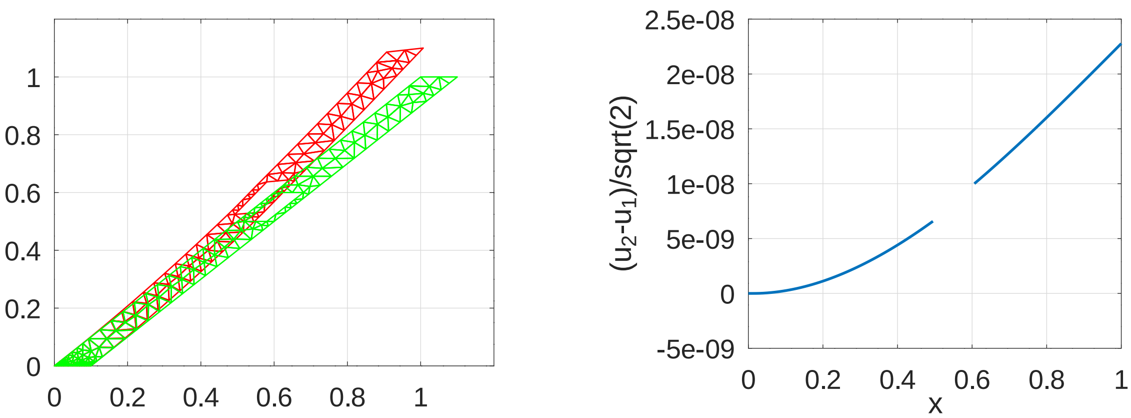
(b) the finer mesh at the lower edge

Figure 22: 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 23. 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);
trimesh(Mesh.elem,Mesh.nodes(:,1)+scale*u1,Mesh.nodes(:,2)+scale*u2,...
                 'color','red','linewidth',1)
hold on
trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),'color','green','linewidth',1)
hold off; 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 23: 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 24. The

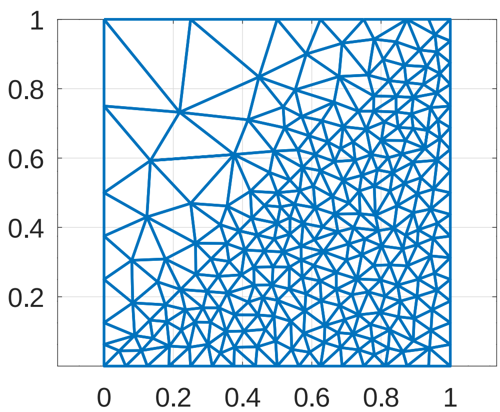


Figure 24: A domain with a two different mesh sizes

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.

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 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 48 on page 98 on how the nodes are placed within the triangles.

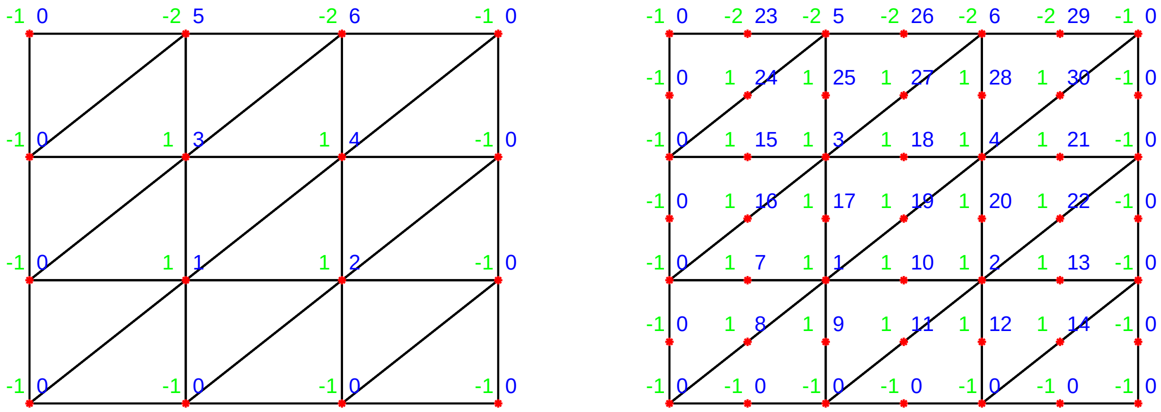
MeshUpgrade()

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

```
        the default is 'quadratic'
    return value: MESHNEW the output mesh of order 2 or 3
```

As example generate a mesh with elements of order 1 on the rectangle $0 \leq x, y \leq 2$ with Dirichlet conditions on three edges and a Neumann condition on the upper edge at $y = 2$. In Figure 25 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 25: 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 interiour 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()` one can 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()` one can 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
```

4.1.6 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 `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 26. For almost all cases `triangle` will generate meshes of better quality. To generate the domain and the solution in Figure 26 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)
           label('x'); ylabel('y'); view([100,45])
figure(3); FEMtricontour(FEMmesh,u);
           xlabel('x'); ylabel('y');
```

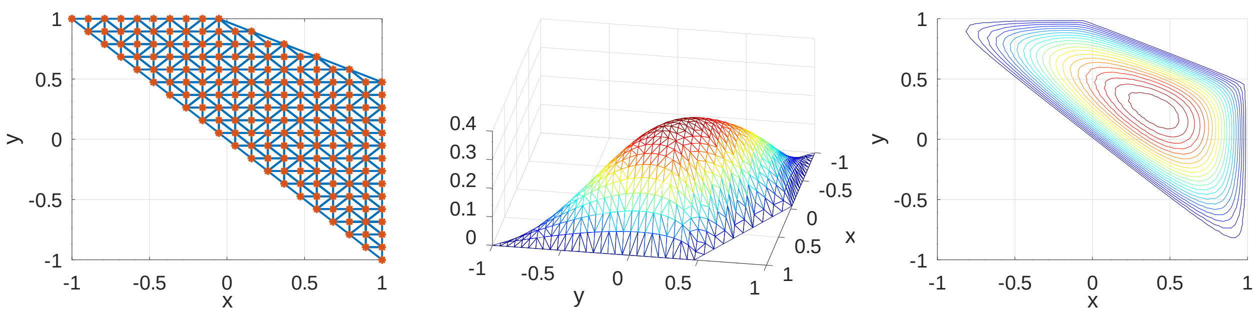


Figure 26: A mesh generated by a Delaunay triangulation and the solution of a BVP

4.1.7 Deforming meshes by MeshDeform()

With the function `MeshDeform()` the nodes of a linear mesh can be deformed.

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
```

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 41 on page 83 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 an example in Section 8.1.

4.2 Evaluation and displaying results

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.[5]

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

[5]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 in to your code.

All functions accept meshes with linear, quadratic or cubic 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.

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, 8.3, 8.4, 8.5 and 8.9. 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!

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.

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 8.7 and 8.9.

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 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.

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 or as scalar
         value, or as a vector with the values at the nodes or 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 $0 \leq x, y \leq 1$. 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, 8.1, 8.7 and 8.9.

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 78.
- 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 8.5 and 8.13.

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 `FEMEvaluateGradient()` look smoother on occasion.

The code below evaluates a function on an L-shaped domain on a rectangular grid. Find the result in Figure 27.

```
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.
```

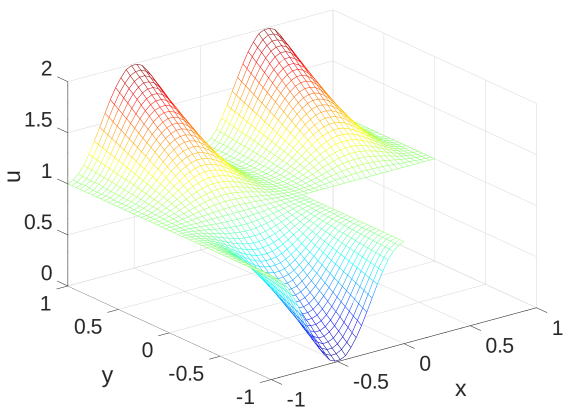


Figure 27: A function evaluated on a uniform grid

Examples are given in Sections 5.3, 8.3, 8.5, 8.8, 8.10 and 8.13.

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**[6]. Due to a recent change in *Octave* the script versions

[6]Function on the boundary are actually called for one point at a time, but this might change. Thus it is advisable to write all functions vectorized.

should use a dummy second argument[7]. 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 8 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.

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

[7] In the script files (`FEMEquation.m` and similar) the function is called with the nodes 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)=....`

4.4 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 of (2)

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

may be solved by

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

Find examples in Sections 3.1.1, 3.1.2, 3.1.3, 8.4, 8.5, 8.7 and 8.13.

4.4.2 General elliptic problems: `BVP2D()`

Equations given in the form of (1)

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

may be 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)

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

4.5 Solving eigenvalue problems: `BVP2Deig()`

To solve an eigenvalue problem of the form (3)

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

use

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

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,Evev]`) 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 the 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 6.9.
    - 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 the the error is always larger than machine accuracy permits.
- The integer parameter `nVec` indicated the number of smallest eigenvalues to be be computed.
- The parameter `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 10^{-5} is used.

An example of an eigenvalue problem is given in Section 3.2.

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

   determine the smallest 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 is the mesh describing the domain and the boundary types
     * A,B0,W,GN2 are 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.
     * NVEC is the number of smallest eigenvalues to be computed
   return values:
     * EVAL is the vector with the eigenvalues
     * EVEC is the matrix with the eigenvectors as columns
     * ERORBOUND is a matrix with error bound of the eigenvalues
```

In Sections 6.8.2 and 6.8.4 find the consequences of the eigenvalues to solutions of dynamic heat and wave equations.

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

To solve an initial boundary value problem (IBVP) of the form (4)

$$\begin{aligned} \rho\,\frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\ u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\ \vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\ u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0 \end{aligned}$$

use the command `IBVP2D()`. Find an example in Section 3.3 and a description of the algorithm in Section 6.8.1.

IBVP2D()

```
[U,T] =  IBVP2D(MESH,M,A,B0,BX,BY,F,GD,GN1,GN2,U0,T0,TEND,STEPS)
   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,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 a 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 results returned.
        If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
   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. The structure of `IBVP2Dsym()` is almost identical to `IBVP2D()`.

IBVP2Dsym()

```
IBVP2Dsym(MESH,M,A,B0,F,GD,GN1,GN2,U0,T0,TEND,STEPS)
   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
 ...
```

4.7 Solving hyperbolic problems: `I2BVP2D()`

Examine an IBVP (6) of hyperbolic type.

$$\begin{aligned}
\rho\,\frac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\
u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0 \\
\frac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0
\end{aligned}$$

To solve this wave type equation use the command `I2BVP2D()`. Find examples in Sections 8.2 and 8.12 and a description of the algorithm in Section 6.8.3.

I2BVP2D()

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

   Solve an 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 a 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 results returned.
        If STEPS = [n,nint], then n*nint steps are taken and (n+1) results returned.
   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.
```

4.8 Solving plane stress and plane strain problems: `PlaneStress()`, `PlaneStrain()`

For a plane stress problem the total energy in expression (14)

$$U(\vec{u}) \quad = \quad U_{elast} + U_{Vol} + U_{Surf}$$

$$= \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \, dA -$$
$$- \iint\limits_{\Omega} \vec{f} \cdot \vec{u} \, dA - \int_{\Gamma_2} \vec{g}_N \cdot \vec{u} \, ds \; .$$

is minimized, respecting the boundary conditions (7), i.e.

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

PlaneStress()

```
[U1,U2] = PlaneStress(MESH,E,NU,F,GD,GN)
   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
     * 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
     * U1 vector with the values of the x-displacement at the nodes
     * U2 vector with the values of the y-displacement at the nodes
```

For a plane strain problem the total energy in expression (52)

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

$$-\iint\limits_{\Omega} \vec{f}\cdot\vec{u}\,dA - \int_{\Gamma_2} \vec{g}_N\cdot\vec{u}\,ds\ .$$

is minimized, again respecting the boundary conditions (7). Thus the code for `PlaneStrain()` is almost identical to `PlaneStress()` .

PlaneStrain()

```
[U1,U2] = PlaneStrain(MESH,E,NU,F,GD,GN)
     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
     * 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
     * U1 vector with the values of the x-displacement at the nodes
     * U2 vector with the values of the y-displacement at the nodes
```

4.8.1 Evaluating plane stress and plane strain solutions

In Table 6 find the commands related to solving plane elasticity problems and analyzing their solutions. Observe the functions `EvaluateStress()`, `EvaluateStrain()`, `EvaluateVonMises()`, `EvaluateTresca()` and `EvaluatePrincipalStress()` find the values at the nodes of the mesh. Thus for many applications these function have to be followed by a call of `FEMgriddata()` to evaluate at arbitrary points.

Observe that the two computational paths

1. Evaluate the partial derivative $\frac{\partial u_1}{dx}$ by a piecewise interpolation of the values of u_1 at the nodes.
2. Evaluate the normal strain ε_{xx} at the nodes, followed by a piecewise interpolation to determine the value at the arbitrary point (x, y).

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 to find the value of the partial derivative $\frac{\partial u_1}{\partial x}$ at the point (x, y).
2. The second option `eps_xx_2` will first find values of the strain ε_{xx} at 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 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)
```

command	purpose
`PlaneStress()`	solve a plane stress problem
`PlaneStrain()`	solve a plane strain problem
`PStressEquationM()`	script to generate plane stress equations, order 1
`PStressEquationQuadM()`	script to generate plane stress equations, order 2
`PStressEquationCubicM()`	script to generate plane stress equations, order 3
`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

Table 6: Commands to solve and examine plane elasticity problems

4.8.2 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. 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 $\varepsilon_{zz} = 0$.

EvaluateStrain()

```
[EPS_XX,EPS_YY,EPS_XY] = EvaluateStrain(MESH,U1,U2)

  evaluate the normal and shearing strains at the nodes

  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
  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. 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)

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
  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
```

4.8.3 Evaluation of stress expressions: `EvaluateVonMises()`,`EvaluatePrincipalStress()` and `EvaluateTresca()`

There are many expressions used for post processing elasticity problems. The following commands 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 principals 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.

$$\begin{aligned}\sigma_M^2 &= \frac{1}{2}\,\left((\sigma_x-\sigma_y)^2+(\sigma_y-\sigma_z)^2+(\sigma_z-\sigma_x)^2\right)+3\,\left(\tau_{xy}^2+\tau_{yz}^2+\tau_{zx}^2\right)\\ &= \frac{1}{2}\,\left((\sigma_x-\sigma_y)^2+\sigma_y^2+\sigma_x^2\right)+3\,\tau_{xy}^2\ =\ \sigma_x^2+\sigma_y^2-\sigma_y\sigma_x^2+3\,\tau_{xy}^2\end{aligned}$$

- 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 the three arguments σ_x, σ_y and τ_{xy} are given, then a plane stress situation is used.
- If the 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.

$$\begin{aligned} 0 &= \det \begin{bmatrix} \sigma_x - \sigma & \tau_{xy} \\ \tau_{xy} & \sigma_y - \sigma \end{bmatrix} = \sigma^2 - \sigma\,(\sigma_x + \sigma_y) + \sigma_x \sigma_y - \tau_{xy}^2 \\ \sigma_{1,2} &= \frac{1}{2}\left((\sigma_x + \sigma_y) \pm \sqrt{(\sigma_x + \sigma_y)^2 - 4\,\sigma_x\sigma_y + 4\,\tau_{xy}^2}\right) \\ &= \frac{1}{2}\left((\sigma_x + \sigma_y) \pm \sqrt{(\sigma_x - \sigma_y)^2 + 4\,\tau_{xy}^2}\right) \end{aligned}$$

  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 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 the three input arguments σ_x, σ_y and τ_{xy} are given, then a plane stress situation is used.
- If the 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
```

4.9 Solving axisymmetric elasticity problems, `AxiStress()`

By minimizing the energy given by equation (18) on page 17 an axisymmetric elasticity problem can be solved. The construction of the elements is shown in Section 7.6 starting on page 146.

The commands to solve axially symmetric problems and analyze their solutions are shown in Table 7.

4.9.1 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 problem
`AxiStressEquationM()`	script to generate the equations, order 1
`AxiStressEquationQuadM()`	script to generate the equations, order 2
`AxiStressEquationCubicM()`	script to generate the equations, 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

Table 7: Commands to solve and examine axially symmetric elasticity problems

4.9.2 Evaluation of strains and stress for axisymmetric problems

Based on Section 2.9 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 4.8.1 starting on page 58.

- **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
```

4.10 Internal commands in FEMoctave

In this section a few internal commands are documented. Usually these commands are not used when solving boundary value problems or elasticity problems. But the 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 96. They can also be useful to illustrate the essential steps of finite element algorithms, e.g. individual element stiffness matrices.

4.10.1 Linear elements: `FEMEquation.cc` and `FEMEquation.m`

This is the fundamental function that transforms a BVP to a system of linear equations. First order triangular elements are used. To speed it up it is written in C++, leading to the file `FEMEquation.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 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.
   return values:
     * A, B: matrix and vector for the linear system to be solved, A*u-B=0
```

The script function `FEMEquation.m` perfoms the same task and is easier to read and understand, but considerably slower than the compiled code.

4.10.2 Quadratic elements: `FEMEquationQuad.cc` and `FEMEquationQuad.m`

This is the fundamental function that transforms 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.
  return values:
    * A, B: matrix and vector for the linear system to be solved, A*u-B=0
```

The script function `FEMEquationQuad.m` performs the same task and is easier to read and understand, but considerably slower than the compiled code.

4.10.3 Cubic elements: `FEMEquationCubic.cc` and `FEMEquationCubic.m`

These two commands are very similar to the above section, but use triangular elements of order 3.

4.10.4 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 $\mathbf{wMat}\,\vec{f}$.

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()`.

4.10.5 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 the only recompute the changes for different `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
```

4.10.6 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 a mxn matrix, where n is the number of eigenvalues desired
     it contains the initial eigenvectors for the iteration
 tol is the relative error, used as the stopping criterion

 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()
```

4.10.7 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 7 (starting on page 137) and easier to read and understand than C++ code

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
```

For axially symmetric problems similar commands are used, i.e. 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
```

The *Octave* codes might be replaced by a compiled code for speed reasons.

4.11 External programs

To construct nonuniform triangular meshes FEMoctave uses an external program.

- **Triangle** to generate a good mesh. The source code is given in FEMoctave. Find documentation on the web page www.cs.cmu.edu/∼quake/triangle.html.
- **CuthillMcKee** to obtain a good numbering. Not necessary any more, since the sparse factorizations 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 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 solution of the boundary value problem

$$\begin{array}{rcll} -\nabla \cdot \nabla u & = & -2\,\sin(x) \cdot \sin(y) & \text{for} \quad 0 \leq x, y \leq 1 \\ \frac{\partial\, u(x,1)}{\partial y} & = & -\sin(x) \cdot \cos(1) & \text{for} \quad 0 \leq x \leq 1 \quad \text{and} \quad y = 1 \\ u(x,y) & = & u_e(x,y) & \text{on the other sections of the boundary} \end{array} .$$

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 6.7. A double logarithmic plot leads to Figure 28.

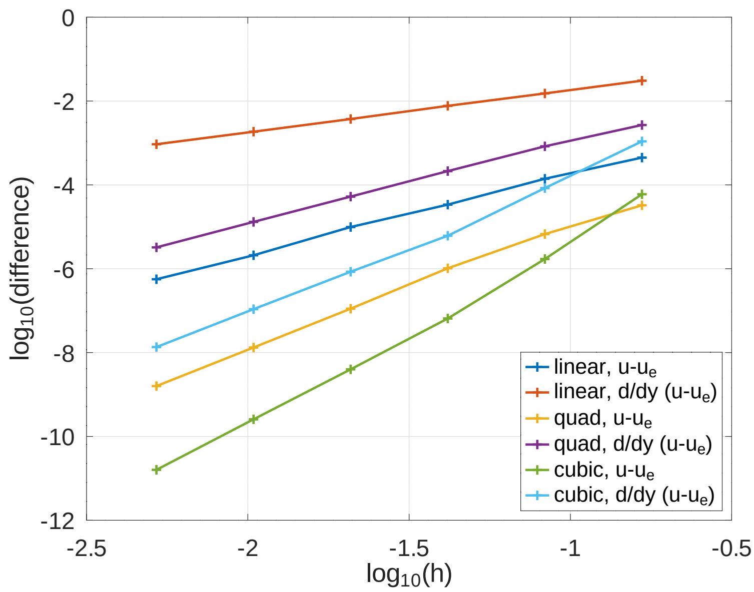


Figure 28: Convergence results for linear, quadratic and cubic elements

- For linear 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 $\frac{\partial}{\partial y}\,(u(x,y) - u_e(x,y))$ 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 $\frac{\partial}{\partial y}(u(x,y) - u_e(x,y))$ 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 $\frac{\partial}{\partial y}(u(x,y) - u_e(x,y))$ 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 6.7 on page 130. It is rather obvious from Figure 28 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 29.

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
```

$$\mathbf{A} = \frac{\sqrt{3}}{6}\begin{bmatrix} +2 & -1 & -1 \\ -1 & +2 & -1 \\ -1 & -1 & +2 \end{bmatrix}$$

$$\vec{b} = \frac{\sqrt{3}}{4\cdot 3}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} = \frac{\text{area of triangle}}{3}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

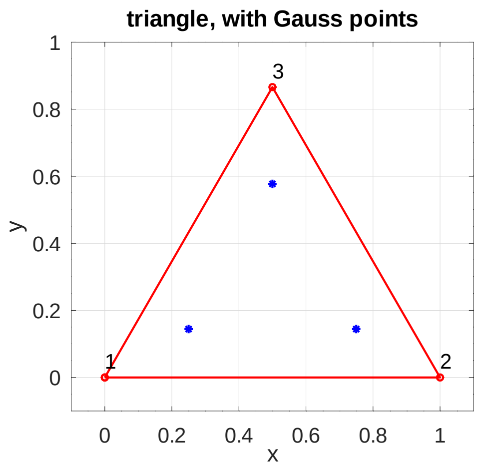


Figure 29: An linear, equilateral triangle, the Gauss integration points and the element stiffness matrix

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 29.

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 30. Then examine the linear equation corresponding to an interior point at (x_i, y_i).

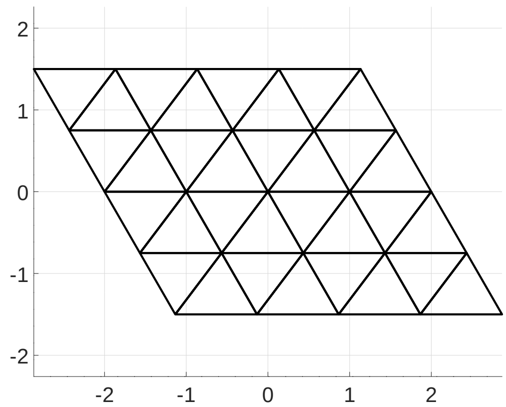


Figure 30: Uniform meshes consisting of equilateral triangles

- 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 29, 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 29, 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).

$$\frac{1}{h^2}\left(\frac{6}{\sqrt{3}}\,u(x_i,y_i) - \frac{1}{\sqrt{3}}\sum_{\text{neigbours}} u(x_j,y_j)\right) = +\frac{\sqrt{3}}{2}\,f$$

$$\frac{1}{h^2}\left(u(x_i,y_i)-\frac{1}{6}\sum_{\text{neigbours}} u(x_j,y_j)\right) \quad = \quad +\frac{1}{4}\,f$$

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 31. The

$$\mathbf{A} = \frac{\sqrt{3}}{18}\begin{bmatrix} 6 & 1 & 1 & 0 & -4 & -4 \\ 1 & 6 & 1 & -4 & 0 & -4 \\ 1 & 1 & 6 & -4 & -4 & 0 \\ 0 & -4 & -4 & 24 & -8 & -8 \\ -4 & 0 & -4 & -8 & 24 & -8 \\ -4 & -4 & 0 & -8 & -8 & 24 \end{bmatrix}$$

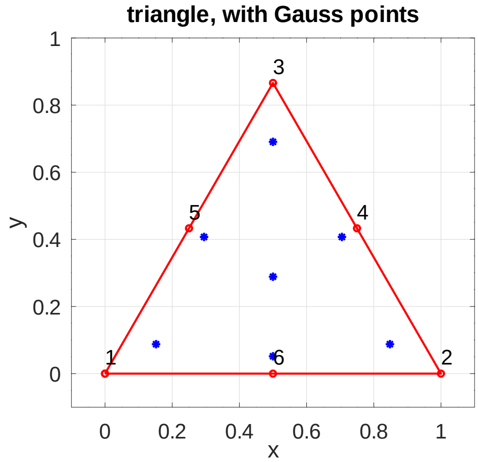


Figure 31: An equilateral, quadratic triangle, the Gauss integration points and the element stiffness matrix

vector 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 30

- 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 equilateral triangle with the help of `CeateMeshTriangle`. The code in `CreateTriangle.m` generates the mesh and Figure 32. 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
```

$$\mathbf{A} = \begin{bmatrix} +1 & -0.5 & -0.5 \\ -0.5 & +1 & 0 \\ -0.5 & 0 & +1 \end{bmatrix} \quad , \quad \vec{b} = \frac{1}{6}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

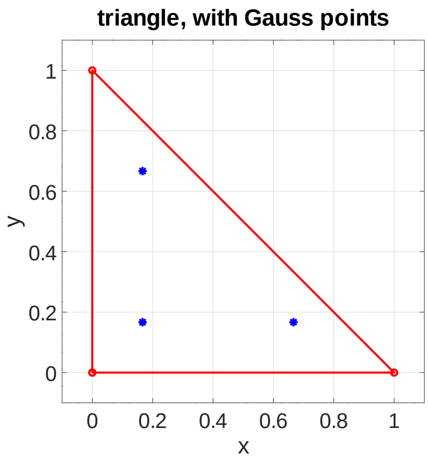
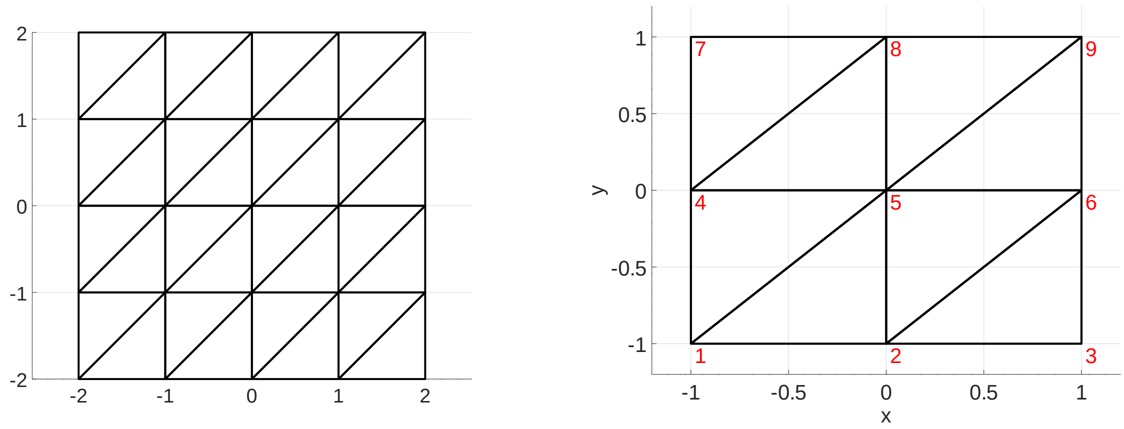


Figure 32: A right triangle, the Gauss integration points and the element stiffness matrix

Based on elements of the above type there is a connection of FEM to the finite difference method. Generate a rectangular grid, shown in Figure 33. Examine 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 equation $\mathbf{A}\,\vec{u} + \vec{b}$ 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) a section (b) a small section, numbered

Figure 33: Uniform meshes consisting of rectangular triangles

```
a=1; b0=bx=by= 0; f=pi;
[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 34. The vector is given by

$$\vec{b} = \frac{1}{2\cdot 3}\begin{pmatrix} 0\\ 0\\ 0\\ -1\\ -1\\ -1 \end{pmatrix} = \frac{\text{area of triangle}}{3}\begin{pmatrix} 0\\ 0\\ 0\\ -1\\ -1\\ -1 \end{pmatrix}.$$

$$\mathbf{A} = \frac{1}{6}\begin{bmatrix} 6 & 1 & 1 & 0 & -4 & -4 \\ 1 & 3 & 0 & 0 & 0 & -4 \\ 1 & 0 & 3 & 0 & -4 & 0 \\ 0 & 0 & 0 & 16 & -8 & -8 \\ -4 & 0 & -4 & -8 & 16 & 0 \\ -4 & -4 & 0 & -8 & 0 & 16 \end{bmatrix}$$

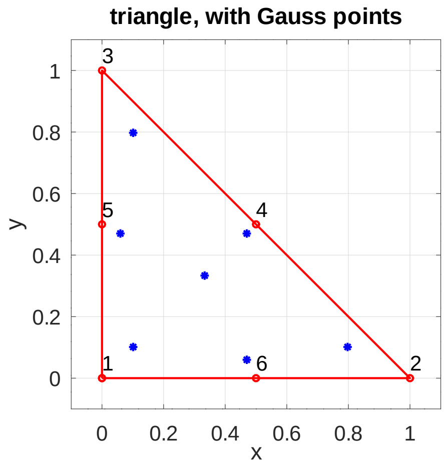


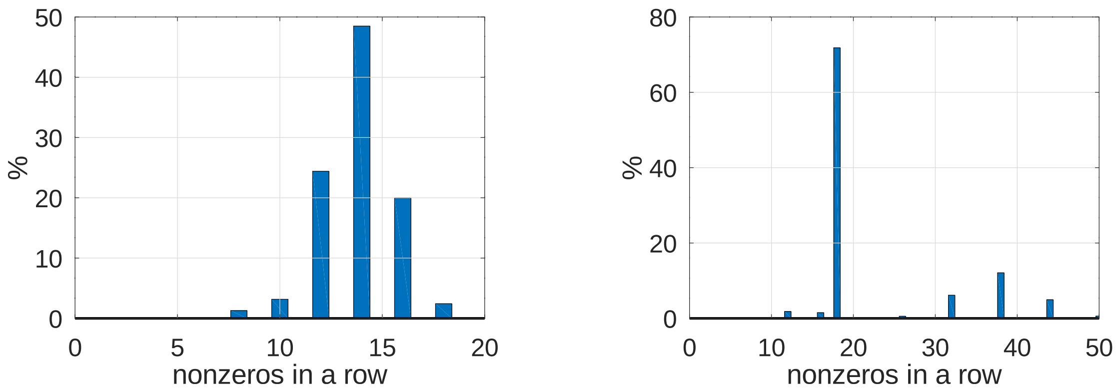
Figure 34: A right angle triangle, the Gauss integration points and the element stiffness matrix

5.2.3 Element stiffness matrices for elasticity problems

For the equilateral triangle in Figure 31 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 30 (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 35(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 35: 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 30 (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 35(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

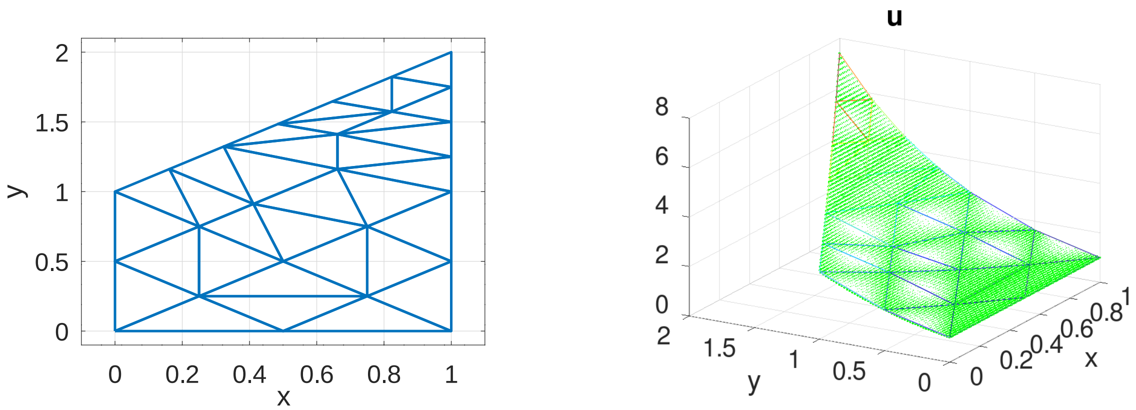
To examine the behavior of a solution within each of the triangular elements use the boundary value problem

$$\begin{array}{rcllc} -\Delta u & = & -\exp(y) & \text{for} & (x,y) \in \Omega \\ u(x,y) & = & \exp(y) & \text{for} & (x,y) \in \Gamma \end{array} .$$

on the domain Ω displayed in Figure 36(a). The exact solution is given by $u(x, y) = \exp(y)$, shown in Figure 36(b). The problem is solved twice:

1. using 32 triangular elements of order 1.
2. using 8 triangular elements of order 2.

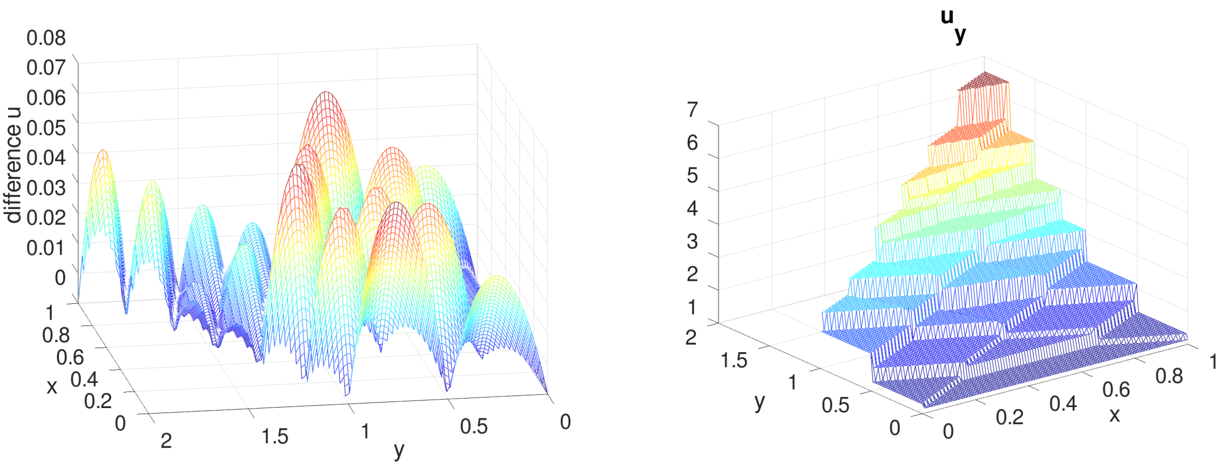
The nodes used coincide for the two approaches, i.e four triangles in Figure 36(a) for the linear elements correspond to one of the eight triangles for the quadratic elements.



(a) the mesh (b) the solution

Figure 36: The mesh and the solution for a BVP

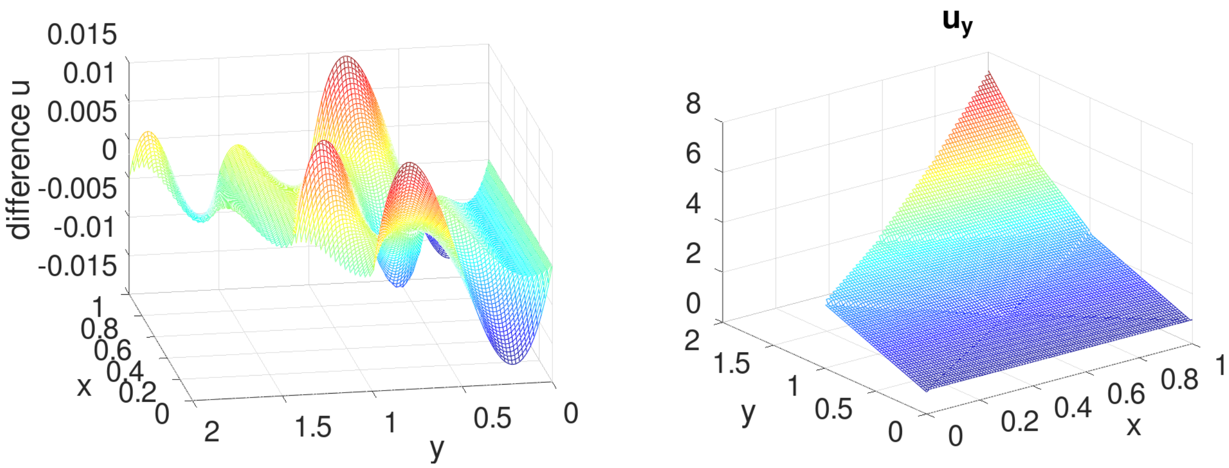
Figure 37(a) shows the difference of the computed solution with first order elements to the exact solution. Within each of the 32 elements the difference is not too far from a quadratic function. Figure 37(b) shows the values of the partial derivative $\frac{\partial u}{\partial y}$. It is clearly visible 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 $\frac{\partial u}{\partial y}$

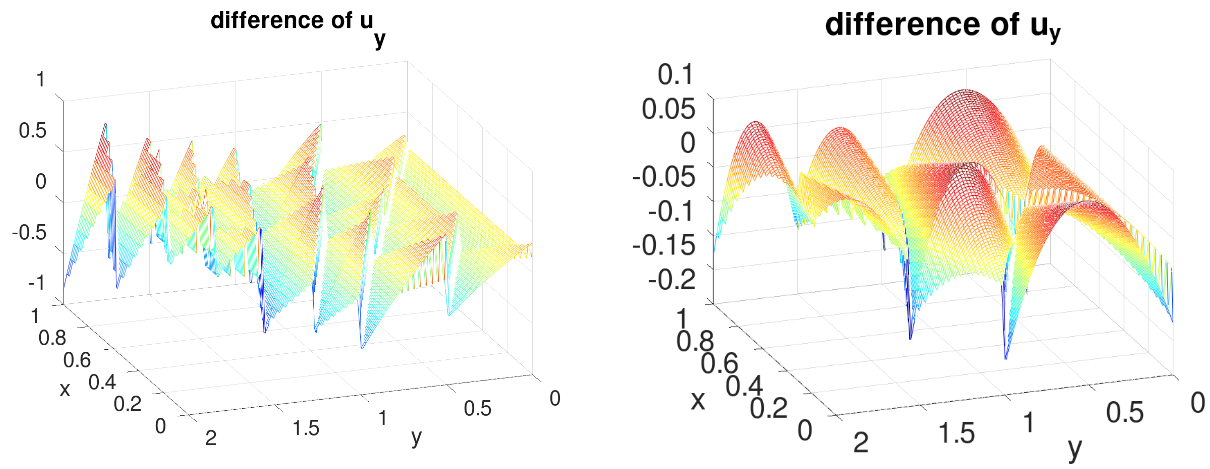
Figure 37: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a first order mesh

Figure 38(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 38(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 39 shows the errors for the partial derivative $\frac{\partial u}{\partial y}$ and confirms this observation.



(a) the difference to the exact solution (b) the values of $\frac{\partial u}{\partial y}$

Figure 38: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a second order mesh



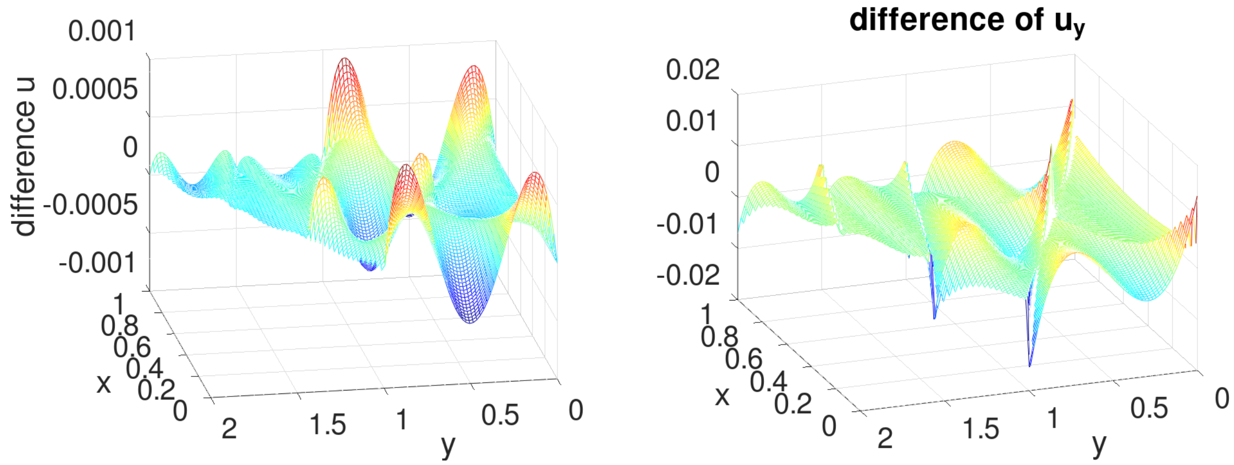
(a) using linear elements (b) using quadratic elements

Figure 39: Difference of the approximate values of $\frac{\partial u}{\partial y}$ to the exact values

In Figure 40 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 $\frac{\partial u}{\partial y}$ is not continuous across the limits of the triangles, since these third order elements are not c^1–conforming.

FEMInsideElement

```
N = 2; MeshType = 'quadratic'  %% use 'linear', 'quadratic' or 'cubic'
```



(a) $u - u_{exact}$ (b) $\frac{\partial}{\partial y}\,(u - u_{exact})$

Figure 40: Difference of the approximate values of u and $\frac{\partial u}{\partial y}$ to the exact values for cubic elements

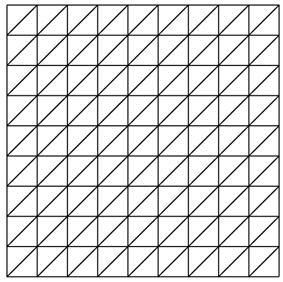
```
Mesh = CreateMeshTriangle('test',[0 0 -1;1 0 -1;1 2 -1; 0 1 -1],1/N^2);
switch MeshType
  case 'quadratic'
    Mesh = MeshUpgrade(Mesh,'quadratic');
  case 'cubic'
    Mesh = MeshUpgrade(Mesh,'cubic');
endswitch

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.

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.

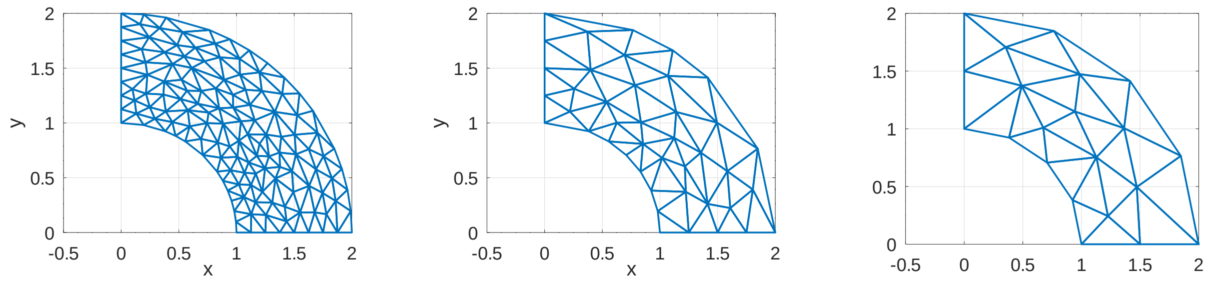
This points to about a factor of $\frac{11.5}{7} \approx 1.6$ more nonzero entries in the matrix for quadratic elements for the same number of degrees of freedom. For cubic elements expect a factor of $\frac{17}{7} \approx 2.4$. This implies that the computational effort is larger, the actual effect 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

$$\begin{array}{rcll} -\nabla\left((1+x^2)\nabla u(x,y)\right) &=& -4\,(1+x^2)\,\exp(-2\,y) & \text{for} \quad (x,y)\in\Omega \\ \frac{\partial\,u(y,0)}{\partial x} &=& 0 & \text{for} \quad 1\leq y\leq 2 \\ u(x,y) &=& \exp(-2\,y) & \text{on other sections of the boundary} \end{array} .$$

on the domain shown in Figure 41. The exact solution is given by $u_e(x,y) = \exp(-2\,y)$. For different values of



(a) linear elements (b) quadratic elements (c) cubic elements

Figure 41: Meshes for linear, quadratic and cubic elements, leading to similar size linear systems to be solved.

the 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 41. 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.

$$\text{error} = \left(\iint\limits_{\Omega} |u(x,y)-u_e(x,y)|^2\;dA\right)^{1/2} .$$

- 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 6.7 on page 130.
- 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 8 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 8: Results for elements of order 1, 2 and 3

5.6 Shear locking of 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 $\pm c\, y = \pm 5 \cdot 10^{-4}\, y$. Use the material parameters $E = 100 \cdot 10^9$ and $\nu = 0$. Find the original and deformed domain in Figure 42. One can verify[8] that an exact solution of the

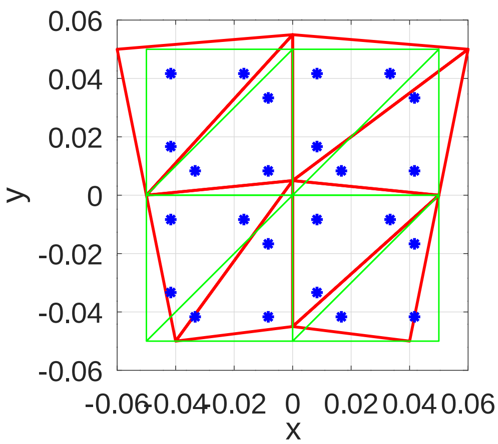


Figure 42: Original and deformed domain and the Gauss integration points for linear elements

[8]E.g. use [Stah08, §5] with $\nu = 0$ and $u_1 = x\,y$ and $u_2 = -\frac{x^2}{2}$ to arrive at

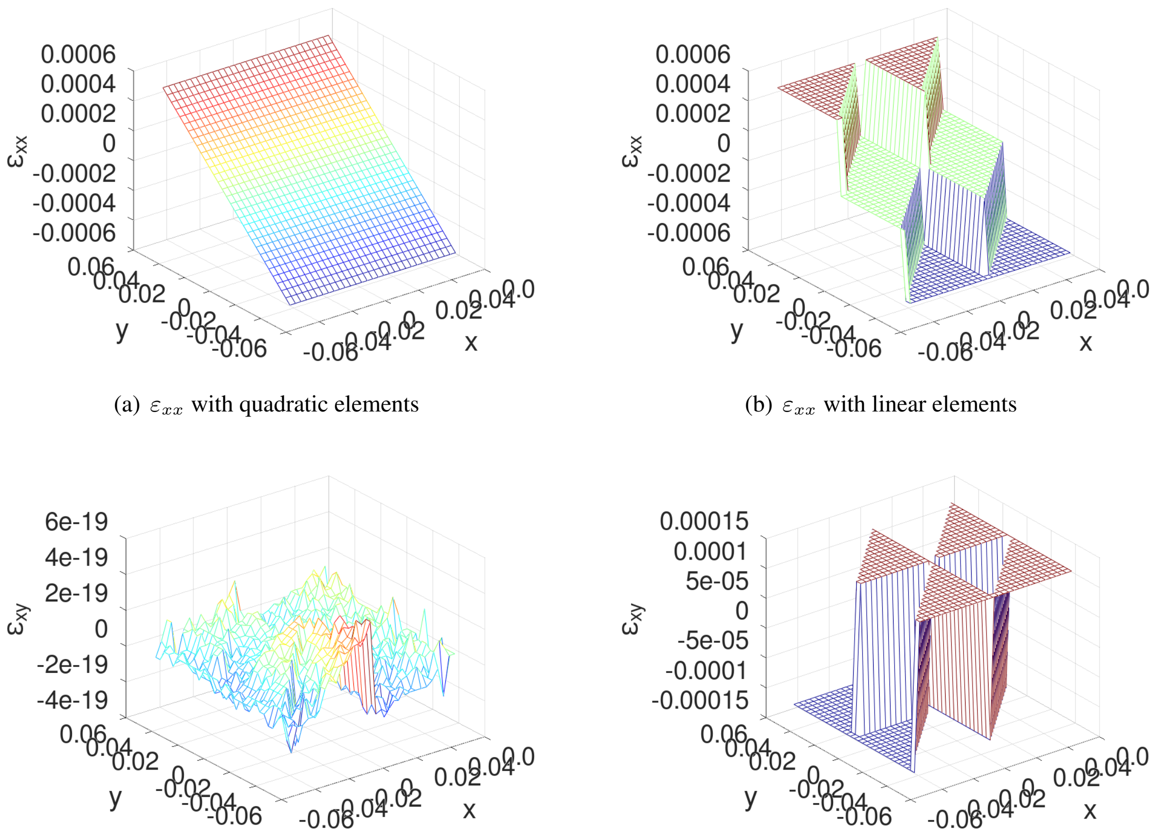
$$0 \stackrel{?}{=} \frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} + \frac{\partial}{\partial x}\left(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}\right) = +0 + 0 + \frac{\partial}{\partial x}\,(y + 0) \qquad \text{OK}$$

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 $\varepsilon_{xx} = \frac{2\,c}{L}\,y$ and $\varepsilon_{yy} = \varepsilon_{xy} = 0$ and thus the elastic energy (use the elastic energy density (13) with $\nu = 0$)

$$\begin{aligned} U_{elast} &= U_{\varepsilon_{xx}} + U_{\varepsilon_{yy}} + U_{\varepsilon_{xy}} \\ &= \frac{E}{2}\iint\limits_{\Omega} \varepsilon_{xx}^2\,dA + \frac{E}{2}\iint\limits_{\Omega} \varepsilon_{yy}^2\,dA + \frac{E}{2}\iint\limits_{\Omega} 2\,\varepsilon_{xy}^2\,dA \\ &= \frac{E}{2}\int_{-H/2}^{+H/2}\int_{-L/2}^{+L/2} \frac{4\,c^2}{L^2}\,y^2\,dx\,dy + 0 + 0 = \frac{E}{2}\,\frac{4\,c^2}{L^2}\,L\,\frac{2\,H^3}{3\,8} = \frac{125}{3} \approx 41.667 \end{aligned}$$



(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 43: The strains ε_{xx} and ε_{xy} with two layers in each direction for linear and quadratic elements

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()`

$$0 \stackrel{?}{=} \frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} + \frac{\partial}{\partial y}\left(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}\right) = -1 + 0 + \frac{\partial}{\partial y}\,(y+0) \qquad \text{OK}$$

to evaluate the strains ε_{xx}, ε_{yy} and ε_{xy} on a fine xy–grid. Find the results for two layer (NL=NH=2) in Figure 43. 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 $\varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}\right)$, 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.
3. Use the command `mesh()` to visualize a few strains, leading to Figure 43.
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 9. 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 9: Elastic energy contributions for shearing

- The results generated be 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 89), where an exact result is obtained.

- The results based on linear elements are severely different. The elastic energy is considerably to 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);
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 on the inside of 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

$$\begin{aligned}
\iint\limits_{\Omega} \mathrm{W} &= \iint\limits_{\Omega} \mathrm{Wxx} + \mathrm{Wyy} + \mathrm{Wxy} + \mathrm{Wxxyy} \\
\iint\limits_{\Omega} \mathrm{Wxx} &= \frac{E}{2\,(1-\nu^2)} \iint\limits_{\Omega} \varepsilon_{xx}^2 \; dA \\
\iint\limits_{\Omega} \mathrm{Wyy} &= \frac{E}{2\,(1-\nu^2)} \iint\limits_{\Omega} \varepsilon_{yy}^2 \; dA \\
\iint\limits_{\Omega} \mathrm{Wxy} &= \frac{E}{2\,(1-\nu^2)} \iint\limits_{\Omega} 2\,(1-\nu)\,\varepsilon_{xy}^2 \; dA
\end{aligned}$$

- 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.

5.7 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

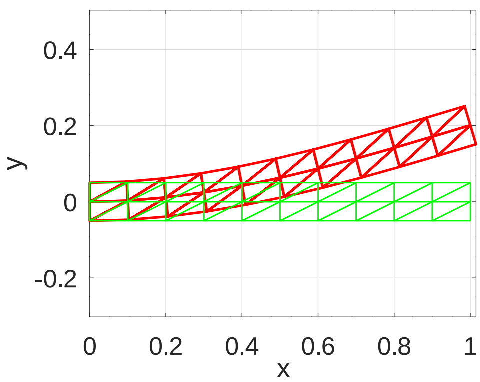


Figure 44: The original shape of the a beam and its (exaggerated) deformed shape, using two layers of elements

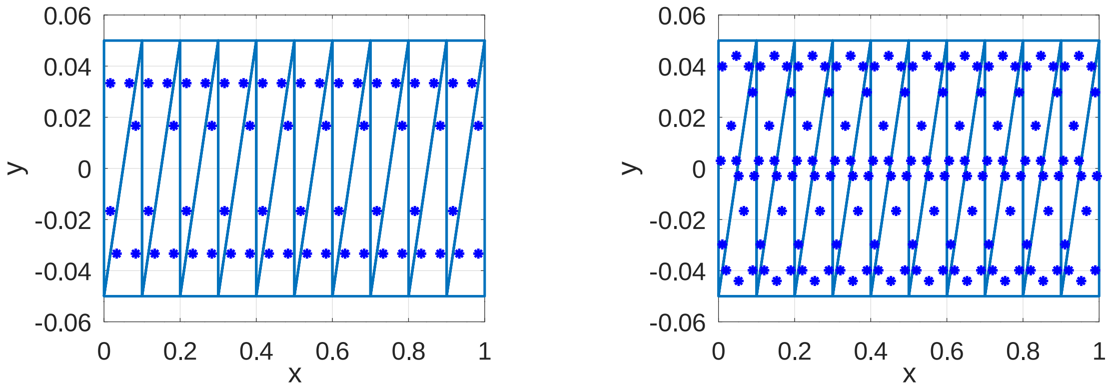
$$
\begin{aligned}
\frac{\partial^2}{\partial x^2}\, u_2(x,y) &= \frac{M}{E\,I} = \frac{F}{E\,I}\,(L-x) \quad , \quad \frac{\partial}{\partial x}\, u_2(x,y) = \frac{F}{E\,I}\,(L\,x - \frac{1}{2}\,x^2) \\
u_2(x,y) &= \frac{F}{E\,I}\,(\frac{L}{2}\,x^2 - \frac{1}{6}\,x^3) \\
u_1(x,y) &= -y\,\frac{\partial}{\partial x}\, u_2(x,y) = -\frac{F}{E\,I}\,(L\,x - \frac{1}{2}\,x^2)\,y \\
\varepsilon_{xx}(x,y) &= \frac{\partial\, u_1(x,y)}{\partial x} = -\frac{F}{E\,I}\,(L-x)\,y \quad , \quad \varepsilon_{yy}(x,y) = \frac{\partial\, u_2(x,y)}{\partial y} = 0 \\
\varepsilon_{xy}(x,y) &= \frac{1}{2}\,(\frac{\partial\, u_1}{\partial y} + \frac{\partial\, u_2}{\partial x}) = \frac{F}{E\,I}\,\left(-(L\,x - \frac{1}{2}\,x^2) + (\frac{L}{2}\,2\,x - \frac{1}{2}\,x^2)\right) = 0
\end{aligned}
$$

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.

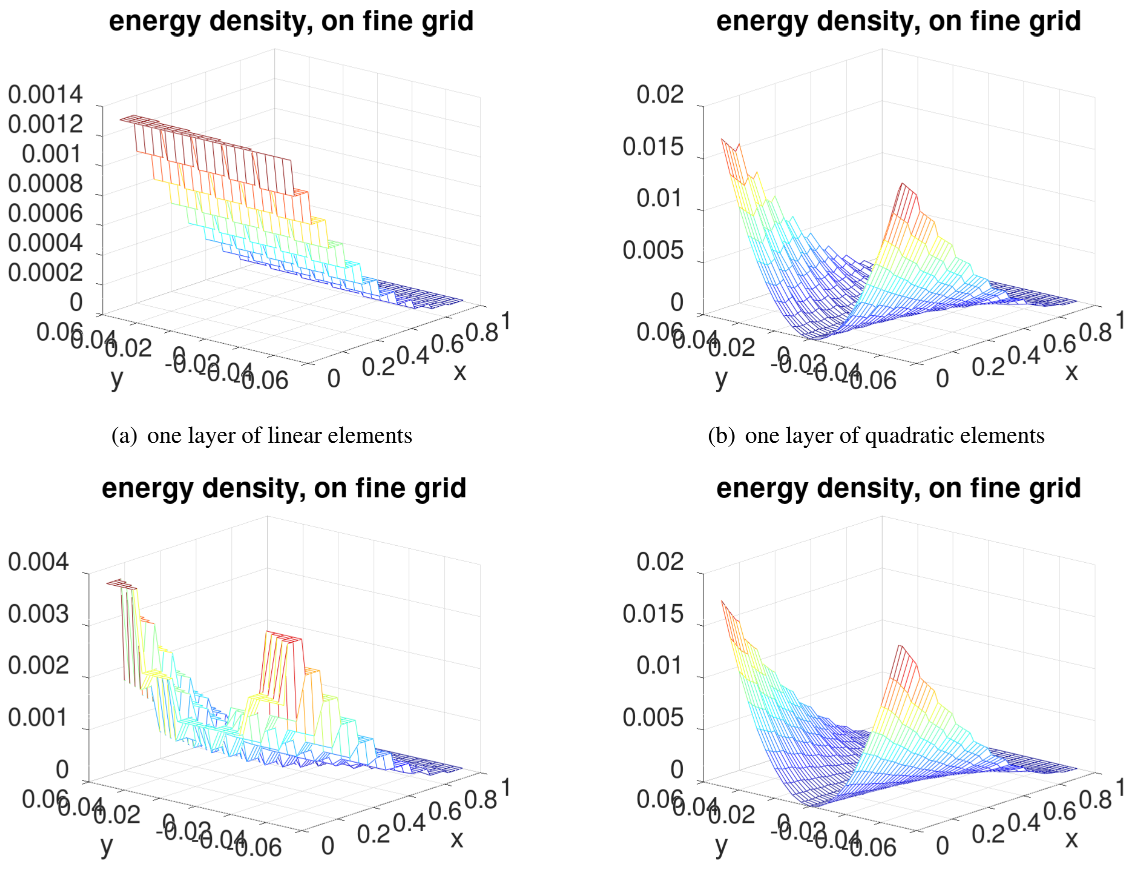
To examine the performance of the FEM algorithms use a rectangular mesh with NL sections along the horizontal x–axis and NH layers in the vertical y–direction. The code is using first, second or third order elements. In Figure 45 find the mesh and the corresponding integration points for meshes with 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 46. Observe the piecewise constant energy density for linear elements, i.e. CST elements.



(a) linear elements (b) quadratic elements

Figure 45: Meshes for linear and quadratic elements with one layer, with the integration points

Multiple runs of the code `BendingBeam.m` lead to the results in Table 10. 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.



(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 46: The elastic energy density of the bending beam with one or five layers

element order	NL	NH	$\max\{u_2\}$	$\max\{\varepsilon_{xx}\}$	energy
Euler beam, exact			$4 \cdot 10^{-6}$	$6 \cdot 10^{-7}$	$2 \cdot 10^{-4}$
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 10: 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 or 2
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
[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);
EnergiesFEMIntegrateGauss = 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);  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);
              hold off; xlabel('x'); ylabel('y'); axis equal
```

5.8 Missing boundary constraints and null spaces

Examine a domain $\Omega \subset \mathbb{R}^2$ and minimize the elastic energy given by equation (14)

$$U(\vec u) = \iint\limits_{\Omega} \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \; dA \;,$$

i.e. no external force 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 prevent rotations.

As a consequence the global stiffness matrix $\mathbf{A}$ should have a three dimensional null space, consisting of constant displacements and rotations. Thus expect three very small eigenvalues. Constant displacements vectors u in x and 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 last expression corresponds to the elastic energy. The code below verifies this on a 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[9]. 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 a regular 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]

%% at point [x,y] add displacement [-y,x]
x = Mesh.nodes(:,1); y = Mesh.nodes(:,2);
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
```

[9] 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*.

```
Norm_Rotation =   3.3145e-07   2.2532e-09
Norm_Random   =   5.1146e+07   5.3317e+06
```

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 flag 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 increase the elastic energy.
- Rotating the intersection of the object with the plane $y = 0$ does deform the 3D object and thus increase 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

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 presentaion are the lecture notes [Stah08]. Find a list of books on FEM in [Stah08, §0].

6.1 Classical solutions and weak solutions

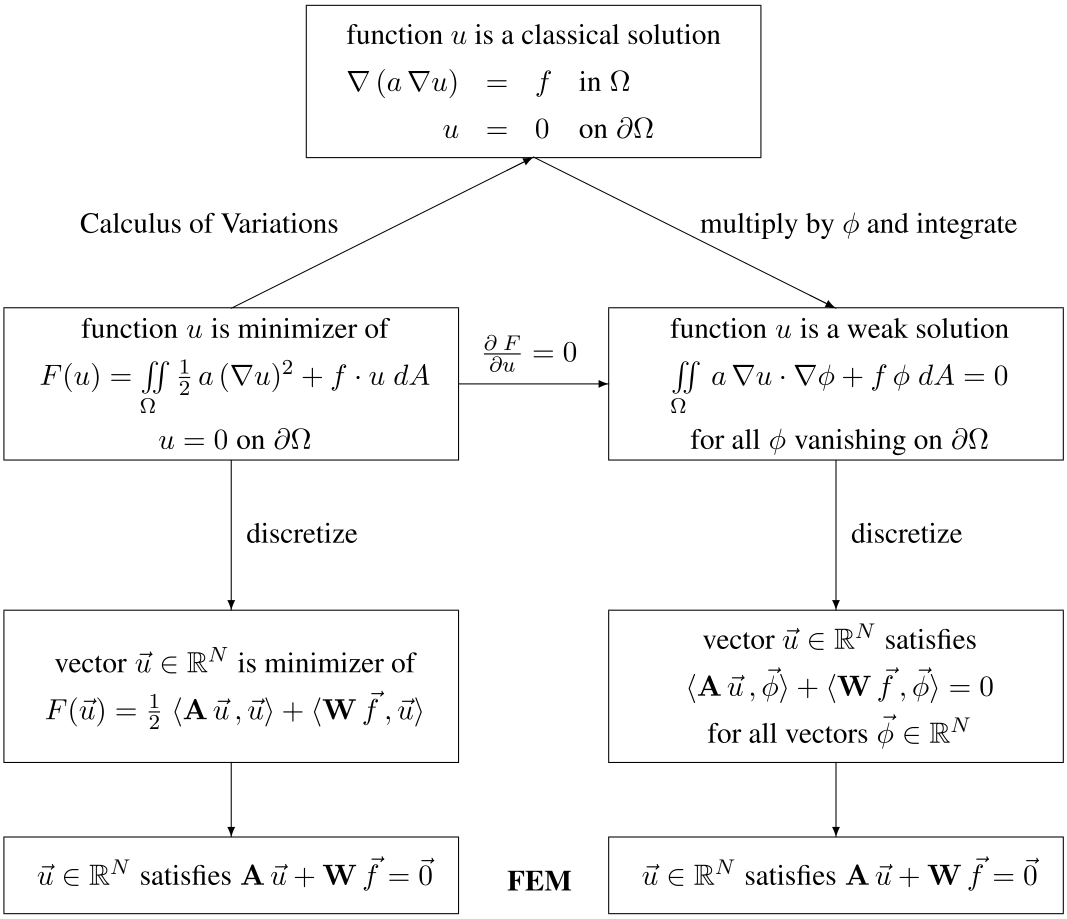


Figure 47: Classical and weak solutions, minimizers and FEM

A function $u = u(x,y)$ is called a **classical solution** of the the BVP (1) iff it is twice differentiable and

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

Multiply this equation with a smooth function ϕ, vanishing on Γ_1, and integrate over the domain Ω to arrive at

$$\begin{aligned} 0 &= -\nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u - f \\ 0 &= \iint_\Omega \phi\,\left(-\nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u - f\right)\,dA \end{aligned}$$

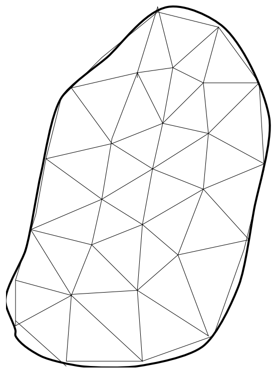
$$\begin{aligned}
&= \iint\limits_{\Omega} \nabla\phi\cdot(a\,\nabla u - u\,\vec{b}) + \phi\;(b_0\,u - f)\;dA - \int_{\Gamma} \phi\;\left(a\,\nabla u - u\,\vec{b}\right)\cdot\vec{n}\;ds \\
&= \iint\limits_{\Omega} \nabla\phi\cdot(a\,\nabla u - u\,\vec{b}) + \phi\;(b_0\,u - f)\;dA - \int_{\Gamma_2} \phi\;(g_2 + g_3\,u)\;ds\,. \qquad (19)
\end{aligned}$$

If a function u satisfies (19) 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

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

among all functions u satisfying the boundary condition $u = g_1$ on Γ_1. Figure 47 shows connections between classical solutions, weak solutions and the resulting system of (linear) equations for the finite element approach. The left branch in Figure 47 illustrates the usage of minimization and calculus of variations in the context of FEM algorithms.

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, using grid points $(x_i, y_i) \in \Omega$, $1 \leq i \leq n$. On each triangle T_k we 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



$$\iint\limits_{\Omega} \ldots\,dA = \sum_k \iint\limits_{T_k} \ldots\,dA\,.$$

The gradients of u and ϕ are replaced by the gradients of the piecewise polynomials. Each contribution has to be written in the form

$$\iint\limits_{T_k} \ldots\,dA = \langle \mathbf{A}_k\vec{u}_k\,,\,\vec{\phi}_k\rangle + \langle \mathbf{W}_k\vec{f}_k\,,\,\vec{\phi}_k\rangle\,,$$

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 48 find a graphical representation of a few commonly used elements.

- A solid dot at a position indicates that the value at this point is used as a DOF.
- 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.

The codes in FEMoctave only use linear, quadratic and cubic elements.

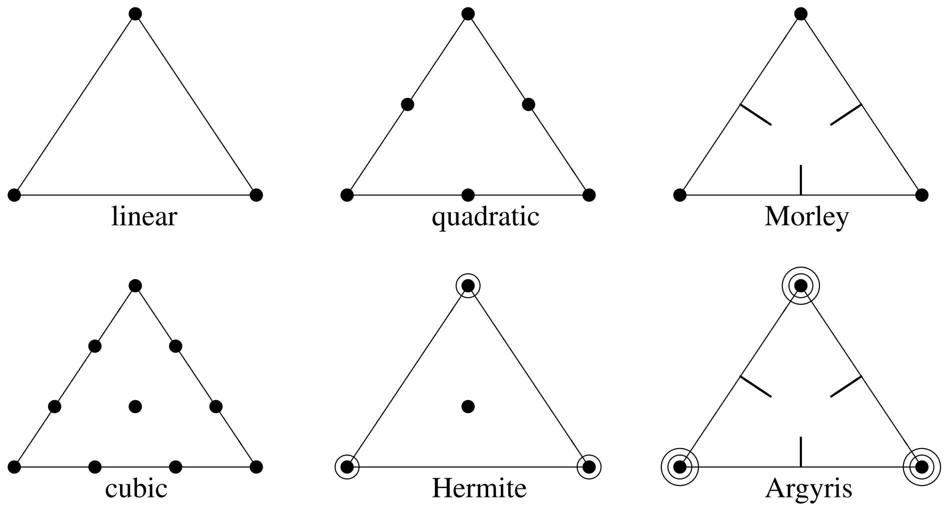


Figure 48: A few triangular elements

	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 11: Properties of triangular elements

6.3 Transformation, 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 49. The transformation is given by

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \nu \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \\ &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} \end{aligned}$$

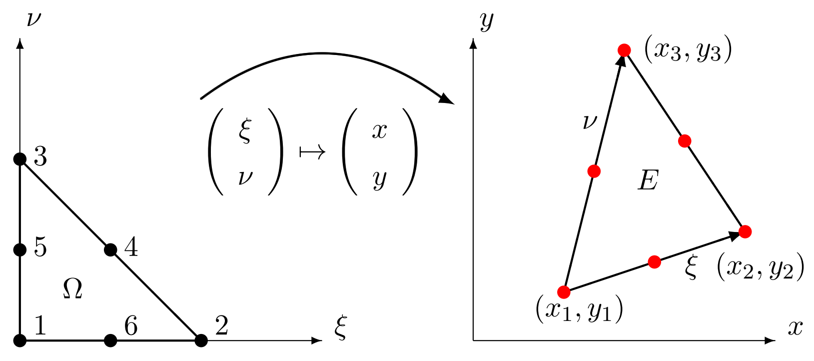


Figure 49: Transformation of the standard triangle Ω to a general triangle E

with the transformation matrix

$$\mathbf{T} = \left[\begin{array}{cc} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{array}\right] .$$

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

$$\left(\begin{array}{c} \xi \\ \nu \end{array}\right) = \mathbf{T}^{-1} \cdot \left(\begin{array}{c} x - x_1 \\ y - y_1 \end{array}\right) = \frac{1}{\det(\mathbf{T})} \left[\begin{array}{cc} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{array}\right] \cdot \left(\begin{array}{c} x - x_1 \\ y - y_1 \end{array}\right) .$$

If a function $f(x, y)$ is to be integrated over the triangle E use the transformation

$$\iint\limits_E f\; dA = \iint\limits_\Omega f\left(\vec{x}\left(\xi, \nu\right)\right) \left|\det\left(\frac{\partial\left(x, y\right)}{\partial\left(\xi, \nu\right)}\right)\right| \; d\xi\, d\nu = |\det(\mathbf{T})| \int_0^1 \left(\int_0^\nu f\left(\vec{x}\left(\xi, \nu\right)\right)\, d\xi\right) d\nu\,. \tag{20}$$

The Jaccobi determinant is given by

$$\left|\det\left(\frac{\partial\left(x, y\right)}{\partial\left(u, v\right)}\right)\right| = |\det(\mathbf{T})| = \left|\left(x_2 - x_1\right)\left(y_3 - y_1\right) - \left(x_3 - x_1\right)\left(y_2 - y_1\right)\right|$$

If the orientation of the triangle is positive, then $\det(\mathbf{T})$ will be positive. Since the area of the standard triangle Ω equals $\frac{1}{2}$ 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.

$$\iint\limits_\Omega f(\vec{\xi})\; dA \approx \sum_{j=1}^m w_j\, f(\vec{g}_j)\,. \tag{21}$$

The integration points and weights have to be chosen, such that the approximation error is as small as possible. Required are two 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 must be inside the triangle (or on the triangle boundary)
- All weights w_i must be positive.

6.3.2 Gauss integration on the standard triangle with 3 Gauss points

In Figure 50 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

$$\iint\limits_{\Delta} f\, dA \approx w\,\left(f(\vec{g}_1) + f(\vec{g}_2) + f(\vec{g}_3)\right)\,.$$

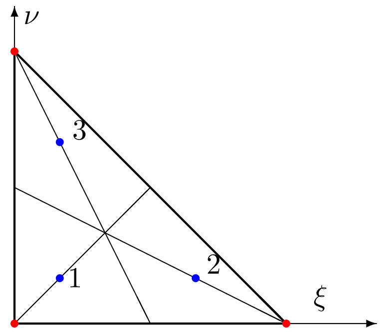


Figure 50: Gauss integration of order 2 on the standard triangle, using 3 integration points

To determine the optimal values determine a solution of a nonlinear system of 2 equations for the unknowns λ and w. Require that ξ^k for $0 \leq k \leq 2$ 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

$$\mathbf{G} = \begin{bmatrix} 1/6 & 1/6 \\ 2/3 & 1/6 \\ 1/2 & 2/3 \end{bmatrix} \quad \text{and} \quad w = \frac{1}{6}\,.$$

For a general triangle the Gauss points are located at

$$\mathbf{X}_G = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \mathbf{G}^T\,.$$

This integration scheme will be used for linear elements.[10]

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 6 integration points where groups of 3 have the same weight. Finally add the midpoint with weight w_3. This is illustrated in Figure 51. 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 $\vec{w}$ with the corresponding integration weights. To determine the optimal

[10] 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.

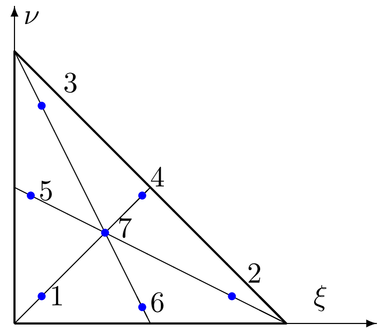


Figure 51: Gauss integration of order 5 on the standard triangle, using 7 integration points

values solve a 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 inside the triangle) and positive weights w_1, w_2 and w_3.

This approximate integration yields the exact results for polynomials f 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 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 the error.

The optimal choice of Gauss points and integration weights is given by[11]

$$\mathbf{G} = \begin{bmatrix} \lambda_1/2 & \lambda_1/2 \\ 1-\lambda_1 & \lambda_1/2 \\ \lambda_1/2 & 1-\lambda_1 \\ \lambda_2/2 & \lambda_2/2 \\ 1-\lambda_2 & \lambda_2/2 \\ \lambda_2/2 & 1-\lambda_2 \\ 1/3 & 1/3 \end{bmatrix} \approx \begin{bmatrix} 0.101287 & 0.101287 \\ 0.797427 & 0.101287 \\ 0.101287 & 0.797427 \\ 0.470142 & 0.470142 \\ 0.059716 & 0.470142 \\ 0.470142 & 0.059716 \\ 0.333333 & 0.333333 \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{pmatrix} w_1 \\ w_1 \\ w_1 \\ w_2 \\ w_2 \\ w_2 \\ w_3 \end{pmatrix} \approx \begin{pmatrix} 0.0629696 \\ 0.0629696 \\ 0.0629696 \\ 0.0661971 \\ 0.0661971 \\ 0.0661971 \\ 0.1125000 \end{pmatrix} . \tag{22}$$

Using the transformation results in this section compute the coordinates $\mathbf{X}_G$ for the Gauss integration in a general triangle by

$$\mathbf{X}_G = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \mathbf{G}^T . \tag{23}$$

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 (19) 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 (19)

$$0 = \iint\limits_{\Omega} \nabla\phi \cdot (a\,\nabla u - u\,\vec{b}) + \phi\;(b_0\,u - f)\;dA - \int_{\Gamma_2} \phi\;(g_2 + g_3\,u)\;ds$$

[11] 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 $w_7 = 9/80$.

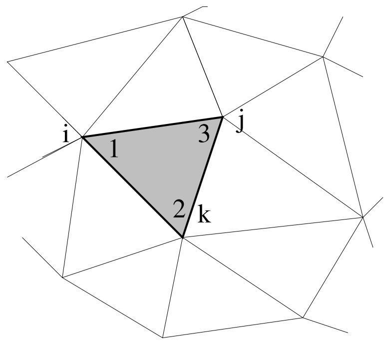
have to be transformed into

$$0 = \langle \vec{\phi}\,,\; \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\rangle\,. \tag{24}$$

By integration over one triangle E find

$$\iint\limits_E \nabla\phi\cdot(a\,\nabla u - u\,\vec{b}) + \phi\;(b_0\,u - f)\;dA \approx \langle \begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\mathbf{A}_E\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle + \langle\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\mathbf{W}_E\vec{f}_E\rangle\,.$$

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 52. 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 $\mathbf{A}$.



local	$\longleftrightarrow$	global
triangle	$\longleftrightarrow$	mesh
1	$\longleftrightarrow$	i
2	$\longleftrightarrow$	k
3	$\longleftrightarrow$	j

Figure 52: Local and global numbering of nodes

$$\mathbf{A}_E = \begin{bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix} \longrightarrow \quad \mathbf{A} = \mathbf{A} + \begin{array}{cc} & \begin{array}{ccccccc} & \text{col i} & & \text{col j} & & \text{col k} & \end{array}\\ \begin{array}{c} \\ \text{row i}\\ \\ \text{row j}\\ \\ \text{row k}\\ \\ \end{array} & \begin{bmatrix} \ddots & \vdots & & \vdots & & \vdots & \\ \cdots & a_{11} & \cdots & a_{13} & \cdots\cdots & a_{12} & \cdots\\ & \vdots & \ddots & \vdots & & \vdots & \\ \cdots & a_{31} & \cdots & a_{33} & \cdots\cdots & a_{32} & \cdots\\ & \vdots & & \vdots & \ddots & \vdots & \\ \cdots & a_{21} & \cdots & a_{23} & \cdots\cdots & a_{22} & \cdots\\ & \vdots & & \vdots & & \vdots & \ddots \end{bmatrix}\end{array}$$

Similar procedures have to be appplied to the vectors.

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 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

$$\begin{aligned} \iint\limits_E f\,\phi\,dA &\approx w\;2\,\text{area}(E)\,\left(f_1\,\phi(\vec g_1) + f_2\,\phi(\vec g_2) + f_3\,\phi(\vec g_3)\right) \\ &= \frac{2\,\text{area}(E)}{6}\,\langle \mathbf{M}\,\vec\phi\,,\,\vec f\rangle = \frac{\text{area}(E)}{3}\,\langle\vec\phi\,,\,\mathbf{M}^T\vec f\rangle\,. \end{aligned}$$

  Thus find one contribution to (24).

- 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 (24).

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 may be simplified to

$$\iint\limits_E b_0\, u\, \phi\; dA \approx b_0\, \frac{\text{area}(E)}{12}\,\langle \vec\phi\,,\, \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2\end{bmatrix}\,\vec u\rangle\,.$$

Thus find another contribution to (24).

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 gradients are 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[12] 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. Find

$$\nabla u = \begin{pmatrix} \frac{d\,u}{\partial x} \\ \frac{d\,u}{\partial y} \end{pmatrix} = \frac{-1}{2\,\text{area}(E)} \begin{pmatrix} +(y_2 - y_1)\cdot(u_3 - u_1) - (u_2 - u_1)\cdot(y_3 - y_1) \\ -(x_2 - x_1)\cdot(u_3 - u_1) + (u_2 - u_1)\cdot(x_3 - x_1) \end{pmatrix}.$$

This formula can be written in the form

$$\nabla u = \frac{-1}{2\,\text{area}(E)} \begin{bmatrix} (y_3 - y_2) & (y_1 - y_3) & (y_2 - y_1) \\ (x_2 - x_3) & (x_3 - x_1) & (x_1 - x_2) \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{-1}{2\,\text{area}(E)}\,\mathbf{G}\cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \tag{25}$$

[12]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 allow to assure $n_3 > 0$.

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

$$\iint\limits_E a\,\nabla\phi\cdot\nabla u\;dA \approx \frac{a_1+a_2+a_3}{12\,\text{area}(E)}\,\langle \begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\mathbf{G}^T\cdot\mathbf{G}\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle\,.$$

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 $u\,\vec{b}\cdot\nabla\phi$

Since the gradient of ϕ is constant on each of the triangles use

$$\begin{pmatrix}\phi_x\\ \phi_y\end{pmatrix} = \nabla\phi = \frac{-1}{2\,\text{area}(E)}\,\mathbf{G}\cdot\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix} = \frac{-1}{2\,\text{area}(E)}\,\begin{bmatrix}\mathbf{G}_x\\ \mathbf{G}_y\end{bmatrix}\cdot\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,$$

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

$$\begin{aligned}
\iint\limits_E u\,b_1\,\phi_x\;dA &\approx \frac{\text{area}(E)}{3}\sum_{i=1}^{3} u(\vec{g}_i)\,b_{1,i}\,\phi_{x,i}\\
&= \frac{-\text{area}(E)}{3\cdot 2\,\text{area}(E)}\,\langle \begin{bmatrix}\mathbf{G}_x\\ \mathbf{G}_x\\ \mathbf{G}_x\end{bmatrix}\begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\begin{bmatrix} b_{1,1} & 0 & 0\\ 0 & b_{1,2} & 0\\ 0 & 0 & b_{1,3}\end{bmatrix}\mathbf{M}\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle\\
&= \frac{-1}{6}\,\langle \begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\begin{bmatrix}\mathbf{G}_x^T & \mathbf{G}_x^T & \mathbf{G}_x^T\end{bmatrix}\begin{bmatrix} b_{1,1} & 0 & 0\\ 0 & b_{1,2} & 0\\ 0 & 0 & b_{1,3}\end{bmatrix}\mathbf{M}\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle\\
&= \frac{-1}{6}\,\langle \begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\mathbf{G}_x^T\begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3}\end{bmatrix}\mathbf{M}\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle\\
&= \frac{-1}{6}\,\langle \begin{pmatrix}\phi_1\\ \phi_2\\ \phi_3\end{pmatrix}\,,\,\begin{bmatrix} b_{1,1}(y_3-y_2) & b_{1,2}(y_3-y_2) & b_{1,3}(y_3-y_2)\\ b_{1,1}(y_1-y_3) & b_{1,2}(y_1-y_3) & b_{1,3}(y_1-y_3)\\ b_{1,1}(y_2-y_1) & b_{1,2}(y_2-y_1) & b_{1,3}(y_2-y_1)\end{bmatrix}\mathbf{M}\begin{pmatrix}u_1\\ u_2\\ u_3\end{pmatrix}\rangle\,.
\end{aligned}$$

If the values of the second component of $\vec{b}$ at the Gauss nodes are given by $b_{2,i}$ find by similar computations

$$
\begin{aligned}
\iint\limits_E u\, b_2\, \phi_y\, dA &\approx \frac{-\text{area}(E)}{3} \sum_{i=1}^{3} u(\vec{g}_i)\, b_{2,i}\, \phi_{y,i} \\
&= \frac{-1}{6} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \mathbf{G}_y^T \begin{bmatrix} b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle \\
&= \frac{-1}{6} \langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{2,1}(x_2 - x_3) & b_{2,2}(x_2 - x_3) & b_{2,3}(x_2 - x_3) \\ b_{2,1}(x_3 - x_1) & b_{2,2}(x_3 - x_1) & b_{2,3}(x_3 - x_1) \\ b_{2,1}(x_1 - x_2) & b_{2,2}(x_1 - x_2) & b_{2,3}(x_1 - x_2) \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle .
\end{aligned}
$$

This leads to two more contributions to (24).

6.4.6 Integration over boundary segments

In expression (19) 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. Based on the two endpoints $\vec{x}_1$ and $\vec{x}_2$ use the values at the two Gauss integration points[13]

$$
\begin{aligned}
\vec{p}_1 &= \tfrac{1}{2}\,(\vec{x}_1 + \vec{x}_2) - \tfrac{1}{2\sqrt{3}}\,(\vec{x}_2 - \vec{x}_1) \\
\vec{p}_2 &= \tfrac{1}{2}\,(\vec{x}_1 + \vec{x}_2) + \tfrac{1}{2\sqrt{3}}\,(\vec{x}_2 - \vec{x}_1)\ .
\end{aligned}
$$

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

$$\int \phi\, g_2\, ds \approx \frac{L}{2} \langle \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}, \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \end{pmatrix} \rangle$$

[13]To derive the formula integrate 1, t, t^2 and t^3 over the interval $[-1, 1]$.

$$
\begin{aligned}
\textstyle\int_{-1}^{+1} f(t)\, dt &= w_1\, f(-\xi) + w_1\, f(+\xi) \\
\textstyle\int_{-1}^{+1} 1\, dt\ =\ 2 &= w_1\, 1 + w_1\, 1 \quad \Longrightarrow \quad w_1 = 1 \\
\textstyle\int_{-1}^{+1} t\, dt\ =\ 0 &= -\,w_1\, \xi + w_1\, \xi\ =\ 0 \\
\textstyle\int_{-1}^{+1} t^2\, dt\ =\ \frac{2}{3} &= +\,w_1\, \xi^2 + w_1\, \xi^2 \quad \Longrightarrow \quad \xi = \sqrt{1/3} \\
\textstyle\int_{-1}^{+1} t^3\, dt\ =\ 0 &= -\,w_1\, \xi^3 + w_1\, \xi^3\ =\ 0
\end{aligned}
$$

Thus t^4 is not integrated exactly and the error is proportional to h^4.

$$
\begin{aligned}
&= \frac{L}{2}\langle\begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix},\begin{bmatrix}(1-\alpha) & \alpha\\ \alpha & (1-\alpha)\end{bmatrix}\begin{pmatrix}g_2(\vec p_1)\\ g_2(\vec p_2)\end{pmatrix}\rangle\\
\int \phi\, g_3\, u\; ds &\approx \frac{L}{2}\langle\begin{bmatrix}(1-\alpha) & \alpha\\ \alpha & (1-\alpha)\end{bmatrix}\begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix},\begin{bmatrix}g_3(\vec p_1) & 0\\ 0 & g_3(\vec p_2)\end{bmatrix}\begin{bmatrix}(1-\alpha) & \alpha\\ \alpha & (1-\alpha)\end{bmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}\rangle\\
&= \frac{L}{2}\langle\begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix},\begin{bmatrix}(1-\alpha) & \alpha\\ \alpha & (1-\alpha)\end{bmatrix}\begin{bmatrix}(1-\alpha)\, g_3(\vec p_1) & \alpha\, g_3(\vec p_1)\\ \alpha\, g_3(\vec p_2) & (1-\alpha)\, g_3(\vec p_2)\end{bmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}\rangle\\
&= \frac{L}{2}\langle\begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix},\begin{bmatrix}(1-\alpha)^2\, g_3(\vec p_1)+\alpha^2\, g_3(\vec p_2) & (1-\alpha)\,\alpha\,(g_3(\vec p_1)+g_3(\vec p_2))\\ (1-\alpha)\,\alpha\,(g_3(\vec p_1)+g_3(\vec p_2)) & \alpha^2\, g_3(\vec p_1)+(1-\alpha)^2\, g_3(\vec p_2)\end{bmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}\rangle\,.
\end{aligned}
$$

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

$$\int \phi\, g_3\, u\, ds \approx \langle\vec\phi, \mathbf{B}\,\vec u\rangle = \langle\begin{pmatrix}\phi_2\\ \phi_2\end{pmatrix},\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}\rangle \tag{26}$$

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 $\vec f$.
- 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 (19)

$$0 = \iint\limits_{\Omega} \nabla\phi\cdot(a\,\nabla u - u\,\vec b) + \phi\ (b_0\, u - f)\ dA - \int_{\Gamma_2}\phi\ (g_2 + g_3\, u)\ ds$$

have to be transformed into

$$0 = \langle\vec\phi\,,\ \mathbf{A}\vec u + \mathbf{W}\vec f\rangle\,.$$

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 49 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 49. The parameters ξ and ν at the nodes are given by Table 12. 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 12: Coordinates of the nodes in the standard quadratic triangle

$$\vec{\Phi}(\xi,\nu) = \begin{pmatrix} \Phi_1(\xi,\nu) \\ \Phi_2(\xi,\nu) \\ \Phi_3(\xi,\nu) \\ \Phi_4(\xi,\nu) \\ \Phi_5(\xi,\nu) \\ \Phi_6(\xi,\nu) \end{pmatrix} = \begin{pmatrix} (1-\xi-\nu)\,(1-2\,\xi-2\,\nu) \\ \xi\,(2\,\xi-1) \\ \nu\,(2\,\nu-1) \\ 4\,\xi\,\nu \\ 4\,\nu\,(1-\xi-\nu) \\ 4\,\xi\,(1-\xi-\nu) \end{pmatrix} \tag{27}$$

and find their graphs in Figure 53.

Any quadratic polynomial f on the standard triangle Ω can be written as linear combination of the basis functions by using

$$f(\xi,\nu) = \sum_{i=1}^{6} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu)\,. = \sum_{i=1}^{6} f_i\,\Phi_i(\xi,\nu)\,. \tag{28}$$

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 49 determine the correct values of the parameters ξ and ν, i.e. solve

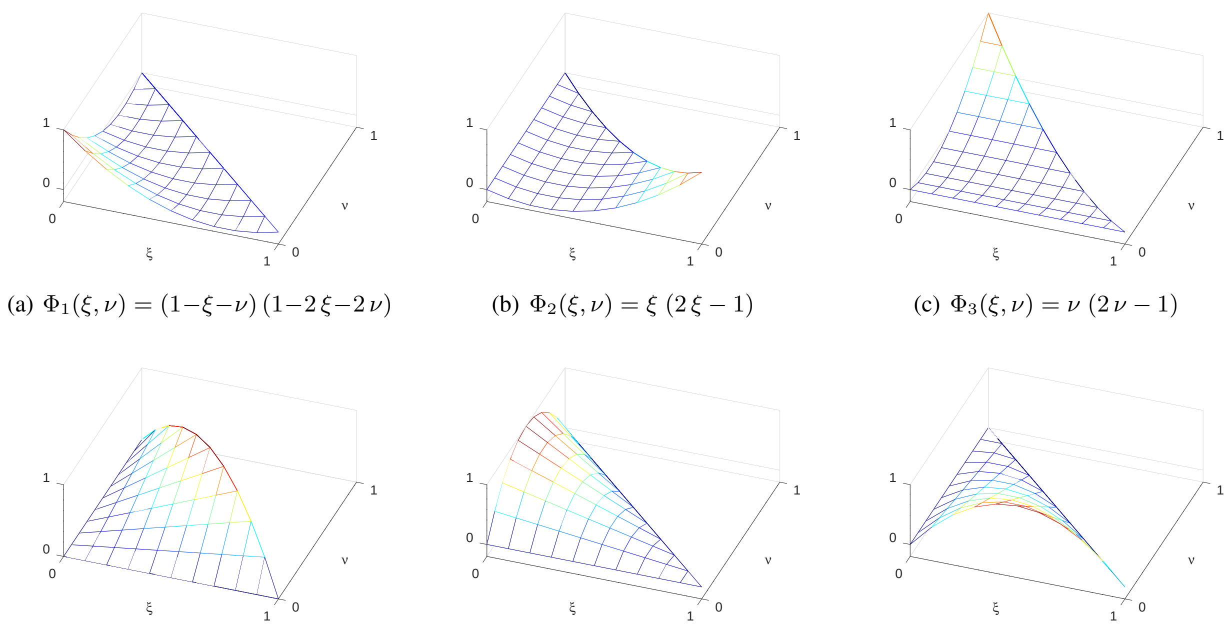
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \nu \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix}.$$

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

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$



(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 53: Basis functions for second order triangular elements

6.5.2 Determine values at the Gauss points and apply Gauss integration

Use equation (23) 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 $\mathbf{M} \in \mathbb{R}^{7\times 6}$

$$\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,6} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,6} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_6 \end{pmatrix} = \mathbf{M}\cdot\vec{f} \tag{29}$$

$$\approx \begin{bmatrix} +0.474353 & -0.080769 & -0.080769 & 0.041036 & 0.323074 & 0.323074 \\ -0.080769 & +0.474353 & -0.080769 & 0.323074 & 0.041036 & 0.323074 \\ -0.080769 & -0.080769 & +0.474353 & 0.323074 & 0.323074 & 0.041036 \\ -0.052584 & -0.028075 & -0.028075 & 0.884134 & 0.112300 & 0.112300 \\ -0.028075 & -0.052584 & -0.028075 & 0.112300 & 0.884134 & 0.112300 \\ -0.028075 & -0.028075 & -0.052584 & 0.112300 & 0.112300 & 0.884134 \\ -0.111111 & -0.111111 & -0.111111 & 0.444444 & 0.444444 & 0.444444 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix}$$

The Gauss integration can be written in the form

$$\iint\limits_{\Omega} f(\xi,\nu)\;dA \approx \sum_{j=1}^{7} w_j\; f(\vec{g}_j) = \langle \vec{w}\,,\,\mathbf{M}\cdot\vec{f}\rangle\,.$$

To integrate over the general triangle E use the transformation (20), i.e.

$$\iint\limits_{E} f\;dA = \iint\limits_{\Omega} f\left(\vec{x}\left(\xi,\nu\right)\right)\;\left|\det\left(\frac{\partial\left(x,y\right)}{\partial\left(\xi,\nu\right)}\right)\right|\;d\xi\;d\nu \approx \left|\det \mathbf{T}\right|\;\langle \vec{w}\,,\,\mathbf{M}\cdot\vec{f}\rangle\,.$$

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

$$\begin{aligned}\iint\limits_{E} f\,\phi\;dA \quad &\approx \quad |\det(\mathbf{T})|\sum_{j=1}^{7} w_j\;f_j\;\phi(g_j) \;=\; |\det(\mathbf{T})|\,\langle \operatorname{diag}(\vec{w})\vec{f}\,,\,\mathbf{M}\,\vec{\phi}\rangle \\ &= \quad |\det(\mathbf{T})|\,\langle \mathbf{M}^T\operatorname{diag}(\vec{w})\vec{f}\,,\,\vec{\phi}\rangle\,,\end{aligned}$$

  Thus find one contribution to (24).

- 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

$$\iint\limits_{E} f\,\phi\;dA \approx |\det(\mathbf{T})|\,\langle \operatorname{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\,\mathbf{M}\,\vec{\phi}\rangle = |\det(\mathbf{T})|\,\langle \mathbf{M}^T\operatorname{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\,\vec{\phi}\rangle\,.$$

The matrices $\mathbf{M}^T\operatorname{diag}(\vec{w})$ and $\mathbf{M}^T\operatorname{diag}(\vec{w})\,\mathbf{M}$ 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

$$\begin{aligned}\iint\limits_{E} b_0\,u\,\phi\;dA \quad &\approx \quad |\det(\mathbf{T})|\sum_{j=1}^{7} w_j\,b_0(g_j)\,u(g_j)\,\phi(g_j) \;=\; |\det(\mathbf{T})|\,\langle \operatorname{diag}(\vec{w})\;\operatorname{diag}(\vec{b}_0)\mathbf{M}\,\vec{u}\,,\,\mathbf{M}\,\vec{\phi}\rangle \\ &= \quad |\det(\mathbf{T})|\,\langle \mathbf{M}^T\operatorname{diag}(\vec{w})\;\operatorname{diag}(\vec{b}_0)\,\mathbf{M}\,\vec{u}\,,\,\vec{\phi}\rangle\,,\end{aligned}$$

where $\operatorname{diag}(\vec{b}_0) = \operatorname{diag}(b_0(\vec{g}_1), b_0(\vec{g}_2), b_0(\vec{g}_3),\ldots,\;b_0(\vec{g}_7))$.

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.

According to 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

$$\left(\frac{\partial\,g}{\partial\xi}\,,\,\frac{\partial\,g}{\partial\nu}\right) = \left(\frac{\partial\,f}{\partial x}\,,\,\frac{\partial\,f}{\partial y}\right)\cdot\mathbf{T}\,. \tag{30}$$

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

$$\begin{pmatrix}\frac{\partial\,f}{\partial x}\\ \frac{\partial\,f}{\partial y}\end{pmatrix} = \frac{1}{\det\mathbf{T}}\begin{bmatrix} y_3-y_1 & -y_2+y_1\\ -x_3+x_1 & x_2-x_1\end{bmatrix}\begin{pmatrix}\frac{\partial\,g}{\partial\xi}\\ \frac{\partial\,g}{\partial\nu}\end{pmatrix}. \tag{31}$$

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 (27). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (27). The result is

$$\operatorname{grad}\vec{\Phi} = \begin{bmatrix} -3+4\,\xi+4\,\nu & -3+4\,\xi+4\,\nu \\ 4\,\xi-1 & 0 \\ 0 & 4\,\nu-1 \\ 4\,\nu & 4\,\xi \\ -4\,\nu & 4-4\,\xi-8\,\nu \\ 4-8\,\xi-4\,\nu & -4\,\xi \end{bmatrix} = \left[\ \vec{\Phi}_\xi(\xi,\nu) \quad \vec{\Phi}_\nu(\xi,\nu)\ \right]. \tag{32}$$

Thus find on the standard triangle Ω

$$\left(\frac{\partial\, g}{\partial\xi}\,,\,\frac{\partial\, g}{\partial\nu}\right) = (g_1,g_2,g_3,g_4,g_5,g_6)\cdot\left[\ \vec{\Phi}_\xi(\xi,\nu) \quad \vec{\Phi}_\nu(\xi,\nu)\ \right] = \vec{g}^T\cdot\left[\ \vec{\Phi}_\xi(\xi,\nu) \quad \vec{\Phi}_\nu(\xi,\nu)\ \right].$$

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

$$\left(\frac{\partial\,\varphi}{\partial x}\,,\,\frac{\partial\,\varphi}{\partial y}\right) = \left(\frac{\partial\,\varphi}{\partial \xi}\,,\,\frac{\partial\,\varphi}{\partial \nu}\right)\cdot\mathbf{T}^{-1} = \vec{\varphi}^T\cdot\left[\ \vec{\Phi}_\xi \quad \vec{\Phi}_\nu\ \right]\cdot\mathbf{T}^{-1}$$

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.

$$\frac{\partial\,\varphi}{\partial x} = \frac{1}{\det(\mathbf{T})}\left[\ (+y_3-y_1)\,\vec{\Phi}_\xi^T + (-y_2+y_1)\,\vec{\Phi}_\nu^T\ \right]\cdot\vec{\varphi}\,, \tag{33}$$

$$\frac{\partial\,\varphi}{\partial y} = \frac{1}{\det(\mathbf{T})}\left[\ (-x_3+x_1)\,\vec{\Phi}_\xi^T + (+x_2-x_1)\,\vec{\Phi}_\nu^T\ \right]\cdot\vec{\varphi} \tag{34}$$

For the numerical integration use the values of the gradients 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

$$\begin{aligned}
\mathbf{M}_\xi &= \begin{bmatrix} -3+4\,\xi_1+4\,\nu_1 & 4\,\xi_1-1 & 0 & 4\,\nu_1 & -4\,\nu_1 & 4-8\,\xi_1-4\,\nu_1 \\ -3+4\,\xi_2+4\,\nu_2 & 4\,\xi_2-1 & 0 & 4\,\nu_2 & -4\,\nu_2 & 4-8\,\xi_2-4\,\nu_2 \\ \vdots & & & & & \vdots \\ -3+4\,\xi_7+4\,\nu_7 & 4\,\xi_7-1 & 0 & 4\,\nu_7 & -4\,\nu_7 & 4-8\,\xi_7-4\,\nu_7 \end{bmatrix} \\
&\approx \begin{bmatrix} -2.18971 & -0.59485 & 0.00000 & 0.40515 & -0.40515 & 2.78456 \\ 0.59485 & 2.18971 & 0.00000 & 0.40515 & -0.40515 & -2.78456 \\ 0.59485 & -0.59485 & 0.00000 & 3.18971 & -3.18971 & 0.00000 \\ 0.76114 & 0.88057 & 0.00000 & 1.88057 & -1.88057 & -1.64170 \\ -0.88057 & -0.76114 & 0.00000 & 1.88057 & -1.88057 & 1.64170 \\ -0.88057 & 0.88057 & 0.00000 & 0.23886 & -0.23886 & 0.00000 \\ -0.33333 & 0.33333 & 0.00000 & 1.33333 & -1.33333 & 0.00000 \end{bmatrix}
\end{aligned}$$

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 the function at the nodes. On the standard triangle Ω use the formulas for the partial derivatives of the basis functions in expression (32) to find them at the nodes, given by the (ξ, ν) coordinates in Table 12 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

$$\begin{pmatrix} \varphi_\nu(\xi_1, \nu_1) \\ \varphi_\nu(\xi_2, \nu_2) \\ \varphi_\nu(\xi_3, \nu_3) \\ \varphi_\nu(\xi_4, \nu_4) \\ \varphi_\nu(\xi_5, \nu_5) \\ \varphi_\nu(\xi_6, \nu_6) \end{pmatrix} = \begin{bmatrix} -3 & 1 & 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & 1 & 1 & -1 \\ 0 & 4 & 0 & 2 & 0 & 2 \\ 4 & 0 & -4 & -2 & 0 & 2 \\ 0 & -4 & 0 & -2 & 0 & -2 \end{bmatrix} \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix} = \mathbf{N}_\nu \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \\ \varphi_6 \end{pmatrix}$$

Now use the transformation formulas (33) and (34) 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

$$\begin{aligned} \iint\limits_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA &\approx \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\big((y_3-y_1)\,\mathbf{M}_\xi^T + (-y_2+y_1)\,\mathbf{M}_\nu^T\big)\cdot\operatorname{diag}(\overrightarrow{wb_1})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle \\ \iint\limits_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA &\approx \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\big((-x_3+x_1)\,\mathbf{M}_\xi^T + (x_2-x_1)\,\mathbf{M}_\nu^T\big)\cdot\operatorname{diag}(\overrightarrow{wb_2})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle\,. \end{aligned}$$

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

$$\begin{aligned} \mathbf{A}_x &= \left[\;(+y_3-y_1)\,\mathbf{M}_\xi + (-y_2+y_1)\,\mathbf{M}_\nu\;\right]^T\cdot\operatorname{diag}(\overrightarrow{wa})\cdot\left[\;(+y_3-y_1)\,\mathbf{M}_\xi + (-y_2+y_1)\,\mathbf{M}_\nu\;\right] \\ \mathbf{A}_y &= \left[\;(-x_3+x_1)\,\mathbf{M}_\xi + (+x_2-x_1)\,\mathbf{M}_\nu\;\right]^T\cdot\operatorname{diag}(\overrightarrow{wa})\cdot\left[\;(-x_3+x_1)\,\mathbf{M}_\xi + (+x_2-x_1)\,\mathbf{M}_\nu\;\right]\,. \end{aligned}$$

6.5.8 Integration over boundary segments

In expression (19) we have to 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. 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[14]

$$\int_{-h/2}^{h/2} f(x)\ dx \approx \frac{h}{18}\ \left(5\,f(-\frac{\sqrt{3}}{2\sqrt{5}}\,h) + 8\,f(0) + 5\,f(\frac{\sqrt{3}}{2\sqrt{5}}\,h)\right)$$

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

$$\begin{aligned}
\vec{p}_1 &= \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_3) - \tfrac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_3 - \vec{x}_1)\\
\vec{p}_2 &= \vec{x}_2 = \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_3)\\
\vec{p}_3 &= \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_3) + \tfrac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_3 - \vec{x}_1)
\end{aligned}$$

use a quadratic interpolation of a function with $f_- = f(-h/2)$, $f_0 = f(0)$ and $f_+ = f(+h/2)$. Since[15]

$$f(x) = f_0 + \frac{f_+ - f_-}{h}\,x + 2\,\frac{f_- - 2\,f_0 + f_+}{h^2}\,x^2$$

the quadratic interpolation result at $\pm\alpha\,h$ is

$$\begin{aligned}
f(\pm\alpha\,h) &= f_0 \pm (f_+ - f_-)\,\alpha + 2\,(f_- - 2\,f_0 + f_+)\,\alpha^2\\
&= f_-\,(\pm\alpha + 2\,\alpha^2) + f_0\,(1 - 4\,\alpha^2) + f_+\,(\mp\alpha + 2\,\alpha^2)
\end{aligned}$$

[14]To derive the 3 point Gauss integration scheme use

$$\begin{aligned}
\int_{-1}^{+1} f(t)\,dt &= w_1\,f(-\xi) + w_0\,f(0) + w_1\,f(+\xi)\\
\int_{-1}^{+1} 1\,dt = 2 &= w_1\,1 + w_0\,1 + w_1\,1\\
\int_{-1}^{+1} t\,dt = 0 &= -\,w_1\,\xi + w_0\,0 + w_1\,\xi = 0\\
\int_{-1}^{+1} t^2\,dt = \tfrac{2}{3} &= +\,w_1\,\xi^2 + w_1\,\xi^2\\
\int_{-1}^{+1} t^3\,dt = 0 &= -\,w_1\,\xi^3 + w_1\,\xi^3 = 0\\
\int_{-1}^{+1} t^4\,dt = \tfrac{2}{5} &= +\,w_1\,\xi^4 + w_1\,\xi^4\\
\int_{-1}^{+1} t^5\,dt = 0 &= -\,w_1\,\xi^5 + w_1\,\xi^5 = 0
\end{aligned}$$

Thus t^6 is not integrated exactly and the error is proportional to h^6. The system to be solved is

$$\left\{\begin{aligned}
w_0 + 2\,w_1 &= 2\\
2\,w_1\,\xi^2 &= \tfrac{2}{3}\\
2\,w_1\,\xi^4 &= \tfrac{2}{5}
\end{aligned}\right. \quad\Longrightarrow\quad \xi^2 = \frac{3}{5},\ w_1 = \frac{5}{9},\ w_0 = \frac{8}{9}\,.$$

[15]To verify use $f(0) = f_0$ and

$$f(\pm h/2) = f_0 \pm \frac{f_+ - f_-}{h}\,\frac{h}{2} + 2\,\frac{f_- - 2\,f_0 + f_+}{h^2}\,\frac{h^2}{4} = f_0 \pm \frac{1}{2}(f_+ - f_-) + \frac{1}{2}\,(f_- - 2\,f_0 + f_+)\,.$$

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

$$\begin{aligned}
\begin{pmatrix} u(\vec p_1) \\ u(\vec p_2) \\ u(\vec p_3) \end{pmatrix} &= \begin{bmatrix} +\alpha + 2\,\alpha^2 & 1-4\,\alpha^2 & -\alpha+2\,\alpha^2 \\ 0 & 1 & 0 \\ -\alpha+2\,\alpha^2 & 1-4\,\alpha^2 & +\alpha+2\,\alpha^2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \\
&= \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \approx \begin{bmatrix} +0.68730 & 0.4 & -0.08730 \\ 0 & 1 & 0 \\ -0.08730 & 0.4 & +0.68730 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}
\end{aligned} \tag{35}$$

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 (19)

$$0 = \iint\limits_{\Omega} \nabla\phi\cdot(a\,\nabla u - u\,\vec{b}) + \phi\ (b_0\,u - f)\ dA - \int_{\Gamma_2} \phi\ (g_2 + g_3\,u)\ ds$$

have to be transformed into

$$0 = \langle\vec{\phi}\,,\,\mathbf{A}\vec{u} + \mathbf{W}\vec{f}\rangle\,. \tag{36}$$

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^2y, $x\,y^2$ and y^3.

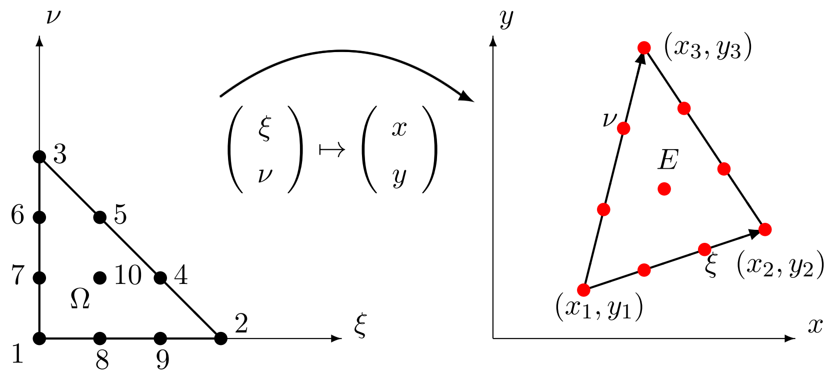


Figure 54: Transformation of the cubic standard triangle Ω to a general triangle E

6.6.1 The basis functions for a third order element and cubic interpolation

Examine the standard triangle Ω in Figure 54 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 54. The parameters ξ and ν at the nodes are given by Table 13. 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[16]

[16] 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.

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 13: Coordinates of the nodes in the standard cubic triangle

$$\vec{\Phi}(\xi,\nu) = \begin{pmatrix} \Phi_1(\xi,\nu) \\ \Phi_2(\xi,\nu) \\ \Phi_3(\xi,\nu) \\ \Phi_4(\xi,\nu) \\ \Phi_5(\xi,\nu) \\ \Phi_6(\xi,\nu) \\ \Phi_7(\xi,\nu) \\ \Phi_8(\xi,\nu) \\ \Phi_9(\xi,\nu) \\ \Phi_{10}(\xi,\nu) \end{pmatrix} = \begin{pmatrix} (1-(\xi+\nu))\,(1-3\,(\xi+\nu))\,(1-\frac{3}{2}\,(\xi+\nu)) \\ \xi\,(3\,\xi-1)\,(\frac{3}{2}\,\xi-1) \\ \nu\,(3\,\nu-1)\,(\frac{3}{2}\,\nu-1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\xi-1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\nu-1) \\ \frac{9}{2}\,\nu\,(1-(\xi+\nu))\,(3\,\nu-1) \\ 9\,\nu\,(1-(\xi+\nu))\,(1-\frac{3}{2}\,(\xi+\nu)) \\ 9\,\xi\,(1-\frac{3}{2}\,(\xi+\nu))\,(1-(\xi+\nu)) \\ \frac{9}{2}\,\xi\,(3\,\xi-1)\,(1-(\xi+\nu)) \\ 27\,\xi\,\nu\,(1-(\xi+\nu)) \end{pmatrix} \tag{37}$$

$$= \begin{pmatrix} 1-\frac{11}{2}\,\xi-\frac{11}{2}\,\nu+9\,\xi^2+18\,\xi\,\nu+9\,\nu^2-\frac{9}{2}\,\xi^3-\frac{27}{2}\,\xi^2\,\nu-\frac{27}{2}\,\xi\,\nu^2-\frac{9}{2}\,\nu^3 \\ \xi-\frac{9}{2}\,\xi^2+\frac{9}{2}\,\xi^3 \\ \nu-\frac{9}{2}\,\nu^2+\frac{9}{2}\,\nu^3 \\ -\frac{9}{2}\,\xi\,\nu+\frac{27}{2}\,\xi^2\,\nu \\ -\frac{9}{2}\,\xi\,\nu+\frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\nu+\frac{9}{2}\,\xi\,\nu+18\,\nu^2-\frac{27}{2}\,\xi\,\nu^2-\frac{27}{2}\,\nu^3 \\ 9\,\nu-\frac{45}{2}\,\xi\,\nu-\frac{45}{2}\,\nu^2+\frac{27}{2}\,\xi^2\,\nu+27\,\xi\,\nu^2+\frac{27}{2}\,\nu^3 \\ 9\,\xi-\frac{45}{2}\,\xi^2-\frac{45}{2}\,\xi\,\nu+\frac{27}{2}\,\xi^3+27\,\xi^2\,\nu+\frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\xi+18\,\xi^2+\frac{9}{2}\,\xi\,\nu-\frac{27}{2}\,\xi^3-\frac{27}{2}\,\xi^2\,\nu \\ 27\,\xi\,\nu-27\,\xi^2\,\nu-27\,\xi\,\nu^2 \end{pmatrix} \tag{38}$$

and find their graphs in Figure 55.

Any cubic polynomial f on the standard triangle Ω can be written as linear combination of the 10 basis functions by using

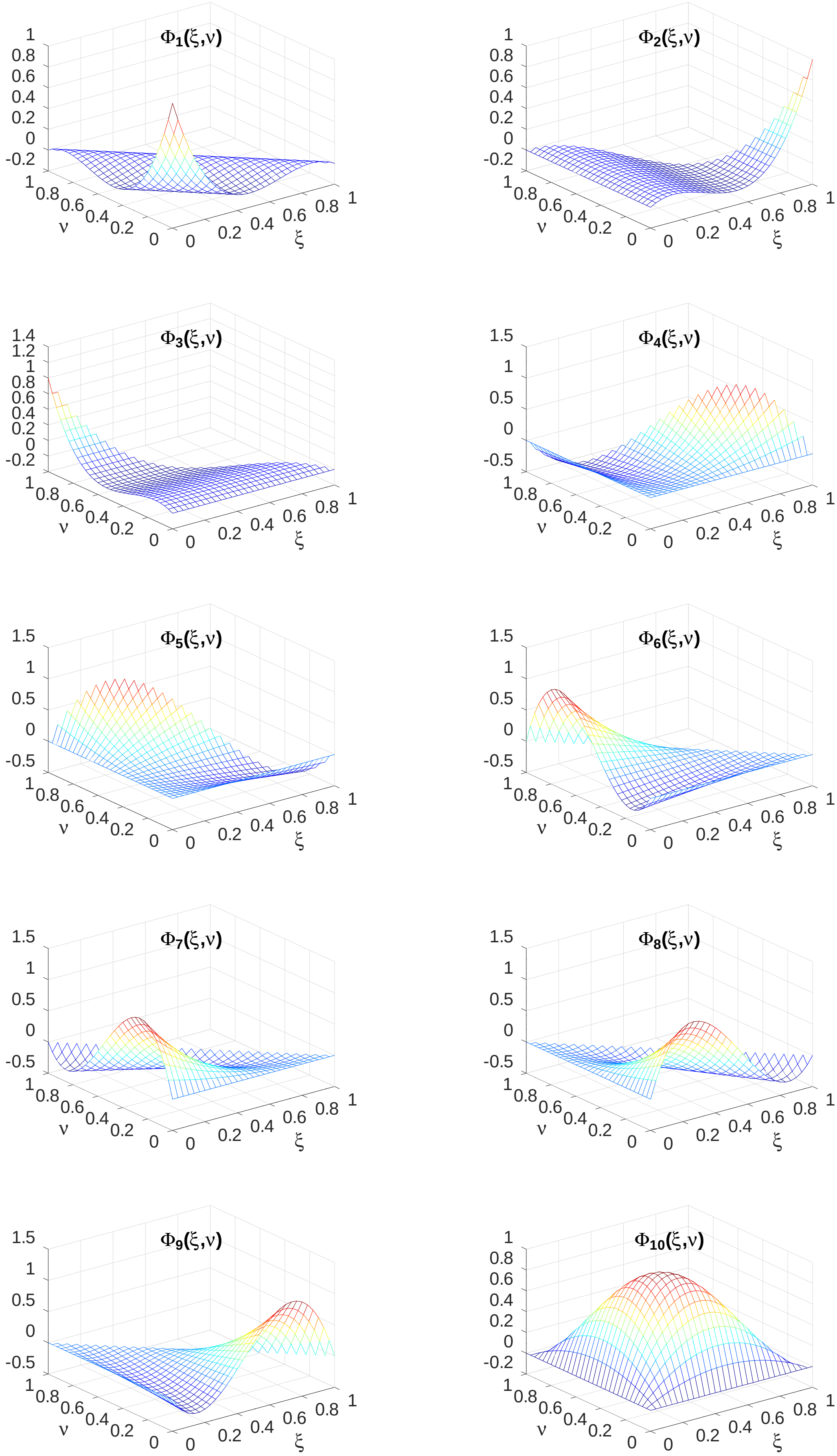
$$f(\xi,\nu) = \sum_{i=1}^{10} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu) = \sum_{i=1}^{10} f_i\,\Phi_i(\xi,\nu)\,. \tag{39}$$

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 54. 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

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1}\cdot\begin{pmatrix} x-x_1 \\ y-y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3-y_1 & -x_3+x_1 \\ -y_2+y_1 & x_2-x_1 \end{bmatrix}\cdot\begin{pmatrix} x-x_1 \\ y-y_1 \end{pmatrix}.$$



SHA 19-5-23

Figure 55: The 10 basis functions for third order triangular elements

6.6.2 Determine values at the Gauss points and apply Gauss integration

Use equation (23) (page 101) 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 $\mathbf{M} \in \mathbb{R}^{7\times 10}$

$$\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,10} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,10} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{10} \end{pmatrix} = \mathbf{M}\cdot\vec{f} \approx \tag{40}$$

$$\approx \begin{bmatrix} +0.22 & +0.06 & +0.06 & -0.03 & -0.03 & -0.25 & +0.51 & +0.51 & -0.25 & +0.22 \\ +0.06 & +0.22 & +0.06 & +0.51 & -0.25 & -0.03 & -0.03 & -0.25 & +0.51 & +0.22 \\ +0.06 & +0.06 & +0.22 & -0.25 & +0.51 & +0.51 & -0.25 & -0.03 & -0.03 & +0.22 \\ +0.04 & -0.06 & -0.06 & +0.41 & +0.41 & +0.05 & -0.10 & -0.10 & +0.05 & +0.36 \\ -0.06 & +0.04 & -0.06 & -0.10 & +0.05 & +0.41 & +0.41 & +0.05 & -0.10 & +0.36 \\ -0.06 & -0.06 & +0.04 & +0.05 & -0.10 & -0.10 & +0.05 & +0.41 & +0.41 & +0.36 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_9 \\ f_{10} \end{pmatrix}$$

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 (20), i.e.

$$\iint\limits_{E} f\;dA = \iint\limits_{\Omega} f\,(\vec{x}\,(\xi,\nu))\;\left|\det\left(\frac{\partial\,(x,y)}{\partial\,(\xi,\nu)}\right)\right|\;d\xi\,d\nu \approx |\det\mathbf{T}|\;\langle\vec{w}\,,\,\mathbf{M}\cdot\vec{f}\rangle\,.$$

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 (36).

- 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

$$\iint\limits_E f\,\phi\,dA \approx |\det(\mathbf{T})|\,\langle \operatorname{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\,\mathbf{M}\,\vec{\phi}\rangle = |\det(\mathbf{T})|\,\langle \mathbf{M}^T \operatorname{diag}(\vec{w})\,\mathbf{M}\vec{f}\,,\,\vec{\phi}\rangle\,.$$

- 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

$$\begin{aligned}\iint\limits_E b_0\,u\,\phi\,dA &\approx |\det(\mathbf{T})| \sum_{j=1}^{7} w_j\,b_0(g_j)\,u(g_j)\,\phi(g_j) = |\det(\mathbf{T})|\,\langle \operatorname{diag}(\vec{w})\ \operatorname{diag}(\vec{b}_0)\mathbf{M}\,\vec{u}\,,\,\mathbf{M}\,\vec{\phi}\rangle \\ &= |\det(\mathbf{T})|\,\langle \mathbf{M}^T \operatorname{diag}(\vec{w})\ \operatorname{diag}(\vec{b}_0)\,\mathbf{M}\,\vec{u}\,,\,\vec{\phi}\rangle\,.\end{aligned}$$

The matrices $\mathbf{M}^T \operatorname{diag}(\vec{w})$ and $\mathbf{M}^T \operatorname{diag}(\vec{w})\,\mathbf{M}$ 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 (38). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (38).

The results are

$$\vec{\Phi}_\xi(\xi,\nu) = \frac{\partial}{\partial\xi}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 1 - 9\,\xi + \frac{27}{2}\,\xi^2 \\ 0 \\ -\frac{9}{2}\,\nu + 27\,\xi\,\nu \\ -\frac{9}{2}\,\nu + \frac{27}{2}\,\nu^2 \\ \frac{9}{2}\,\nu - \frac{27}{2}\,\nu^2 \\ -\frac{45}{2}\,\nu + 27\,\xi\,\nu + 27\,\nu^2 \\ 9 - 45\,\xi - \frac{45}{2}\,\nu + \frac{81}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2} + 36\,\xi + \frac{9}{2}\,\nu - \frac{81}{2}\,\xi^2 - 27\,\xi\,\nu \\ 27\,\nu - 54\,\xi\,\nu - 27\,\nu^2 \end{pmatrix} \tag{41}$$

and

$$\vec{\Phi}_\nu(\xi,\nu) = \frac{\partial}{\partial\nu}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 0 \\ 1 - 9\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2}\,\xi + \frac{27}{2}\,\xi^2 \\ -\frac{9}{2}\,\xi + 27\,\xi\,\nu \\ -\frac{9}{2} + \frac{9}{2}\,\xi + 36\,\nu - 27\,\xi\,\nu - \frac{81}{2}\,\nu^2 \\ 9 - \frac{45}{2}\,\xi - 45\,\nu + \frac{27}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{81}{2}\,\nu^2 \\ -\frac{45}{2}\,\xi + 27\,\xi^2 + 27\,\xi\,\nu \\ +\frac{9}{2}\,\xi - \frac{27}{2}\,\xi^2 \\ 27\,\xi - 27\xi^2 - 54\,\xi\,\nu \end{pmatrix} . \tag{42}$$

Thus find on the standard triangle Ω

$$\left(\frac{\partial\,u}{\partial\xi}\,,\,\frac{\partial\,u}{\partial\nu}\right) = (u_1,u_2,\ldots,u_{10})\cdot\left[\begin{array}{cc} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{array}\right] = \vec{u}^T\cdot\left[\begin{array}{cc} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{array}\right] .$$

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 gradients at the Gauss integration points $\vec{g}_j = (\xi_j,\nu_j)$. Using expression (38) the values of the function φ at the Gauss points can be computed with the help of the interpolation matrix $\mathbf{M}$ byc

$$\begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{10} \end{pmatrix} .$$

Similarly, using (41) and (42), define the interpolation matrices for the partial derivatives.

$$\frac{\partial}{\partial \xi} \begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\xi \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{10} \end{pmatrix} \quad \text{and} \quad \frac{\partial}{\partial \nu} \begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\nu \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{10} \end{pmatrix} . \tag{43}$$

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

$$\iint_E u\,\vec{b}\,\nabla\phi\;dA = \iint_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA + \iint_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA$$

is approximated by

$$\begin{aligned}
\iint_E u\,b_1\,\frac{\partial\,\phi}{\partial x}\;dA \quad &\approx \quad \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\left((y_3 - y_1)\,\mathbf{M}_\xi^T + (-y_2 + y_1)\,\mathbf{M}_\nu^T\right)\cdot \operatorname{diag}(\overrightarrow{wb_1})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle \\
\iint_E u\,b_2\,\frac{\partial\,\phi}{\partial y}\;dA \quad &\approx \quad \frac{|\det\mathbf{T}|}{\det\mathbf{T}}\,\langle\left((-x_3 + x_1)\,\mathbf{M}_\xi^T + (x_2 - x_1)\,\mathbf{M}_\nu^T\right)\cdot \operatorname{diag}(\overrightarrow{wb_2})\cdot\mathbf{M}\cdot\vec{u}\,,\,\vec{\phi}\rangle\,.
\end{aligned}$$

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

$$\begin{aligned}
\iint_E a\,\frac{\partial\,u(\vec{x})}{\partial x}\,\frac{\partial\,\phi(\vec{x})}{\partial x}\;dA \quad &= \quad |\det\mathbf{T}|\,\iint_\Omega a(\vec{x}(\xi,\nu))\frac{\partial\,u(\vec{x}(\xi,\nu))}{\partial x}\,\frac{\partial\,\phi(\vec{x}(\xi,\nu))}{\partial x}\;d\xi\,d\nu \\
&\approx \quad \frac{|\det\mathbf{T}|}{(\det\mathbf{T})^2}\,\langle\mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det\mathbf{T}|}\,\langle\mathbf{A}_x\cdot\vec{u}\,,\,\vec{\phi}\rangle \\
\iint_E a\,\frac{\partial\,u(\vec{x})}{\partial y}\,\frac{\partial\,\phi(\vec{x})}{\partial y}\;dA \quad &= \quad |\det\mathbf{T}|\,\iint_\Omega a(\vec{x}(\xi,\nu))\frac{\partial\,u(\vec{x}(\xi,\nu))}{\partial y}\,\frac{\partial\,\phi(\vec{x}(\xi,\nu))}{\partial y}\;d\xi\,d\nu \\
&\approx \quad \frac{|\det\mathbf{T}|}{(\det\mathbf{T})^2}\,\langle\mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle = \frac{1}{|\det\mathbf{T}|}\,\langle\mathbf{A}_y\cdot\vec{u}\,,\,\vec{\phi}\rangle
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_x \quad &= \quad \left[\ (+y_3 - y_1)\,\mathbf{M}_\xi + (-y_2 + y_1)\,\mathbf{M}_\nu\ \right]^T\cdot\operatorname{diag}(\overrightarrow{wa})\cdot\left[\ (+y_3 - y_1)\,\mathbf{M}_\xi + (-y_2 + y_1)\,\mathbf{M}_\nu\ \right] \\
\mathbf{A}_y \quad &= \quad \left[\ (-x_3 + x_1)\,\mathbf{M}_\xi + (+x_2 - x_1)\,\mathbf{M}_\nu\ \right]^T\cdot\operatorname{diag}(\overrightarrow{wa})\cdot\left[\ (-x_3 + x_1)\,\mathbf{M}_\xi + (+x_2 - x_1)\,\mathbf{M}_\nu\ \right]\,.
\end{aligned}$$

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 (41) and (42) to find them at the nodes, given by the (ξ, ν) coordinates in Table 13 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 (33) and (34) 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 (19) 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

$$\begin{aligned}\vec{p}_1 &= \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_4) - \tfrac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_4 - \vec{x}_1) \\ \vec{p}_2 &= \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_4) \\ \vec{p}_3 &= \tfrac{1}{2}\ (\vec{x}_1 + \vec{x}_4) + \tfrac{\sqrt{3}}{2\sqrt{5}}\ (\vec{x}_4 - \vec{x}_1)\end{aligned}$$

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 56 with the values of the function $f(x)$ indicated by red spots and the interpolation position and values in green. The

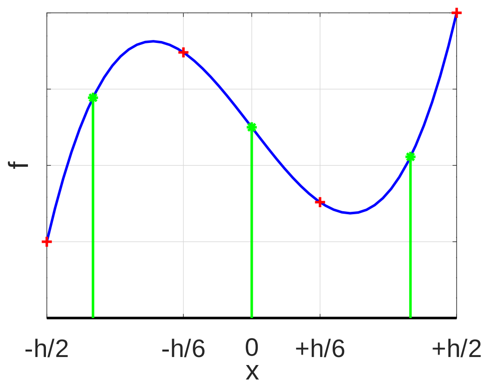


Figure 56: The interpolation from four nodes to three Gauss points on an interval $[-\frac{h}{2}, +\frac{h}{2}]$

computations are tedious[17] and lead to

$$\begin{pmatrix} u(\vec p_1) \\ u(\vec p_2) \\ u(\vec p_3) \end{pmatrix} = \mathbf{M}_B \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

[17] 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

$$\begin{aligned} p(\lambda\,h) &= \frac{1}{16} \begin{bmatrix} 1 & \lambda & \lambda^2 & \lambda^3 \end{bmatrix} \begin{bmatrix} -1 & +9 & +9 & -1 \\ +2 & -54 & +54 & -2 \\ +36 & -36 & -36 & +36 \\ -72 & +216 & -216 & +72 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \\ \begin{pmatrix} p(-\lambda\,h) \\ p(0) \\ p(+\lambda\,h) \end{pmatrix} &= \frac{1}{16} \begin{bmatrix} 1 & -\lambda & \lambda^2 & -\lambda^3 \\ 1 & 0 & 0 & 0 \\ 1 & +\lambda & \lambda^2 & +\lambda^3 \end{bmatrix} \begin{bmatrix} -1 & +9 & +9 & -1 \\ +2 & -54 & +54 & -2 \\ +36 & -36 & -36 & +36 \\ -72 & +216 & -216 & +72 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \\ &\approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \end{aligned}$$

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 $\mathbf{A}$.

6.7 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 solution $u_h(x, y)$ 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

$$\|u\|_2^2 = \iint\limits_{\Omega} u^2(x,y)\;dA \quad \text{and} \quad \|u\|_V^2 = \iint\limits_{\Omega} u^2(x,y) + \|\nabla u(x,y)\|^2\;dA\,.$$

The convergence results assume that the meshes are well defined, e.g. satisfy a minimal angle condition.

- 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 8.4.

6.8 Dynamic problems

The 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.

6.8.1 Dynamic problems of the heat equation type

Examine an IBVP (4) of parabolic type.

$$\begin{array}{rcllr}
\rho\,\frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u & = & f & \text{for} & (x,y,t)\in\Omega\times(0,T] \\
u & = & g_1 & \text{for} & (x,y,t)\in\Gamma_1\times(0,T] \\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) & = & g_2 + g_3\,u & \text{for} & (x,y,t)\in\Gamma_2\times(0,T] \\
u & = & u_0 & \text{on} & \Omega \quad \text{at} \quad t=0
\end{array}$$

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.

$$\begin{array}{rcllr}
-\nabla\cdot(a\,\nabla u_B - u_B\,\vec{b}) + b_0\,u_B & = & 0 & \text{for} & (x,y,t)\in\Omega \\
u_B & = & g_1 & \text{for} & (x,y)\in\Gamma_1 \\
\vec{n}\cdot(a\,\nabla u_B + u_B\,\vec{b}) & = & g_2 + g_3\,u_B & \text{for} & (x,y,t)\in\Gamma_2
\end{array} \tag{44}$$

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. $g_1 = g_2 = 0$.

$$\begin{array}{rcllr}
\rho\,\frac{\partial}{\partial t}\,v - \nabla\cdot(a\,\nabla u - v\,\vec{b}) + b_0\,v & = & f & \text{for} & (x,y,t)\in\Omega\times(0,T] \\
v & = & 0 & \text{for} & (x,y,t)\in\Gamma_1\times(0,T] \\
\vec{n}\cdot(a\,\nabla v + v\,\vec{b}) & = & g_3\,v & \text{for} & (x,y,t)\in\Gamma_2\times(0,T] \\
v & = & u_0 - u_B & \text{on} & \Omega \quad \text{at} \quad t=0
\end{array}$$

This equation is transformed to a system of ordinary differential equations.

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{v}_0\,. \tag{45}$$

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 and the position (x,y). Then use a Crank–Nicolson[18] approximation to advance the solution from time t to $t+\Delta t$.

$$\begin{aligned}
\mathbf{W}\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t)}{\Delta t} &= -\mathbf{A}\,\frac{\vec{v}(t+\Delta t) + \vec{v}(t)}{2} + \vec{f}(t+\Delta t/2) \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= +\left(\mathbf{W} - \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v}(t) + \Delta t\;\vec{f}(t+\Delta t/2)
\end{aligned}$$

For each time step such a system has to be solved. Observe that the matrix on the left does not change as time advances. Using an sparsity preserving LU factorization of the matrix on the left, these systems can be solved

[18] This is a standard choice and unconditionally stable, see e.g. [Stah08, §4].

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.

$$\begin{array}{rcll}
\mathbf{P}\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q} &=& \mathbf{L}\,\mathbf{U} & \text{LU factorization} \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v} &=& \vec{b} & \text{system to be solved} \\
\mathbf{P}\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q}\,\mathbf{Q}^{-1}\,\vec{v} &=& \mathbf{P}\,\vec{b} & \\
\mathbf{L}\,\mathbf{U}\,\mathbf{Q}^{-1}\,\vec{v} &=& \mathbf{P}\,\vec{b} & \\
\vec{v} &=& \mathbf{Q}\,(\mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{P}\,\vec{b}))) & \text{in the } \textit{Octave} \text{ code}
\end{array}$$

With the computed $\vec{v}(t)$ then find the solution $\vec{u}(t) = \vec{v}(t) + \vec{u}_B$ of the original problem.

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. but it not!

$$\begin{array}{rcll}
\mathbf{Q}^T\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q} &=& \mathbf{R}^T\,\mathbf{R} & \text{Cholesky factorization} \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v} &=& \vec{b} & \text{system to be solved} \\
\mathbf{Q}^T\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q}\,\mathbf{Q}^T\,\vec{v} &=& \mathbf{Q}^T\,\vec{b} & \\
\mathbf{R}^T\,\mathbf{R}\,\mathbf{Q}^T\,\vec{v} &=& \mathbf{Q}^T\,\vec{b} & \\
\vec{v} &=& \mathbf{Q}\,(\mathbf{R}\backslash(\mathbf{R}^T\backslash(\mathbf{Q}^T\,\vec{b}))) & \text{in the } \textit{Octave} \text{ code}
\end{array}$$

The *Octave* manual claims that a lower Cholesky factorization is often faster.

$$\begin{array}{rcll}
\mathbf{Q}^T\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q} &=& \mathbf{L}\,\mathbf{L}^T & \text{lower Cholesky factorization} \\
\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\vec{v} &=& \vec{b} & \text{system to be solved} \\
\mathbf{Q}^T\,\left(\mathbf{W} + \frac{\Delta t}{2}\,\mathbf{A}\right)\,\mathbf{Q}\,\mathbf{Q}^T\,\vec{v} &=& \mathbf{Q}^T\,\vec{b} & \\
\mathbf{L}\,\mathbf{L}^T\,\mathbf{Q}^T\,\vec{v} &=& \mathbf{Q}^T\,\vec{b} & \\
\vec{v} &=& \mathbf{Q}\,(\mathbf{L}^T\backslash(\mathbf{L}\backslash(\mathbf{Q}^T\,\vec{b}))) & \text{in the } \textit{Octave} \text{ code}
\end{array}$$

6.8.2 Using eigenvalues for dynamic problems of the heat equation type

With equation (45) for $\vec{f} = \vec{0}$

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{v}_0$$

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

$$\begin{array}{rcl}
\mathbf{W}\,\frac{d}{dt}\,\vec{u}(t) &=& -\lambda\,\mathbf{W}\,\vec{v}\,\exp(-\lambda\,t) \\
\mathbf{A}\,\vec{u}(t) &=& +\lambda\,\mathbf{W}\,\vec{v}\,\exp(-\lambda\,t)
\end{array}$$

Thus for $\lambda > 0$ find an exponentially decaying solution of the IBVP.

6.8.3 Dynamic problems of the wave equation type

Examine an IBVP (6) of hyperbolic type.

$$\begin{aligned}
\rho\,\frac{\partial^2}{\partial t^2}\,u + 2\,\alpha\,\frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u - u\,\vec{b}) + b_0\,u &= f && \text{for} \quad (x,y,t)\in\Omega\times(0,T]\\
u &= g_1 && \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T]\\
\vec{n}\cdot(a\,\nabla u - u\,\vec{b}) &= g_2 + g_3\,u && \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T]\\
u &= u_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0\\
\frac{\partial}{\partial t}\,u &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0
\end{aligned}$$

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$, using (44). 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. $g_1 = g_2 = 0$.

$$\begin{aligned}
\rho\,\frac{\partial^2}{\partial t^2}\,v + 2\,\alpha\,\frac{\partial}{\partial t}\,v(t) - \nabla\cdot(a\,\nabla u - v\,\vec{b}) + b_0\,v &= f && \text{for} \quad (x,y,t)\in\Omega\times(0,T]\\
v &= 0 && \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T]\\
\vec{n}\cdot(a\,\nabla v - v\,\vec{b}) &= g_3\,v && \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T]\\
v &= u_0 - u_B && \text{on} \quad \Omega \quad \text{at} \quad t=0\\
\frac{\partial}{\partial t}\,v &= v_0 && \text{on} \quad \Omega \quad \text{at} \quad t=0
\end{aligned}$$

This equation is transformed to a system of ordinary differential equations.

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{u}_0 - \vec{u}_B\,, \quad \frac{d}{dt}\,\vec{v}(0) = \vec{v}_0 \tag{46}$$

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[19] to advance the solution from time $t-\Delta t$ and t to $t+\Delta t$.

$$\begin{aligned}
\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) - \mathbf{A}\,\vec{v}(t) + \vec{f}(t)\\
\mathbf{W}\,\frac{\vec{v}(t-\Delta t) - 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{(\Delta t)^2} &= -2\,\mathbf{D}\,\frac{\vec{v}(t+\Delta t) - \vec{v}(t-\Delta t)}{2\,\Delta t} - \\
&\quad -\mathbf{A}\,\frac{\vec{v}(t-\Delta t) + 2\,\vec{v}(t) + \vec{v}(t+\Delta t)}{4} + \vec{f}(t)\\
\left(+\mathbf{W} + \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= -\left(\mathbf{W} - \Delta t\,\mathbf{D} + \frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t) + \\
&\quad + \left(2\,\mathbf{W} - \frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t) + (\Delta t)^2\,\vec{f}(t)
\end{aligned}$$

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. The above scheme is unconditionally stable, at least for constant coefficients[20].

[19] This is a standard choice and unconditionally stable, see e.g. [Stah08, §4].

[20] I have a proof in `WaveStability.tex`.

- To construct the solution at the initial time Δt 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

$$\begin{aligned}
\frac{d}{dt}\,\vec{v}(0) &= \vec{v}_0 \approx \frac{\vec{v}(\Delta t)-\vec{v}(-\Delta t)}{2\,\Delta t} \quad\Longrightarrow\quad \vec{v}(-\Delta t)\approx\vec{v}(\Delta t)-2\,\Delta t\,\vec{v}_0\\
\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) &= -2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t)-\mathbf{A}\,\vec{v}(t)+\vec{f}(t)\\
\mathbf{W}\,\frac{\vec{v}(t-\Delta t)-2\,\vec{v}(t)+\vec{v}(t+\Delta t)}{(\Delta t)^2} &= -2\,\mathbf{D}\,\frac{\vec{v}(t+\Delta t)-\vec{v}(t-\Delta t)}{2\,\Delta t}-\mathbf{A}\,\vec{v}(t)+\vec{f}(t)\\
(\mathbf{W}+\Delta t\,\mathbf{D})\,\vec{v}(t+\Delta t) &= -(\mathbf{W}-\Delta t\,\mathbf{D})\,\vec{v}(t-\Delta t)+2\,\mathbf{W}\,\vec{v}(t)+(\Delta t)^2\,(-\mathbf{A}\,\vec{v}(t)+\vec{f}(t))\\
(\mathbf{W}+\Delta t\,\mathbf{D})\,\vec{v}(\Delta t) &= -(\mathbf{W}-\Delta t\,\mathbf{D})\,(\vec{v}(\Delta t)-2\,\Delta t\,\vec{v}_0)+\\
&\qquad +2\,\mathbf{W}\,(\vec{u}_0-\vec{u}_B)+(\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0-u_B)+\vec{f}(0))\\
2\,\mathbf{W}\,\vec{v}(\Delta t) &= +2\,(\mathbf{W}-\Delta t\,\mathbf{D})\,\Delta t\,\vec{v}_0+\\
&\qquad +2\,\mathbf{W}\,(\vec{u}_0-\vec{u}_B)+(\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0-\vec{u}_B)+\vec{f}(0))\\
\mathbf{W}\,\vec{v}(\Delta t) &= (\mathbf{W}-\Delta t\,\mathbf{D})\,\Delta t\,\vec{v}_0+\\
&\qquad +\mathbf{W}\,(\vec{u}_0-\vec{u}_B)+\frac{1}{2}\,(\Delta t)^2\,(-\mathbf{A}\,(\vec{u}_0-\vec{u}_B)+\vec{f}(0))\,.
\end{aligned}$$

  This is currently implemented. The conditional stability for this single step should not cause a major problem.

- One could also use $\vec{v}(-\Delta t)\approx\vec{v}(\Delta t)-2\,\Delta 2\,\vec{v}_0$ in the implicit scheme at $t=0$.

$$\begin{aligned}
\left(+\mathbf{W}+\Delta t\,\mathbf{D}+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t+\Delta t) &= -\left(\mathbf{W}-\Delta t\,\mathbf{D}+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(t-\Delta t)+\\
&\qquad +\left(2\,\mathbf{W}-\frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(t)+(\Delta t)^2\,\vec{f}(t)\\
\left(+\mathbf{W}+\Delta t\,\mathbf{D}+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(\Delta t) &= -\left(\mathbf{W}-\Delta t\,\mathbf{D}+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,(\vec{v}(\Delta t)-2\,\Delta t\,\vec{v}_0)+\\
&\qquad +\left(2\,\mathbf{W}-\frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(0)+(\Delta t)^2\,\vec{f}(0)\\
\left(+2\,\mathbf{W}+2\,\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}(\Delta t) &= +2\,\Delta t\,\left(\mathbf{W}-\Delta t\,\mathbf{D}+\frac{(\Delta t)^2}{4}\,\mathbf{A}\right)\,\vec{v}_0+\\
&\qquad +\left(2\,\mathbf{W}-\frac{(\Delta t)^2}{2}\,\mathbf{A}\right)\,\vec{v}(0)+(\Delta t)^2\,\vec{f}(0)
\end{aligned}$$

  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 should be used.

6.8.4 Using eigenvalues for dynamic problems of the wave equation type

With equation (46) for $\vec{f}=\vec{0}$ and a damping factor $D\,\mathbf{W}$ with a constant $D\geq 0$ (instead of the matrix $\mathbf{D}$)

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t)+2\,D\,\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t)+\mathbf{A}\,\vec{v}(t)=\vec{0} \tag{47}$$

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 $0\leq D<\sqrt{\lambda}$ leads to a solution $\vec{u}(t)=\exp(\mu\,t)\,\vec{v}$ with $\mu\in\mathbb{C}$, since

$$\vec{0} = \mu^2\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t)+\mu\,2\,D\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t)+\lambda\,\mathbf{W}\,\vec{v}\,\exp(\mu\,t)$$

$$\begin{aligned} 0 &= \mu^2 + \mu\, 2\, D + \lambda \\ \mu_{1,2} &= \frac{1}{2}\left(-2\,D \pm \sqrt{4\,D^2 - 4\,\lambda}\right) = -D \pm i\,\sqrt{\lambda - D^2} \in \mathbb{C} \end{aligned}$$

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 (47) 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

  $$\frac{d}{dt}\begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{bmatrix}\begin{pmatrix} v(t) \\ \frac{d}{dt}\,\vec{v}(t) \end{pmatrix} = \begin{bmatrix} \mathbf{0} & \mathbb{I} \\ -\mathbf{A} & -\mathbf{W} \end{bmatrix}\begin{pmatrix} v(t) \\ \frac{d}{dt}\,\vec{v}(t) \end{pmatrix}.$$

  Thus the generalized eigenvalues of

  $$\begin{bmatrix} \mathbf{0} & \mathbb{I} \\ -\mathbf{A} & -\mathbf{W} \end{bmatrix}\vec{x} = \lambda\begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{bmatrix}\vec{x}$$

  provide information on the behavior of the solutions of the wave equation. This is not implemented in FEMoctave.

6.9 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 web.sha1.bfh.science/fem/VarFEM/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

$$\min_{\lambda_j \in \sigma(\mathbf{A})} |\beta_i - \lambda_j| \leq \sqrt{\langle \vec{r} \, , \, \mathbf{B}^{-1}\vec{r} \rangle} \quad \text{and} \quad |\beta_i - \lambda_j| \leq \frac{\langle \vec{r} \, , \, \mathbf{B}^{-1}\vec{r} \rangle}{\text{gap}} \tag{48}$$

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 gap exactly. Thus use the approximate values. Expect the error estimate to have its problems at multiple eigenvalues. For the largest, computed eigenvalue one can not estimate gap reliably, since no information on the next eigenvalue is available.

7 Plane Elasticity and Axially Symmetric Elasticity

Find the description of the plane elasticity problems in Section 2.8, starting on page 10.

7.1 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 (13) by

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

With FEMoctave examine plane stress deformations with (only) three types of boundary conditions.

$$\begin{array}{rcll} \vec{u} & = & \vec{g}_D & \text{on Dirichlet boundary } \Gamma_1\text{, i.e. prescibed displacement} \\ \text{force density} & = & \vec{g}_N & \text{on Neumann boundary } \Gamma_2\text{, i.e. prescibed force density} \\ \text{force density} & = & 0 & \text{on free boundary } \Gamma_3 \end{array} \tag{49}$$

With this the total energy of a plane stress problem can be written in the form[21]

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \qquad (50)\\ &= \iint\limits_\Omega \frac{1}{2}\,\frac{E}{(1-\nu^2)}\,\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2\,(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} , \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \, dA - \\ &\quad - \iint\limits_\Omega \vec{f}\cdot\vec{u}\,dA - \int_{\Gamma_2} \vec{g}_N \cdot \vec{u}\,ds\;. \end{aligned}$$

Using the Bernoulli principle this energy has to be minimized. A discretization of the displacments $u_1(x,y)$ 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 Figure 47 on page 96.

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

$$\begin{aligned} \varepsilon_{xx} &= \frac{\partial\,(u_1+\phi_1)}{\partial x} = \frac{\partial\,u_1}{\partial x} + \frac{\partial\,\phi_1}{\partial x} \\ \varepsilon_{xx}^2 &= (\frac{\partial\,(u_1+\phi_1)}{\partial x})^2 \approx (\frac{\partial\,u_1}{\partial x})^2 + 2\,(\frac{\partial\,u_1}{\partial x})\,(\frac{\partial\,\phi_1}{\partial x}) \\ \varepsilon_{yy}^2 &\approx (\frac{\partial\,u_2}{\partial y})^2 + 2\,(\frac{\partial\,u_2}{\partial y})\,(\frac{\partial\,\phi_2}{\partial y}) \\ \varepsilon_{xx}\varepsilon_{yy} &\approx \frac{\partial\,u_1}{\partial x}\,\frac{\partial\,u_2}{\partial y} + \frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_2}{\partial y} + \frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_1}{\partial x} \\ 2\,\varepsilon_{xy} &= \frac{\partial\,(u_1+\phi_1)}{\partial y} + \frac{\partial\,(u_2+\phi_2)}{\partial x} = \frac{\partial\,u_1}{\partial y} + \frac{\partial\,\phi_1}{dy} + \frac{\partial\,u_2}{\partial x} + \frac{\partial\,\phi_2}{\partial x} \end{aligned}$$

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

$$\begin{aligned}
4\,\varepsilon_{xy}^2 &\approx (\frac{\partial\,u_1}{\partial y})^2+(\frac{\partial\,u_2}{\partial x})^2+2\,\frac{\partial\,u_1}{\partial y}\,(\frac{\partial\,\phi_1}{\partial y}+\frac{\partial\,\phi_2}{\partial x})+2\,\frac{\partial\,u_2}{\partial x}\,(\frac{\partial\,\phi_1}{\partial y}+\frac{\partial\,\phi_2}{\partial x})\\
&= (\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})^2+2\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})+2\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})
\end{aligned}$$

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

$$\begin{aligned}
\frac{2\,(1-\nu^2)}{E}\,\left(W(\vec u+\vec\phi)-W(\vec u)\right) &\approx 2\,\frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_1}{\partial x}+2\,\frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_2}{\partial y}+\\
&\quad+2\,\nu\,(\frac{\partial\,u_1}{\partial x}\,\frac{\partial\,\phi_2}{\partial y}+\frac{\partial\,u_2}{\partial y}\,\frac{\partial\,\phi_1}{\partial x})+\\
&\quad+\frac{4}{4}\,(1-\nu)\,\left(\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})+\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)
\end{aligned}$$

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

$$\begin{aligned}
U(\vec u+\vec\phi)-U(\vec u) &\approx +\iint\limits_\Omega \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)-\phi_1\,f_1\,dA+\\
&\quad+\iint\limits_\Omega \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)-\phi_2\,f_2\,dA-\\
&\quad-\int_{\Gamma_2}\phi_1\,g_1+\phi_2\,g_2\,ds\,. \qquad (51)
\end{aligned}$$

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 (51) in the form

$$\langle\vec\phi,\mathbf{A}\,\vec u+\mathbf{W}\vec f\rangle=0 \quad \text{for all } \vec\phi\,.$$

7.2 Construction of first order elements

The algorithm in this section is based on the results in Section 6.4 (p. 101), with the expressions in equation (51) 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.

7.2.1 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

$$\begin{aligned}
\iint\limits_T f_1\,\phi_1+f_2\,\phi_2\,dA &\approx \frac{\text{area}(T)}{3}\,\left(\langle\mathbf{M}\,\vec\phi_1\,,\,\vec f_1\rangle+\langle\mathbf{M}\,\vec\phi_2\,,\,\vec f_2\rangle\right)\\
&= \frac{\text{area}(T)}{3}\,\left(\langle\vec\phi_1\,,\,\mathbf{M}^T\,\vec f_1\rangle+\langle\vec\phi_2\,,\,\mathbf{M}^T\,\vec f_2\rangle\right)\,.
\end{aligned}$$

$\mathbf{M} \in \mathbb{R}^{3\times 3}$ is the matrix for interpolation from the nodes to the Gauss points, given by

$$\mathbf{M} = \frac{1}{6}\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix} .$$

- 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 (51). With a block matrix notation the above can be written in the form

$$\frac{\text{area}(T)}{3}\,\langle\begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix},\begin{bmatrix}\mathbf{M}^T & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\end{bmatrix}\begin{pmatrix}\vec{f}_1\\ \vec{f}_2\end{pmatrix}\rangle \quad\text{or}\quad \frac{\text{area}(T)}{3}\,\langle\begin{pmatrix}\vec{\phi}_1\\ \vec{\phi}_2\end{pmatrix},\begin{bmatrix}\mathbf{M}^T\mathbf{M} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\mathbf{M}\end{bmatrix}\begin{pmatrix}\vec{f}_1\\ \vec{f}_2\end{pmatrix}\rangle .$$

7.2.2 Integration of the terms involving derivatives of ϕ_1 and ϕ_2

For linear elements the gradient of the functions u_i and ϕ_i are constant and using equation (25) 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

$$\begin{aligned} I_{\phi_1} &= \iint_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\ &\approx a_1\,\langle\mathbf{G}_x\vec{\phi}_1,\mathbf{G}_x\vec{u}_1\rangle + a_2\,\langle\mathbf{G}_x\vec{\phi}_1,\mathbf{G}_y\vec{u}_2\rangle + a_3\,\langle\mathbf{G}_y\vec{\phi}_1,\mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\ &= a_1\,\langle\vec{\phi}_1,\mathbf{G}_x^T\mathbf{G}_x\vec{u}_1\rangle + a_2\,\langle\vec{\phi}_1,\mathbf{G}_x^T\mathbf{G}_y\vec{u}_2\rangle + a_3\,\langle\vec{\phi}_1,\mathbf{G}_y^T\mathbf{G}_y\vec{u}_1+\mathbf{G}_y^T\mathbf{G}_x\vec{u}_2\rangle \\ I_{\phi_2} &= \iint_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x}) + \frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\ &\approx a_1\,\langle\mathbf{G}_y\vec{\phi}_2,\mathbf{G}_y\vec{u}_2\rangle + a_2\,\langle\mathbf{G}_y\vec{\phi}_2,\mathbf{G}_x\vec{u}_1\rangle + a_3\,\langle\mathbf{G}_x\vec{\phi}_2,\mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\ &= a_1\,\langle\vec{\phi}_2,\mathbf{G}_y^T\mathbf{G}_y\vec{u}_2\rangle + a_2\,\langle\vec{\phi}_2,\mathbf{G}_y^T\mathbf{G}_x\vec{u}_1\rangle + a_3\,\langle\vec{\phi}_2,\mathbf{G}_x^T\mathbf{G}_y\vec{u}_1+\mathbf{G}_x^T\mathbf{G}_x\vec{u}_2\rangle \end{aligned}$$

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, containing contributions to (51).

7.2.3 The boundary integral

The boundary integral is similar to (26) on page 107. 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.

7.3 Construction of second order elements

The algorithm in this section is based on the results in Section 6.5 (p. 107), with the expressions in equation (51) 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.

7.3.1 Integration of $f_1\,\phi_1 + f_2\,\phi_2$

Use the Gauss weights $\vec{w} \in \mathbb{R}^7$ from equation (22) on page 101 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

$$\begin{aligned}
\iint\limits_T f_1\,\phi_1 + f_2\,\phi_2\,dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\Big) \\
&= \text{area}(T)\,\Big(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\Big) \\
&= \text{area}(T)\,\langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\,\text{diag}(\vec{w}) & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\,\text{diag}(\vec{w}) \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix}\rangle\ .
\end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 6}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (29) on page 109.

- 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 (51). Observe that $\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}$ is a 6×6 matrix, independent on the triangle T.

7.3.2 Integration of the terms involving derivatives of 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

$$\begin{pmatrix} \varphi_x(\vec{x}_1) \\ \varphi_x(\vec{x}_2) \\ \vdots \\ \varphi_x(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (+y_3 - y_1)\, \mathbf{M}_\xi^T + (-y_2 + y_1)\, \mathbf{M}_\nu^T \, \right] \cdot \vec{\phi} =: \mathbf{G}_x \, \vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix} \varphi_y(\vec{x}_1) \\ \varphi_y(\vec{x}_2) \\ \vdots \\ \varphi_y(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[\, (-x_3 + x_1)\, \mathbf{M}_\xi^T + (+x_2 - x_1)\, \mathbf{M}_\nu^T \, \right] \cdot \vec{\phi} =: \mathbf{G}_y \, \vec{\phi} \, .$$

Evaluate the coefficients E and ν at the Gauss points g_i and multiply by the Gauss integration weights to obtain the three diagonal matrices

$$\mathbf{A}_1 = \text{diag} \begin{pmatrix} w_1 \, \frac{E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)} \\ w_2 \, \frac{E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)} \\ \vdots \\ w_7 \, \frac{E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)} \end{pmatrix} \quad , \quad \mathbf{A}_2 = \text{diag} \begin{pmatrix} w_1 \, \frac{\nu(\vec{g}_1)\, E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)} \\ w_2 \, \frac{\nu(\vec{g}_2)\, E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)} \\ \vdots \\ w_7 \, \frac{E(\nu(\vec{g}_7)\, \vec{g}_7)}{1-\nu^2(\vec{g}_7)} \end{pmatrix} \quad \text{and} \quad \mathbf{A}_3 = \text{diag} \begin{pmatrix} w_1 \, \frac{E(\vec{g}_1)}{2\,(1+\nu(\vec{g}_1))} \\ w_2 \, \frac{E(\vec{g}_2)}{2\,(1+\nu(\vec{g}_2))} \\ \vdots \\ w_7 \, \frac{E(\vec{g}_7)}{2\,(1+\nu(\vec{g}_7))} \end{pmatrix} .$$

This leads to the approximations

$$\begin{aligned}
\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)} \iint\limits_T \frac{E}{1-\nu^2} \left(\frac{\partial\,\phi_1}{\partial x} \left(\frac{\partial\,u_1}{\partial x} + \nu \, \frac{\partial\,u_2}{\partial y} \right) + \frac{1-\nu}{2} \, \frac{\partial\,\phi_1}{\partial y} \left(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x} \right) \right) dA \\
&\approx \langle \mathbf{A}_1 \, \mathbf{G}_x \vec{\phi}_1 , \mathbf{G}_x \vec{u}_1 \rangle + \langle \text{diag}\, \mathbf{A}_2 \, \mathbf{G}_x \vec{\phi}_1 , \mathbf{G}_y \vec{u}_2 \rangle + \langle \mathbf{A}_3 \, \mathbf{G}_y \vec{\phi}_1 , \mathbf{G}_y \vec{u}_1 + \mathbf{G}_x \vec{u}_2 \rangle \\
&= \langle \vec{\phi}_1 , \mathbf{G}_x^T \mathbf{A}_1 \mathbf{G}_x \vec{u}_1 \rangle + \langle \vec{\phi}_1 , \mathbf{G}_x^T \mathbf{A}_2 \mathbf{G}_y \vec{u}_2 \rangle + \langle \vec{\phi}_1 , \mathbf{G}_y^T \mathbf{A}_3 \mathbf{G}_y \vec{u}_1 + \mathbf{G}_y^T \mathbf{A}_3 \mathbf{G}_x \vec{u}_2 \rangle \\
\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)} \iint\limits_T \frac{E}{1-\nu^2} \left(\frac{\partial\,\phi_2}{\partial y} \left(\frac{\partial\,u_2}{\partial y} + \nu \, \frac{\partial\,u_1}{\partial x} \right) + \frac{1-\nu}{2} \, \frac{\partial\,\phi_2}{\partial x} \left(\frac{\partial\,u_1}{\partial y} + \frac{\partial\,u_2}{\partial x} \right) \right) dA \\
&\approx \langle \mathbf{A}_1 \mathbf{G}_y \vec{\phi}_2 , \mathbf{G}_y \vec{u}_2 \rangle + \langle \mathbf{A}_2 \mathbf{G}_y \vec{\phi}_2 , \mathbf{G}_x \vec{u}_1 \rangle + \langle \mathbf{A}_3 \mathbf{G}_x \vec{\phi}_2 , \mathbf{G}_y \vec{u}_1 + \mathbf{G}_x \vec{u}_2 \rangle \\
&= \langle \vec{\phi}_2 , \mathbf{G}_y^T \mathbf{A}_1 \mathbf{G}_y \vec{u}_2 \rangle + \langle \vec{\phi}_2 , \mathbf{G}_y^T \mathbf{A}_2 \mathbf{G}_x \vec{u}_1 \rangle + \langle \vec{\phi}_2 , \mathbf{G}_x^T \mathbf{A}_3 \mathbf{G}_y \vec{u}_1 + \mathbf{G}_x^T \mathbf{A}_3 \mathbf{G}_x \vec{u}_2 \rangle \, .
\end{aligned}$$

With a block matrix notation write the above in the form

$$\begin{aligned}
I_{\vec{\phi}} = I_{\phi_1} + I_{\phi_2} \quad &\approx \quad \text{area}(T) \, \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix} , \begin{bmatrix} \mathbf{G}_x^T \mathbf{A}_1 \mathbf{G}_x + \mathbf{G}_y^T \mathbf{A}_3 \mathbf{G}_y & \mathbf{G}_x^T \mathbf{A}_2 \mathbf{G}_y + \mathbf{G}_y^T \mathbf{A}_3 \mathbf{G}_x \\ \mathbf{G}_y^T \mathbf{A}_2 \mathbf{G}_x + \mathbf{G}_x^T \mathbf{A}_3 \mathbf{G}_y & \mathbf{G}_y^T \mathbf{A}_1 \mathbf{G}_y + \mathbf{G}_x^T \mathbf{A}_3 \mathbf{G}_x \end{bmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \end{pmatrix} \rangle \\
&=: \quad \text{area}(T) \, \langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix} , \mathbf{G} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \end{pmatrix} \rangle \, .
\end{aligned}$$

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 (51).

7.3.3 The boundary integral

The boundary integral is similar to (35) on page 117, 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 6.5.8.

7.4 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 7.3 to construct second order elements.

7.4.1 Integration of $f_1\, \phi_1 + f_2\, \phi_2$

Use the Gauss weights $\vec{w} \in \mathbb{R}^7$ from equation (22) on page 101 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

$$\begin{aligned}
\iint_T f_1\,\phi_1 + f_2\,\phi_2\; dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\Big) \\
&= \text{area}(T)\,\Big(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\vec{f}_2\rangle\Big) \\
&= \text{area}(T)\,\langle \begin{pmatrix} \vec{\phi}_1 \\ \vec{\phi}_2 \end{pmatrix}, \begin{bmatrix} \mathbf{M}^T\,\text{diag}(\vec{w}) & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^T\,\text{diag}(\vec{w}) \end{bmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \end{pmatrix} \rangle\,.
\end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 10}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (40) on page 121.

- 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

$$\begin{aligned}
\iint_T f_1\,\phi_1 + f_2\,\phi_2\; dA &\approx \text{area}(T)\,\Big(\langle \mathbf{M}\,\vec{\phi}_1\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \mathbf{M}\,\vec{\phi}_2\,,\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big) \\
&= \text{area}(T)\,\Big(\langle \vec{\phi}_1\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_1\rangle + \langle \vec{\phi}_2\,,\,\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}\,\vec{f}_2\rangle\Big)\,.
\end{aligned}$$

Thus find one contribution to (51). Observe that $\mathbf{M}^T\,\text{diag}(\vec{w})\,\mathbf{M}$ is a 10×10 matrix, independent on the triangle T.

7.4.2 Integration of the terms involving derivatives of 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 (43) for the partial derivatives and the transformation rule (31) 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 to obtain the three diagonal matrices

$$\mathbf{A}_1 = \operatorname{diag}\begin{pmatrix} w_1\,\frac{E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)} \\ w_2\,\frac{E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)} \\ \vdots \\ w_7\,\frac{E(\vec{g}_7)}{1-\nu^2(\vec{g}_7)} \end{pmatrix}\,,\quad \mathbf{A}_2 = \operatorname{diag}\begin{pmatrix} w_1\,\frac{\nu(\vec{g}_1)\,E(\vec{g}_1)}{1-\nu^2(\vec{g}_1)} \\ w_2\,\frac{\nu(\vec{g}_2)\,E(\vec{g}_2)}{1-\nu^2(\vec{g}_2)} \\ \vdots \\ w_7\,\frac{E(\nu(\vec{g}_7)\,\vec{g}_7)}{1-\nu^2(\vec{g}_7)} \end{pmatrix} \quad\text{and}\quad \mathbf{A}_3 = \operatorname{diag}\begin{pmatrix} w_1\,\frac{E(\vec{g}_1)}{2\,(1+\nu(\vec{g}_1))} \\ w_2\,\frac{E(\vec{g}_2)}{2\,(1+\nu(\vec{g}_2))} \\ \vdots \\ w_7\,\frac{E(\vec{g}_7)}{2\,(1+\nu(\vec{g}_7))} \end{pmatrix}\,.$$

This leads to the approximations

$$\begin{aligned}
\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_1}{\partial x}\,(\frac{\partial\,u_1}{\partial x}+\nu\,\frac{\partial\,u_2}{\partial y})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_1}{\partial y}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\
&\approx \langle \mathbf{A}_1\,\mathbf{G}_x\vec{\phi}_1,\mathbf{G}_x\vec{u}_1\rangle + \langle \operatorname{diag}\mathbf{A}_2\,\mathbf{G}_x\vec{\phi}_1,\mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_3\,\mathbf{G}_y\vec{\phi}_1,\mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\
&= \langle \vec{\phi}_1,\mathbf{G}_x^T\mathbf{A}_1\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_1,\mathbf{G}_x^T\mathbf{A}_2\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_1,\mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1+\mathbf{G}_y^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle \\
\frac{I_{\phi_1}}{\text{area}(T)} &= \frac{1}{\text{area}(T)}\iint\limits_T \frac{E}{1-\nu^2}\left(\frac{\partial\,\phi_2}{\partial y}\,(\frac{\partial\,u_2}{\partial y}+\nu\,\frac{\partial\,u_1}{\partial x})+\frac{1-\nu}{2}\,\frac{\partial\,\phi_2}{\partial x}\,(\frac{\partial\,u_1}{\partial y}+\frac{\partial\,u_2}{\partial x})\right)\,dA \\
&\approx \langle \mathbf{A}_1\mathbf{G}_y\vec{\phi}_2,\mathbf{G}_y\vec{u}_2\rangle + \langle \mathbf{A}_2\mathbf{G}_y\vec{\phi}_2,\mathbf{G}_x\vec{u}_1\rangle + \langle \mathbf{A}_3\mathbf{G}_x\vec{\phi}_2,\mathbf{G}_y\vec{u}_1+\mathbf{G}_x\vec{u}_2\rangle \\
&= \langle \vec{\phi}_2,\mathbf{G}_y^T\mathbf{A}_1\mathbf{G}_y\vec{u}_2\rangle + \langle \vec{\phi}_2,\mathbf{G}_y^T\mathbf{A}_2\mathbf{G}_x\vec{u}_1\rangle + \langle \vec{\phi}_2,\mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_y\vec{u}_1+\mathbf{G}_x^T\mathbf{A}_3\mathbf{G}_x\vec{u}_2\rangle\,.
\end{aligned}$$

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 20×20 matrix $\mathbf{G}\in\mathbb{R}^{20\times 20}$ is the element stiffness matrix for the triangle T, containing contributions to (51).

7.4.3 The boundary integral

The boundary integral is similar to (6.6.7) on page 127, 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.

7.5 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 (16) $\nu^\star = \frac{\nu}{1-\nu}$ 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 (12) 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

$$W_{strain} = \frac{1}{2}\,\frac{E}{(1+\nu)\,(1-2\,\nu)}\,\langle\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & 2\,(1-2\,\nu)\end{bmatrix}\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\,,\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{pmatrix}\rangle$$

$$\begin{aligned}
&= \frac{E\,(1-\nu)}{2\,(1+\nu)\,(1-2\,\nu)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\frac{\nu}{1-\nu}\,\varepsilon_{xx}\varepsilon_{yy}+2\,\frac{1-2\,\nu}{1-\nu}\,\varepsilon_{xy}^2\right)\\
&= \frac{E^\star}{2\,(1-(\nu^\star)^2)}\,\left(\varepsilon_{xx}^2+\varepsilon_{yy}^2+2\,\nu^\star\,\varepsilon_{xx}\varepsilon_{yy}+2\,(1-\nu^\star)\,\varepsilon_{xy}^2\right)\;. \qquad (52)
\end{aligned}$$

This is very similar to the elastic energy density (13) 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.

7.6 Elasticity for axially symmetric setups

Examine displacements

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

and the total energy to be minimized is given by expression (18) on page 17.

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

Use the strains

$$\begin{pmatrix}\varepsilon_{rr}\\ \varepsilon_{\phi\phi}\\ \varepsilon_{zz}\\ \varepsilon_{rz}\end{pmatrix} = \begin{pmatrix}\frac{\partial u_r}{\partial r}\\ \frac{1}{r}\,u_r\\ \frac{\partial u_z}{\partial z}\\ \frac12\,(\frac{\partial u_r}{\partial z}+\frac{\partial u_z}{\partial r})\end{pmatrix}$$

to rewrite the above with the displacement functions u_r and u_z in the form

$$\begin{aligned}
\frac{U(\vec u)}{2\,\pi} &= \iint\limits_\Omega \frac{r\,E}{2\,(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,((\frac{\partial u_r}{\partial r})^2+(\frac{\partial u_z}{\partial z})^2+\frac{1}{r^2}\,u_r^2)+\right.\\
&\qquad \left.+2\,\nu\,((\frac{\partial u_r}{\partial r})(\frac{\partial u_z}{\partial z})+\frac{1}{r}\,u_r\,(\frac{\partial u_r}{\partial r}+\frac{\partial u_z}{\partial z}))\right)\,dA+\\
&\quad+\iint\limits_\Omega \frac{r\,E}{1+\nu}\,\frac14\,(\frac{\partial u_r}{\partial z}+\frac{\partial u_z}{\partial r})^2\,dA-\\
&\quad-\iint\limits_\Omega r\,(f_r\,u_r+f_z\,u_z)\,dA-\int_{\Gamma_2} r\,(g_{Nr}\,u_r+g_{Nz}\,u_z)\,ds\;.
\end{aligned}$$

Expanding and ignoring quadratic terms of the perturbation $\vec\phi$ leads to

$$\begin{aligned}
\frac{U(\vec u+\vec\phi)}{2\,\pi} &\approx \frac{U(\vec u)}{2\,\pi}+\iint\limits_\Omega \frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\,\left((1-\nu)\,\left(\frac{\partial u_r}{\partial r}\,\frac{\partial \phi_r}{\partial r}+\frac{\partial u_z}{\partial z}\,\frac{\partial \phi_z}{\partial z}+\frac{1}{r^2}\,u_r\,\phi_r\right)+\right.\\
&\quad \left.+\nu\,\left((\frac{\partial u_r}{\partial r})(\frac{\partial \phi_z}{\partial z})+(\frac{\partial \phi_r}{\partial r})(\frac{\partial u_z}{\partial z})+\frac{1}{r}\,u_r\,(\frac{\partial \phi_r}{\partial r}+\frac{\partial \phi_z}{\partial z})+\frac{1}{r}\,\phi_r\,(\frac{\partial u_r}{\partial r}+\frac{\partial u_z}{\partial z})\right)\right)\,dA+
\end{aligned}$$

$$+\iint\limits_{\Omega} \frac{r\,E}{1+\nu}\,\frac{1}{2}\left(\frac{\partial\,u_r}{\partial z}\frac{\partial\,\phi_r}{\partial z}+\frac{\partial\,u_z}{\partial r}\frac{\partial\,\phi_z}{\partial r}+\frac{\partial\,u_r}{\partial z}\frac{\partial\,\phi_z}{\partial r}+\frac{\partial\,\phi_r}{\partial z}\frac{\partial\,u_z}{\partial r}\right)\,dA-$$
$$-\iint\limits_{\Omega} r\,(f_r\,\phi_r+f_z\,\phi_z)\,dA-\int_{\Gamma_2} r\,(g_{Nr}\,\phi_r+g_{Nz}\,\phi_z)\,ds\;.$$

These integrals can be separated into contributions with ϕ_r, ϕ_z, f and g_N.

$$\begin{aligned}
\frac{U(\vec u+\vec\phi)}{2\,\pi} &\approx \frac{U(\vec u)}{2\,\pi}+I_{\phi_r}+I_{\phi_z}+I_f+I_g && (53)\\
I_{\phi_r} &= \iint\limits_{\Omega}\frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\left((1-\nu)\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r}+\frac{1}{r^2}\,u_r\,\phi_r)+\right. && (54)\\
&\qquad \left.+\nu\left(\frac{\partial\,u_z}{\partial z}\frac{\partial\,\phi_r}{\partial r}+\frac{1}{r}\,u_r\,\frac{\partial\,\phi_r}{\partial r}+\frac{1}{r}\,\phi_r\,(\frac{\partial\,u_r}{\partial r}+\frac{\partial\,u_z}{\partial z})\right)\right)+\\
&\qquad +\frac{r\,E}{2\,(1+\nu)}\left(\frac{\partial\,u_r}{\partial z}+\frac{\partial\,u_z}{\partial r}\right)\frac{\partial\,\phi_r}{\partial z}\,dA\\
I_{\phi_z} &= \iint\limits_{\Omega}\frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\left((1-\nu)\,\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z}+\nu\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_z}{\partial z}+\frac{1}{r}\,u_r\,\frac{\partial\,\phi_z}{\partial z})\right)+ && (55)\\
&\qquad +\frac{r\,E}{2\,(1+\nu)}\left(\frac{\partial\,u_r}{\partial z}+\frac{\partial\,u_z}{\partial r}\right)\frac{\partial\,\phi_z}{\partial r}\,dA\\
I_f &= -\iint\limits_{\Omega} r\,(f_r\,\phi_r+f_z\,\phi_z)\,dA && (56)\\
I_g &= -\int_{\Gamma_2} r\,(g_{Nr}\,\phi_r+g_{Nz}\,\phi_z)\,ds && (57)
\end{aligned}$$

7.7 Construction of first order elements

This is similar to the computations in Section 7.2, starting in page 138. In this section the element stiffness matrix is constructed. Then use the procedure in Section 6.4 (page 101) to determine the global stiffness matrix.

7.7.1 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 $\mathbf{R}=\operatorname{diag}([r_1,r_2,r_3])$ 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

$$\begin{aligned}
\iint\limits_{T} r\,(f_r\,\phi_r+f_z\,\phi_z)\,dA &\approx \frac{\text{area}(T)}{3}\left(\langle\mathbf{M}\,\vec\phi_r\,,\,\mathbf{R}\vec f_r\rangle+\langle\mathbf{M}\,\vec\phi_z\,,\,\mathbf{R}\vec f_z\rangle\right)\\
&= \frac{\text{area}(T)}{3}\left(\langle\vec\phi_r\,,\,\mathbf{M}^T\mathbf{R}\vec f_r\rangle+\langle\vec\phi_z\,,\,\mathbf{M}^T\mathbf{R}\vec f_z\rangle\right)\;.
\end{aligned}$$

- 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

$$\begin{aligned}\iint_T r\,(f_r\,\phi_r + f_z\,\phi_z)\;dA &\approx \frac{\text{area}(T)}{3}\,\left(\langle \mathbf{M}\,\vec{\phi}_r\,,\,\mathbf{RM}\,\vec{f}_r\rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\,\mathbf{RM}\,\vec{f}_z\rangle\right)\\ &= \frac{\text{area}(T)}{3}\,\left(\langle \vec{\phi}_r\,,\,\mathbf{M}^T\mathbf{RM}\,\vec{f}_r\rangle + \langle \vec{\phi}_z\,,\,\mathbf{M}^T\mathbf{RM}\,\vec{f}_z\rangle\right)\end{aligned}$$

With the above the contributions in (56) for each element stiffness matrix can be determined.

7.7.2 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 (25) given by

$$\nabla u = \frac{-1}{2\,\text{area}(T)}\begin{bmatrix}(z_3-z_2) & (z_1-z_3) & (z_2-z_1)\\ (r_2-r_3) & (r_3-r_1) & (r_1-r_2)\end{bmatrix}\cdot\vec{u} = \begin{bmatrix}\mathbf{G}_r\\ \mathbf{G}_z\end{bmatrix}\,\vec{u}\,.$$

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

$$\begin{aligned}I_{\phi_r} &= \iint_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\,\Big((1-\nu)\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r^2}\,u_r\,\phi_r) +\\ &\qquad +\nu\,\Big(\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,u_r\,\frac{\partial\,\phi_r}{\partial r} + \frac{1}{r}\,\phi_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z})\Big)\Big) +\\ &\qquad +\frac{r\,E}{2\,(1+\nu)}\,\Big(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}\Big)\,\frac{\partial\,\phi_r}{\partial z}\;dA\\ &\approx a_1\,\langle \mathbf{G}_r\vec{\phi}_r, \mathbf{G}_r\vec{u}_r\rangle + \langle \mathbf{M}\vec{\phi}_r, \mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + a_3\,\langle \mathbf{G}_r\vec{\phi}_r, \mathbf{G}_z\vec{u}_z\rangle +\\ &\qquad +\langle \vec{a}_4\mathbf{G}_r\vec{\phi}_r, \mathbf{M}\vec{u}_r\rangle + \langle \mathbf{M}\vec{\phi}_r, \vec{a}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + a_5\,\langle \mathbf{G}_z\vec{\phi}_r, (\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\ &= a_1\,\langle \vec{\phi}_r, \mathbf{G}_r^T\mathbf{G}_r\vec{u}_r\rangle + \langle \vec{\phi}_r, \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r\rangle + a_3\,\langle \vec{\phi}_r, \mathbf{G}_r^T\mathbf{G}_z\vec{u}_z\rangle +\\ &\qquad +\langle \vec{\phi}_r, (\vec{a}_4\mathbf{G}_r)^T\mathbf{M}\vec{u}_r\rangle + \langle \vec{\phi}_r, \mathbf{M}^T\vec{a}_4\,(\mathbf{G}_r\vec{u}_r + \mathbf{G}_z\vec{u}_z)\rangle + a_5\,\langle \vec{\phi}_r, \mathbf{G}_z^T(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle\\ &= \langle a_1\,\mathbf{G}_r^T\mathbf{G}_r\vec{u}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M}\vec{u}_r + a_3\,\mathbf{G}_r^T\mathbf{G}_z\vec{u}_z + (\vec{a}_4\mathbf{G}_r)^T\mathbf{M}\vec{u}_r, \vec{\phi}_r\rangle +\end{aligned}$$

$$
\begin{aligned}
& +\langle \mathbf{M}^T\vec{a}_4\,(\mathbf{G}_r\vec{u}_r+\mathbf{G}_z\vec{u}_z)+a_5\,\mathbf{G}_z^T(\mathbf{G}_z\vec{u}_r+\mathbf{G}_r\vec{u}_z),\vec{\phi}_r\rangle \\
= & \langle\big(a_1\,\mathbf{G}_r^T\mathbf{G}_r+\mathbf{M}^T\mathbf{A}_2\mathbf{M}+(\vec{a}_4\mathbf{G}_r)^T\mathbf{M}+\mathbf{M}^T\vec{a}_4\,\mathbf{G}_r+a_5\,\mathbf{G}_z^T\mathbf{G}_z\big)\ \vec{u}_r,\vec{\phi}_r\rangle \\
& +\langle\big(a_3\,\mathbf{G}_r^T\mathbf{G}_z+\mathbf{M}^T\vec{a}_4\,\mathbf{G}_z+a_5\,\mathbf{G}_z^T\mathbf{G}_r\big)\ \vec{u}_z,\vec{\phi}_r\rangle
\end{aligned}
$$

and

$$
\begin{aligned}
I_{\phi_z} &= \iint\limits_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)}\left((1-\nu)\,\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z}+\nu\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_z}{\partial z}+\frac{1}{r}\,u_r\,\frac{\partial\,\phi_z}{\partial z})\right)+ \\
&\quad +\frac{r\,E}{2\,(1+\nu)}\left(\frac{\partial\,u_r}{\partial z}+\frac{\partial\,u_z}{\partial r}\right)\frac{\partial\,\phi_z}{\partial r}\,dA \\
&\approx a_1\,\langle\mathbf{G}_z\vec{\phi}_z,\mathbf{G}_z\vec{u}_z\rangle+a_3\,\langle\mathbf{G}_z\vec{\phi}_z,\mathbf{G}_r\vec{u}_r\rangle+\langle\vec{a}_4\mathbf{G}_z\vec{\phi}_z,\mathbf{M}\vec{u}_r\rangle+a_5\,\langle\mathbf{G}_r\vec{\phi}_z,(\mathbf{G}_z\vec{u}_r+\mathbf{G}_r\vec{u}_z)\rangle \\
&= a_1\,\langle\vec{\phi}_z,\mathbf{G}_z^T\mathbf{G}_z\vec{u}_z\rangle+a_3\,\langle\vec{\phi}_z,\mathbf{G}_z^T\mathbf{G}_r\vec{u}_r\rangle+\langle\vec{\phi}_z,(\vec{a}_4\mathbf{G}_z)^T\mathbf{M}\vec{u}_r\rangle+a_5\,\langle\vec{\phi}_z,\mathbf{G}_r^T(\mathbf{G}_z\vec{u}_r+\mathbf{G}_r\vec{u}_z)\rangle \\
&= \langle a_1\,\mathbf{G}_z^T\mathbf{G}_z\vec{u}_z+a_3\,\mathbf{G}_z^T\mathbf{G}_r\vec{u}_r+(\vec{a}_4\mathbf{G}_z)^T\mathbf{M}\vec{u}_r+a_5\,\mathbf{G}_r^T(\mathbf{G}_z\vec{u}_r+\mathbf{G}_r\vec{u}_z),\vec{\phi}_z\rangle \\
&= \langle\big(a_3\,\mathbf{G}_z^T\mathbf{G}_r+(\vec{a}_4\mathbf{G}_z)^T\mathbf{M}+a_5\,\mathbf{G}_r^T\mathbf{G}_z\big)\,\vec{u}_r,\vec{\phi}_z\rangle+\langle\big(a_1\,\mathbf{G}_z^T\mathbf{G}_z+a_5\,\mathbf{G}_r^T\mathbf{G}_r\big)\,\vec{u}_z,\vec{\phi}_z\rangle
\end{aligned}
$$

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 $\mathbf{A}\in\mathbb{M}^{6\times6}$, where $\mathbf{A}_{r,z}\in\mathbb{M}^{3\times3}$. For sake of completeness here the 6×6 element stiffness matrix.

$$
\mathbf{A}=\begin{bmatrix} a_1\mathbf{G}_r^T\mathbf{G}_r+\mathbf{M}^T\mathbf{A}_2\mathbf{M}+(\vec{a}_4\mathbf{G}_r)^T\mathbf{M}+\mathbf{M}^T\vec{a}_4\mathbf{G}_r+a_5\mathbf{G}_z^T\mathbf{G}_z & a_3\mathbf{G}_r^T\mathbf{G}_z+\mathbf{M}^T\vec{a}_4\mathbf{G}_z+a_5\mathbf{G}_z^T\mathbf{G}_r \\ a_3\mathbf{G}_z^T\mathbf{G}_r+(\vec{a}_4\mathbf{G}_z)^T\mathbf{M}+a_5\mathbf{G}_r^T\mathbf{G}_z & a_1\mathbf{G}_z^T\mathbf{G}_z+a_5\mathbf{G}_r^T\mathbf{G}_r \end{bmatrix}
$$

7.7.3 The boundary integral

The boundary integral is similar to (26) on page 107. With $\alpha=\frac{1-1/\sqrt{3}}{2}$ use the symmetric interpolation matrix from nodes to Gauss points

$$
\mathbf{M}_b=\begin{bmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{bmatrix}=\mathbf{M}_b^T
$$

to find te 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.

7.8 Construction of second order elements

To construct elements of order 2 combine procedures from Section 7.3 for second order elements for plane stress problems and the previous section 7.7 where first order elements are generated for axisymmetric problems.

7.8.1 Integration of $r\,(f_r\,\phi_r+f_z\,\phi_z)$

Use the Gauss weights $\vec{w}\in\mathbb{R}^7$ from equation (22) on page 101 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

$$
\mathbf{RW}=\mathrm{diag}([r_1w_1,r_2w_2,\ldots,r_7w_7])\in\mathbb{M}^{7\times7}\ .
$$

- 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

$$\begin{aligned}\iint\limits_T r\,(f_r\,\phi_r + f_z\,\phi_z)\;dA \quad &\approx \quad \text{area}(T)\,\left(\langle \mathbf{M}\,\vec{\phi}_r\,,\,\text{diag}(\vec{w})\,\vec{f}_r\rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\,\mathbf{RW}\,\vec{f}_z\rangle\right)\\ &= \quad \text{area}(T)\,\left(\langle \vec{\phi}_r\,,\,\mathbf{M}^T\mathbf{RW}\,\vec{f}_r\rangle + \langle \vec{\phi}_z\,,\,\mathbf{M}^T\mathbf{RW}\,\vec{f}_z\rangle\right)\\ &= \quad \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_r\\ \vec{\phi}_z\end{pmatrix}, \begin{bmatrix}\mathbf{M}^T\mathbf{RW} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\mathbf{RW}\end{bmatrix}\begin{pmatrix}\vec{f}_r\\ \vec{f}_z\end{pmatrix}\rangle\,.\end{aligned}$$

  $\mathbf{M} \in \mathbb{R}^{7\times 6}$ is the matrix for interpolation from the nodes to the Gauss points, given in equation (29) on page 109.

- 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

$$\begin{aligned}\iint\limits_T r\,(f_r\,\phi_r + f_z\,\phi_z)\;dA \quad &\approx \quad \text{area}(T)\,\left(\langle \mathbf{M}\,\vec{\phi}_r\,,\,\mathbf{RW}\,\mathbf{M}\,\vec{f}_r\rangle + \langle \mathbf{M}\,\vec{\phi}_z\,,\,\mathbf{RW}\,\mathbf{M}\,\vec{f}_z\rangle\right)\\ &= \quad \text{area}(T)\,\left(\langle \vec{\phi}_r\,,\,\mathbf{M}^T\mathbf{RW}\,\mathbf{M}\,\vec{f}_r\rangle + \langle \vec{\phi}_z\,,\,\mathbf{M}^T\mathbf{RW}\,\mathbf{M}\,\vec{f}_z\rangle\right)\\ &= \quad \text{area}(T)\,\langle \begin{pmatrix}\vec{\phi}_r\\ \vec{\phi}_z\end{pmatrix}, \begin{bmatrix}\mathbf{M}^T\mathbf{RW}\,\mathbf{M} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}^T\mathbf{RW}\,\mathbf{M}\end{bmatrix}\begin{pmatrix}\vec{f}_r\\ \vec{f}_z\end{pmatrix}\rangle\,.\end{aligned}$$

With the above the contributions in (56) 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.

7.8.2 Integration of the terms involving derivatives of ϕ_z 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

$$\frac{\partial}{\partial r}\,\vec{\phi} = \frac{1}{\det(\mathbf{T})}\,\left[\,(-r_3 + r_1)\,\mathbf{M}_\xi^T + (+r_2 - r_1)\,\mathbf{M}_\nu^T\,\right]\cdot\vec{\phi} =: \mathbf{G}_z\,\vec{\phi}\,.$$

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) + \\
&\quad +\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)\Big) + \\
&\quad +\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 \\
&\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

$$\begin{aligned}
I_{\phi_z} &= \iint\limits_T \frac{r\,E}{(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,\frac{\partial\,u_z}{\partial z}\,\frac{\partial\,\phi_z}{\partial z} + \nu\,(\frac{\partial\,u_r}{\partial r}\,\frac{\partial\,\phi_z}{\partial z} + \frac{1}{r}\,u_r\,\frac{\partial\,\phi_z}{\partial z})\right) + \\
&\quad +\frac{r\,E}{2\,(1+\nu)}\,\left(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r}\right)\,\frac{\partial\,\phi_z}{\partial r}\,dA \\
&\approx \langle \mathbf{G}_z\vec{\phi}_z, \mathbf{A}_1\mathbf{G}_z\vec{u}_z\rangle + \langle \mathbf{G}_z\vec{\phi}_z, \mathbf{A}_3\mathbf{G}_r\vec{u}_r\rangle + \langle \mathbf{A}_4\mathbf{G}_z\vec{\phi}_z, \mathbf{M}\vec{u}_r\rangle + \langle \mathbf{G}_r\vec{\phi}_z, \mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= \langle \vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z\vec{u}_z\rangle + \langle \vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r\vec{u}_r\rangle + \langle \vec{\phi}_z, \mathbf{G}_z^T\mathbf{A}_4\mathbf{M}\vec{u}_r\rangle + \langle \vec{\phi}_z, \mathbf{G}_r^T\mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z)\rangle \\
&= \langle \mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z\vec{u}_z + \mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r\vec{u}_r + \mathbf{G}_z^T\mathbf{A}_4\mathbf{M}\vec{u}_r + \mathbf{G}_r^T\mathbf{A}_5(\mathbf{G}_z\vec{u}_r + \mathbf{G}_r\vec{u}_z), \vec{\phi}_z\rangle \\
&= \langle \big(\mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r + \mathbf{G}_z^T\mathbf{A}_4\mathbf{M} + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_z\big)\,\vec{u}_r, \vec{\phi}_z\rangle + \langle \big(\mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_r\big)\,\vec{u}_z, \vec{\phi}_z\rangle
\end{aligned}$$

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 element stiffness matrix $\mathbf{A} \in \mathbb{M}^{12\times 12}$, where $\mathbf{A}_{r,z} \in \mathbb{M}^{6\times 6}$. For sake of completeness here the 12×12 element stiffness matrix $\mathbf{A}$.

$$\begin{bmatrix} \mathbf{G}_r^T\mathbf{A}_1\mathbf{G}_r + \mathbf{M}^T\mathbf{A}_2\mathbf{M} + \mathbf{G}_r^T\mathbf{A}_4\mathbf{M} + \mathbf{M}^T\mathbf{A}_4\mathbf{G}_r + \mathbf{G}_z^T\mathbf{A}_5\mathbf{G}_z & \mathbf{G}_r^T\mathbf{A}_3\mathbf{G}_z + \mathbf{M}^T\mathbf{A}_4\mathbf{G}_z + \mathbf{G}_z^T\mathbf{A}_5\mathbf{G}_r \\ \mathbf{G}_z^T\mathbf{A}_3\mathbf{G}_r + \mathbf{G}_z^T\mathbf{A}_4\mathbf{M} + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_z & \mathbf{G}_z^T\mathbf{A}_1\mathbf{G}_z + \mathbf{G}_r^T\mathbf{A}_5\mathbf{G}_r \end{bmatrix}$$

7.8.3 The boundary integral

The boundary integral is constucted similar to the procedures in Section 7.3.3, 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

$$\begin{aligned}
\int_{\text{edge}} r\,(g_r\,\phi_r + g_z\,\phi_z)\;ds \quad &\approx \quad \frac{L}{18}\,\langle \mathbf{M}_B\vec{\phi}_r\,,\begin{pmatrix} 5\,r_1\,g_r(\vec{p}_1) \\ 8\,r_2\,g_r(\vec{p}_2) \\ 5\,r_3\,g_r(\vec{p}_3)\end{pmatrix}\rangle + \frac{L}{18}\,\langle \mathbf{M}_B\vec{\phi}_z\,,\begin{pmatrix} 5\,r_1\,g_z(\vec{p}_1) \\ 8\,r_2\,g_z(\vec{p}_2) \\ 5\,r_3\,g_z(\vec{p}_3)\end{pmatrix}\rangle \\
&= \quad \frac{L}{18}\,\langle \vec{\phi}_r\,,\,\mathbf{M}_B^T\begin{pmatrix} 5\,r_1\,g_r(\vec{p}_1) \\ 8\,r_2\,g_r(\vec{p}_2) \\ 5\,r_3\,g_r(\vec{p}_3)\end{pmatrix}\rangle + \frac{L}{18}\,\langle \vec{\phi}_z\,,\,\mathbf{M}_B^T\begin{pmatrix} 5\,r_1\,g_z(\vec{p}_1) \\ 8\,r_2\,g_z(\vec{p}_2) \\ 5\,r_3\,g_z(\vec{p}_3)\end{pmatrix}\rangle
\end{aligned}$$

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 6.5.8.

7.9 Construction of third order elements

To construct elements of order 3 combine procedures from Section 7.4 for third order elements for plane stress problems and the previous section 7.8 where second order elements are generated for axisymmetric problems.

7.9.1 Integration of $r\,(f_r\,\phi_r + f_z\,\phi_z)$

The computations are identical to Section 7.8.1 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 (40) on page 121. The contributions in (56) for each element stiffness matrix can be determined. The matrix $\mathbf{M}^T\mathbf{RW}\,\mathbf{M}$ is a 10×10 matrix, independent on the triangle T.

7.9.2 Integration of the terms involving derivatives of ϕ_z and ϕ_z

The algorithm is extremely similar to Section 7.8.2 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 $\mathbf{A}_r$ and $\mathbf{A}_z$ are of size 10×10, leading to the element stiffness matrix $\mathbf{A} \in \mathbb{M}^{20\times 20}$.

7.9.3 The boundary integral

The algorithm is extremely similar to Section 7.8.3 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.

8 Examples, Examples, Examples

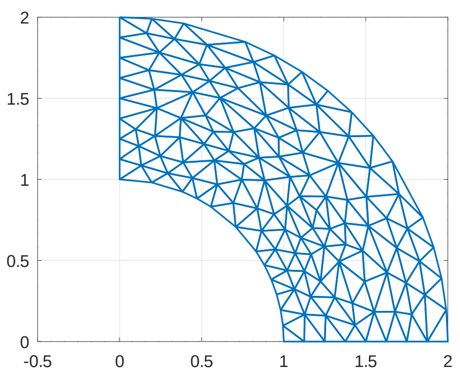
8.1 An elliptic problem with variable coefficients

The elliptic BVP in Section 5.5 is

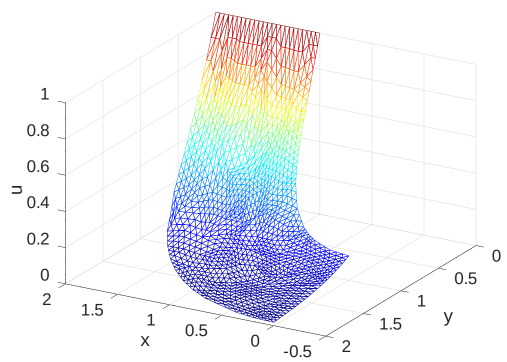
$$\begin{aligned}
-\nabla\left((1+x^2)\nabla u(x,y)\right) &= -4\,(1+x^2)\,\exp(-2\,y) && \text{for} \quad (x,y)\in\Omega \\
\frac{\partial\, u(y,0)}{\partial x} &= 0 && \text{for} \quad 1\le y\le 2 \\
u(x,y) &= \exp(-2\,y) && \text{on other sections of the boundary}
\end{aligned} .$$

on the domain shown in Figure 57(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 57(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 57(b) and then use `FEMIntegrate()` to determine the L_2–error
$$\left(\iint\limits_{\Omega} |u(x,y) - u_{exact}(x,y)|^2\, dA\right)^{1/2} .$$



(a) the mesh



(b) the solution

Figure 57: Difference to the exact solution of a BVP

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
```

8.2 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 58 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
```

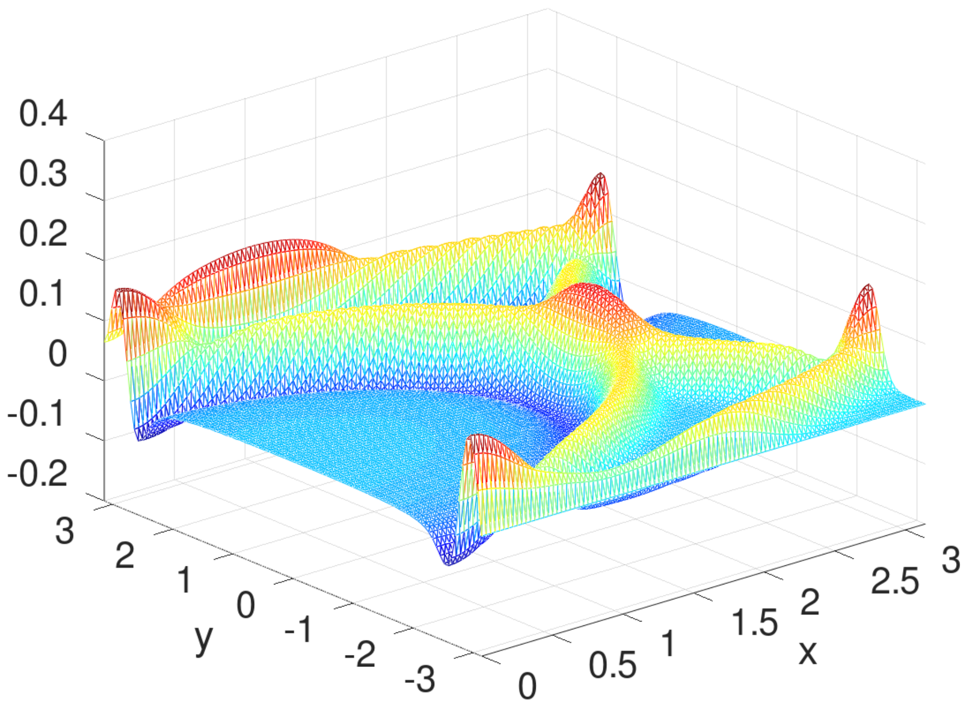


Figure 58: Traveling waves on a rectangle

8.3 An elliptic problem with radial symmetry, superconvergence

The Bessel function

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

is a 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 59. This BVP is solved by two slightly different approaches, and then the difference to the known exact solution is displayed in Figure 60. In both cases first a mesh with linear element 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 60(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.

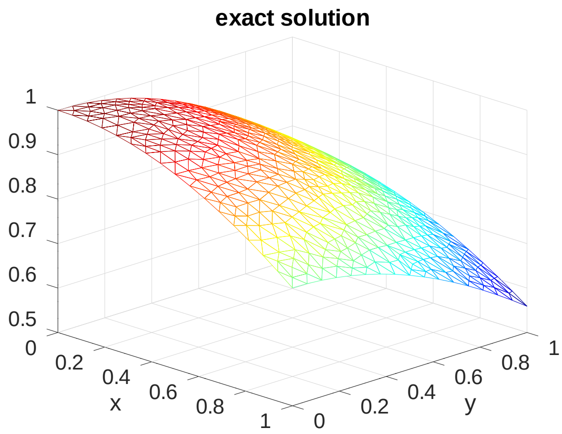


Figure 59: The radial Bessel function as solution of a BVP

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 60(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
  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
```

To generate Figure 61 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. This figure illustrates that the effect of superconvergence does not provide additional accuracy one can reliably count on.

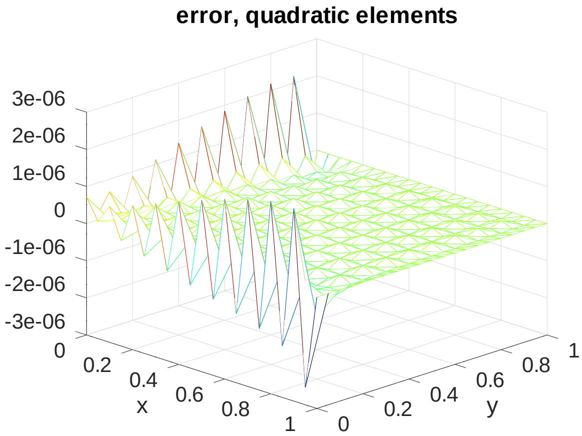
The gradient of this solution u can be determined using $\frac{\partial}{\partial r}\, J_0(r) = -J_1(r)$ and

$$\begin{pmatrix} \frac{\partial\, u}{\partial x} \\ \frac{\partial\, u}{\partial y} \end{pmatrix} = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{\partial\, u}{\partial r} + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{\partial\, u}{\partial \phi} = -\begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} J_1(r)\,.$$

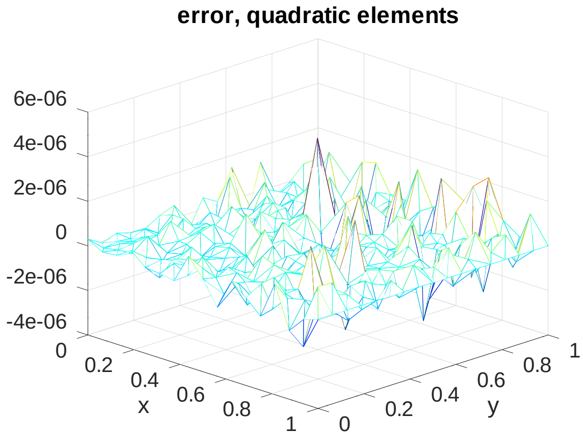
Using the above FEM results compare the true partial derivative $\frac{\partial\, u}{\partial x}$ with the one obtained by FEM with second order elements. Find the result in Figure 62. Observe the structure of the difference for the uniform mesh.

The above can be repeated using first order elements, leading to Figure 63. 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.

Find more information on superconvergence in [Zien13, §15.2] or a short demo in [Stah08, §6.8.2].

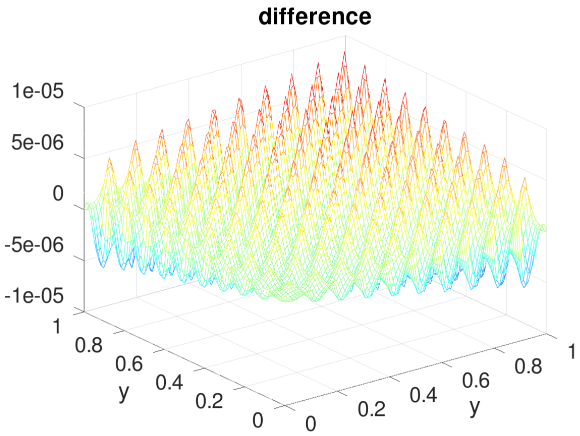


(a) uniform grid

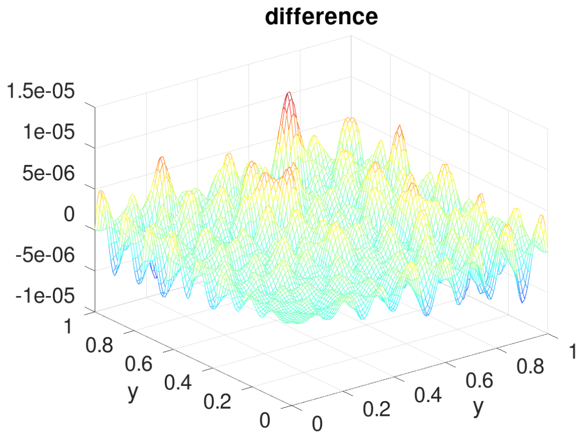


(b) nonumiform grid

Figure 60: Difference to the exact solution of a BVP

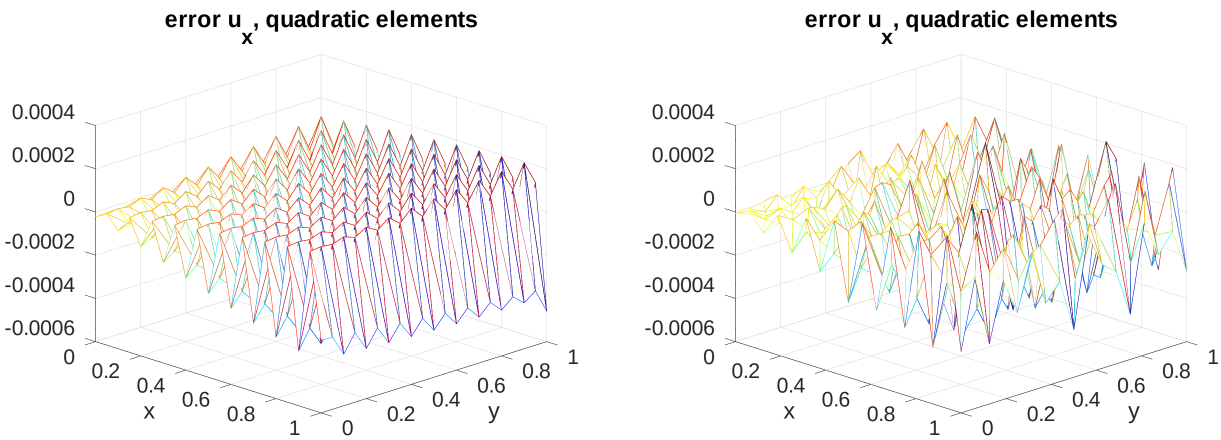


(a) uniform grid



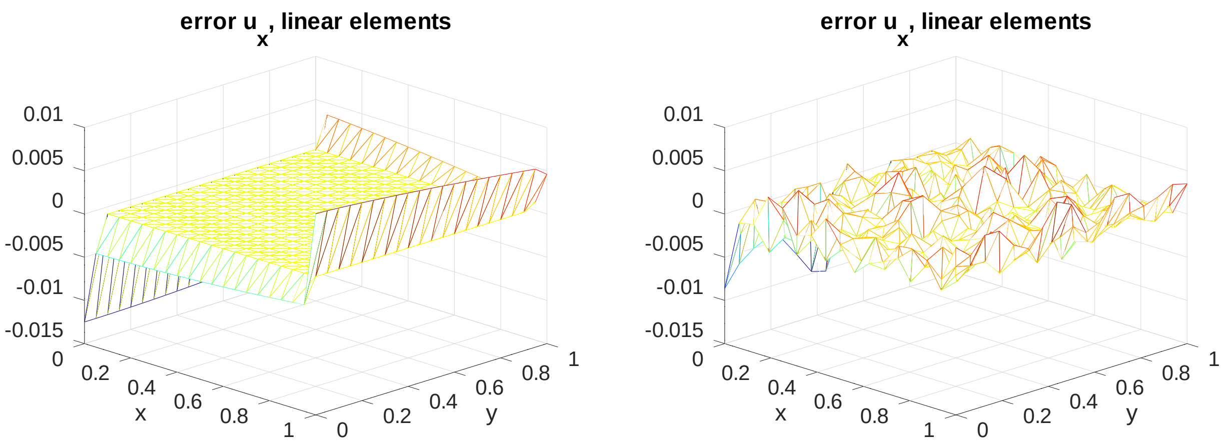
(b) nonumiform grid

Figure 61: Difference to the exact solution of a BVP, using quadratic elements and interpolation to a finer grid.



(a) uniform grid (b) nonumiform grid

Figure 62: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using second order elements



(a) uniform grid (b) nonumiform grid

Figure 63: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using first order elements

8.4 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

$$\begin{aligned} w(z) &= z^{2/3} = \left(r\,e^{i\phi}\right)^{2/3} = r^{2/3}\,e^{i\phi\,2/3} = r^{2/3}\,(\cos(\phi\,2/3) + i\,\sin(\phi\,2/3)) \\ u(z) &= r^{2/3}\,\sin(\phi\,2/3) \\ u(x,y) &= (x^2+y^2)^{1/3}\,\sin(\frac{2}{3}\,\text{atan2}(y,x)) \end{aligned}$$

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 64. 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])
```

The gradient in Cartesian coordinates can be determined by

$$\begin{aligned} \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} &= \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{\partial\,u}{\partial r} + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{\partial\,u}{\partial \phi} \\ &= \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \frac{2}{3}\,r^{-1/3}\,\sin(\phi\,2/3) + \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \frac{2}{3}\,r^{+2/3}\,\cos(\phi\,2/3) \end{aligned}$$

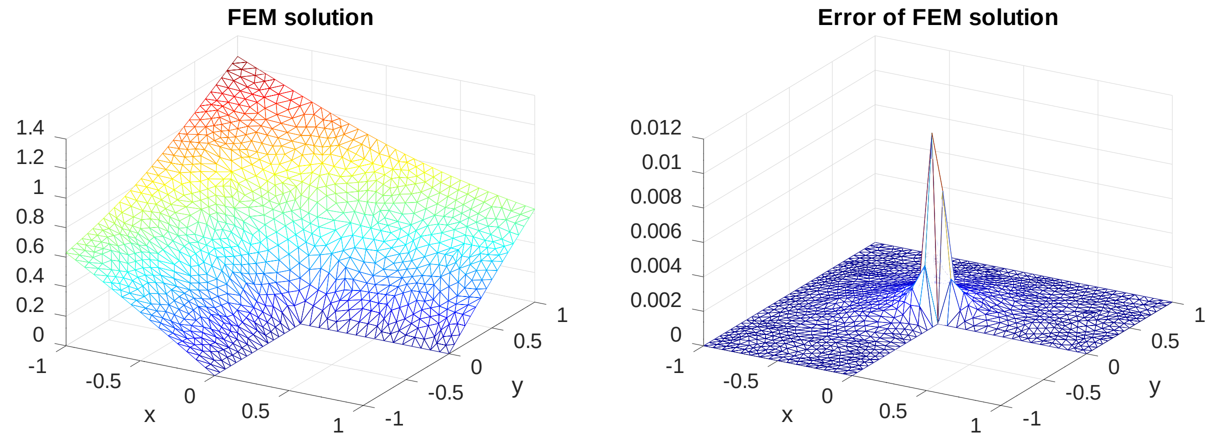


Figure 64: A solution with singular partial derivatives at the origin

and then visualized, leading to Figure 65. 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])
```

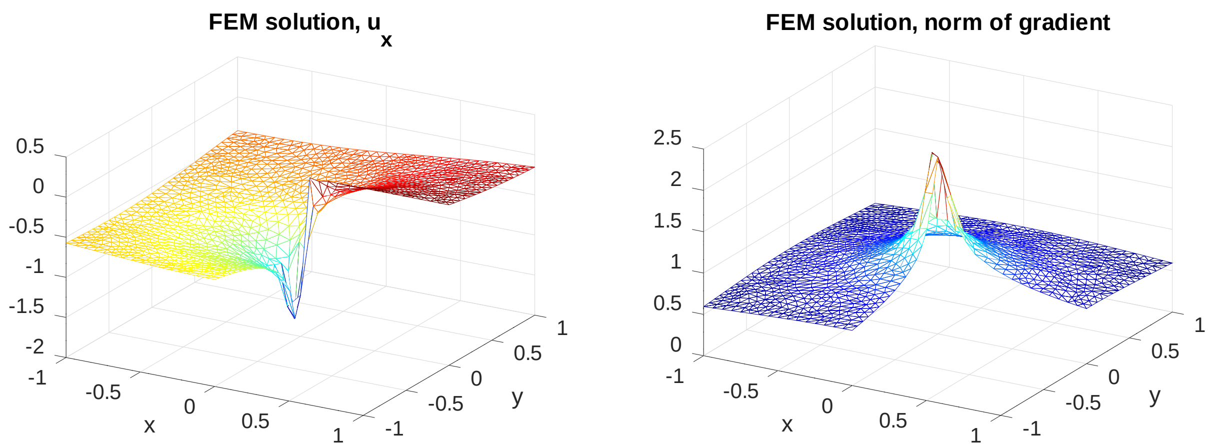


Figure 65: A solution with singular partial derivatives, graphs of $\frac{\partial u}{\partial x}$ and $\|\nabla u\|$

Singularities can show up in mechanical problems, e.g. the washer fastener in Section 8.18.

8.5 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 only shown in Figure 66. The goal is to find the velocity field $\vec{v}$ of the fluid.

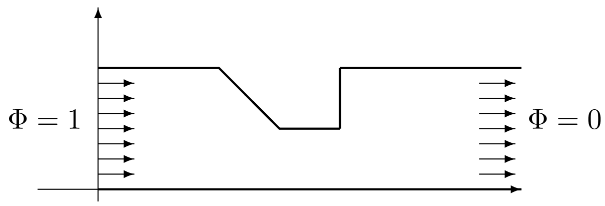


Figure 66: Fluid flow between two plates, the setup

This problem is 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

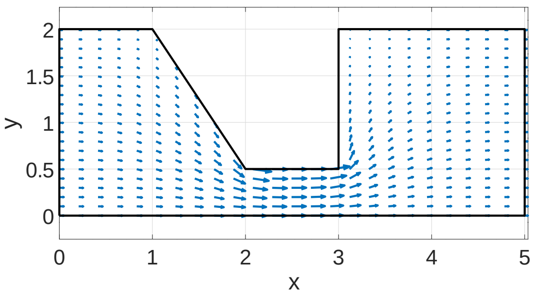
$$\Delta\Phi = \operatorname{div}(\operatorname{grad}\Phi) = 0$$

In Figure 67 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 8.4.

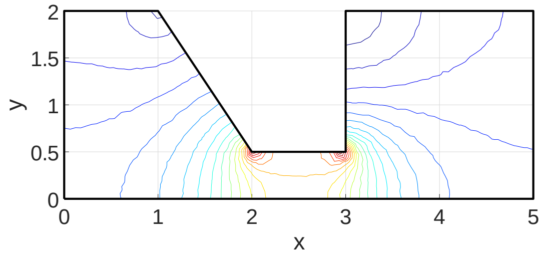
The results are generated by the code below.

PotentialFlow.m

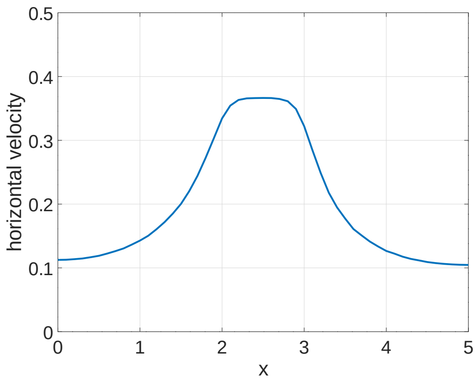
```
%% define the domain
xy = [0 0 -2; 5 0 -1;5 2 -2; 3 2 -2; 3 0.5 -2; 2 0.5 -2; 1 2 -2; 0 2 -1];
if 1      %% linear elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,0.003);
elseif 1  %% quadratic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,4*0.003);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
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);
```



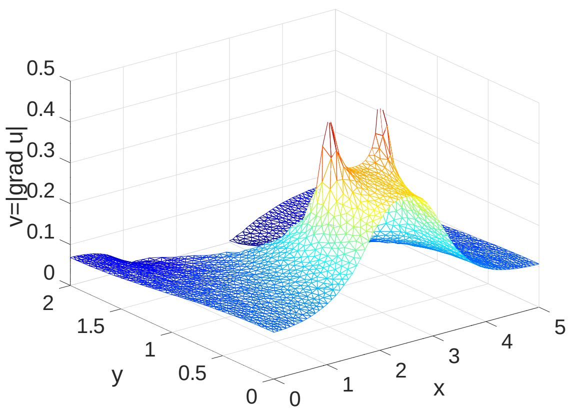
(a) field of velocity vectors



(b) velocity contours



(c) horizontal speed profile along $y = 0.25$



(d) the velocity

Figure 67: Velocity field of a ideal fluid

```
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
```

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)
```

8.6 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

$$R(z) = \begin{cases} 2 & \text{for} \quad |z| \geq 1 \\ 2 - \cos^2(\frac{\pi}{2}\, z) & \text{for} \quad |z| \leq 1 \end{cases} .$$

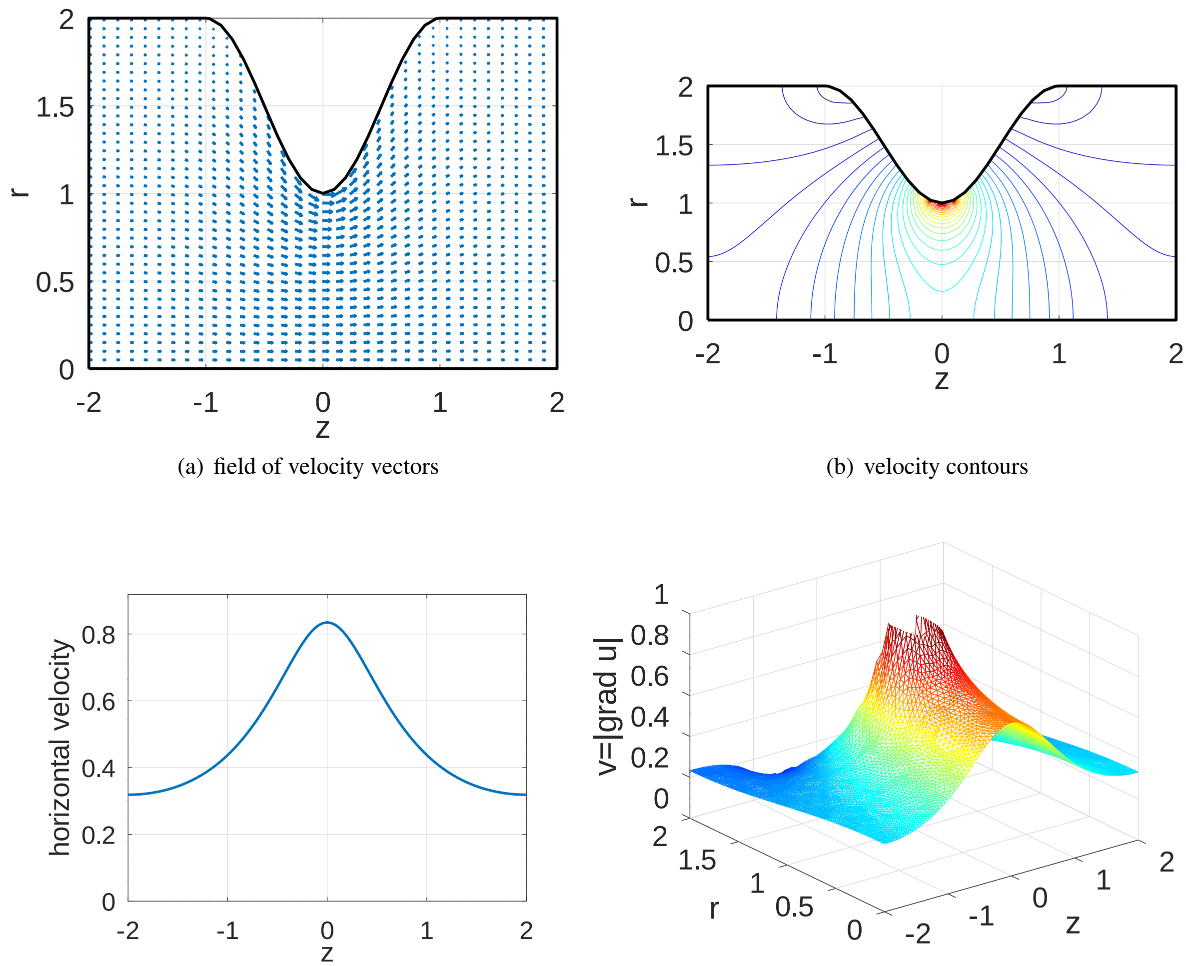
The upper half of a section is visible in Figure 68. Assuming that the solution is independent on the angle θ the equation $\Delta\Phi = 0$ has to be reformulated in cylindrical coordinates and simplified.

$$\begin{aligned} 0 &= \Delta\Phi \;=\; \operatorname{div}(\operatorname{grad}\Phi) \;=\; \Phi_{rr} + \frac{1}{r}\,\Phi_r + \frac{1}{r^2}\,\Phi_{\theta\theta} + \Phi_{zz} \\ 0 &= r\left(\Phi_{rr} + \frac{1}{r}\,\Phi_r + \Phi_{zz}\right) \;=\; r\,\Phi_{rr} + \Phi_r + r\,\Phi_{zz} \;=\; \frac{\partial}{\partial r}\,(r\,\Phi_r) + \frac{\partial}{\partial z}\,(r\,\Phi_z)\;. \end{aligned}$$

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 8.5, 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 68: 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 $z =$ 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.

8.7 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.

$$\begin{aligned}
\text{div}\left(\frac{1}{\sqrt{1+|\,\text{grad}\, u|^2}}\ \text{grad}\, u\right) &= 0 \qquad \text{in domain} \quad \Omega \\
u &= g \qquad \text{on} \quad \Gamma = \partial\Omega
\end{aligned}$$

FEMoctave is not directly capable of solving non linear problems, but a simple iteration will lead to an approxi-

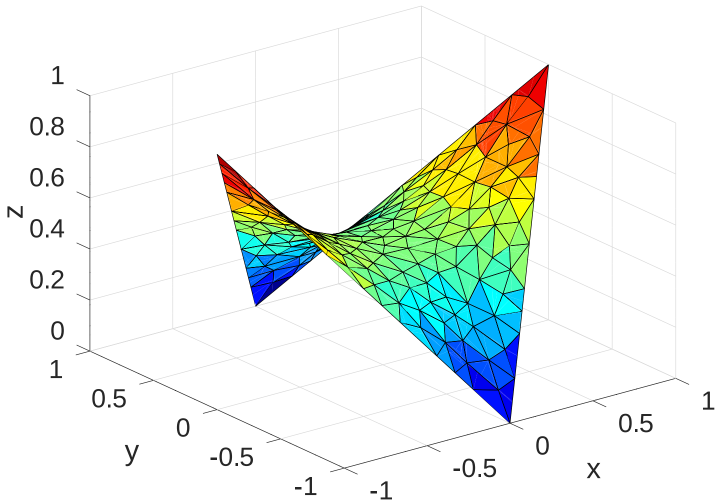


Figure 69: A minimal surface

mation 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

$$\begin{aligned}
\text{div}(a(x, y)\ \text{grad}\, u) &= 0 \qquad \text{in domain} \quad \Omega \\
u &= g \qquad \text{on} \quad \Gamma = \partial\Omega
\end{aligned}$$

The code below implements this algorithm for a square Ω and leads to the result in Figure 69. While iterating the area of each surface is determined by integrating

$$\text{area} = \iint\limits_{\Omega} \sqrt{1+|\nabla u|^2}\, dA$$

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.

8.8 Computing a capacitance

8.8.1 State the problem

Examine a circular plate capacitance as shown in Figure 70. Based on the radial symmetry one should be able to consider a two dimensional section only for the computations.

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

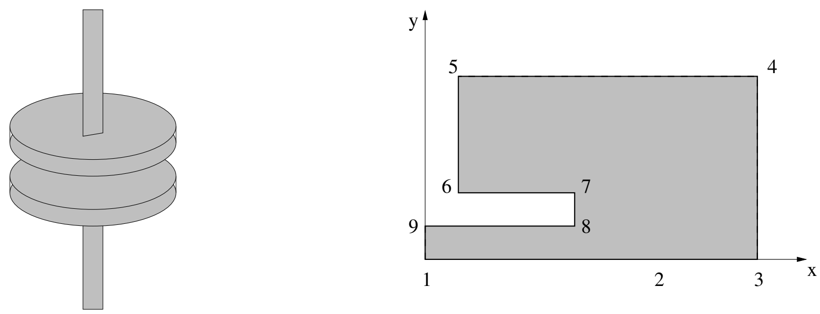


Figure 70: The capacitance and the section used for the modeling

to be solved.

$$\begin{array}{rcll} \operatorname{div}(x \operatorname{grad} u(x,y)) & = & 0 & \text{in domain} \\ u(x,0) & = & 0 & \text{along edge } y=0 \\ u(x,y) & = & 1 & \text{along edges of upper conductor} \\ \frac{\partial}{\partial n}\, u(x,y) & = & 0 & \text{on remaining boundary} \end{array} \tag{58}$$

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

$$\text{flux} = \iint_{\text{disk}} \vec{n} \cdot \operatorname{grad} u \; dA = 2\,\pi \int_0^R x\, \frac{\partial\, u}{\partial y}\, dx \approx 2\,\pi \int_0^r x\, \frac{1}{h}\, dx = \frac{\pi\, r^2}{h}\,.$$

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.

8.8.2 Create the mesh and solve the BVP

According to Figure 70 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 70

- 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 70. With the above use the program `triangle` to generate a mesh.

```
triangle -pqa capacitance.poly
```

The mesh consists of 2189 nodes, forming 4036 triangles.

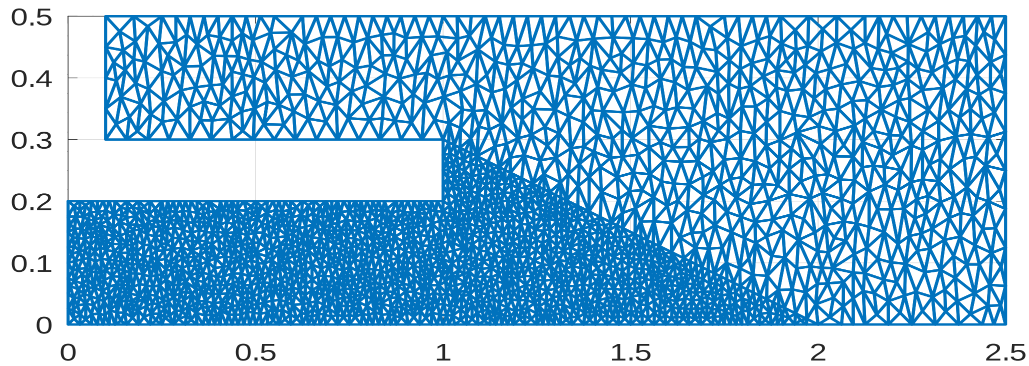


Figure 71: A mesh on the domain

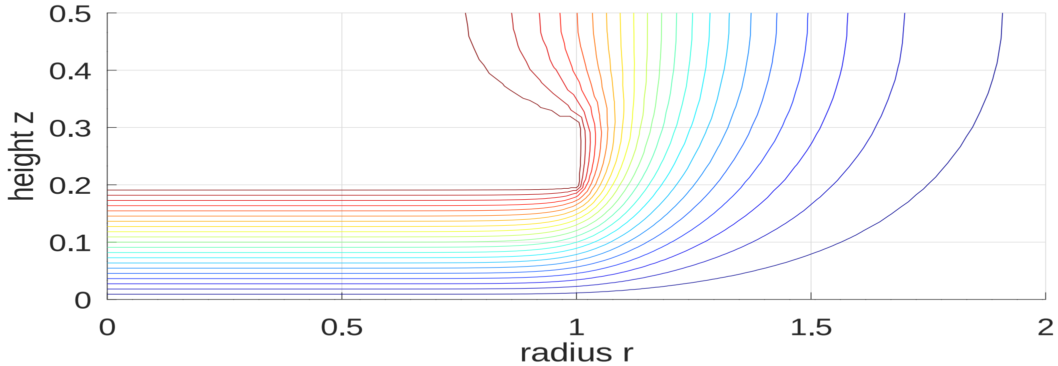
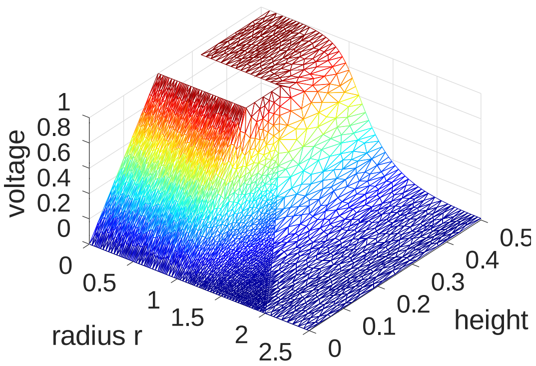


Figure 72: The contour lines of the resulting voltage

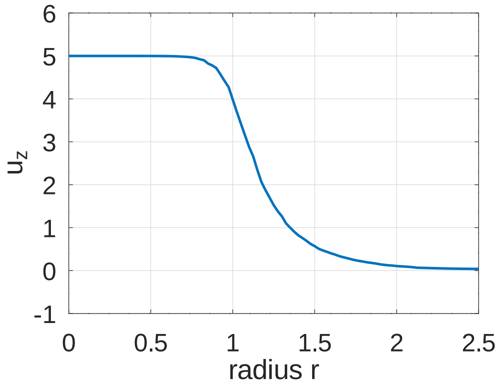
To solve the BVP (58) one needs a definition of the coefficient function and the Dirichlet boundary function. Then set up and solve the system of linear equations. This leads to a system for 1937 unknowns. Now generate a plot of the voltage $u(x, y)$ and its level curves. Find the results in Figure 73.

8.8.3 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 73. The graph confirms that between the



(a) the voltage



(b) vertical field along the horizontal axis

Figure 73: Voltage plot and electric field between the plates of the capacitance

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.

$$\text{flux} = \iint\limits_{\text{disk}} \vec{n} \cdot \operatorname{grad} u \; dA = 2\,\pi \int_0^R x \, \frac{\partial\, u}{\partial y} \, dx$$

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
```

8.9 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 more detailed in [Sout73, §12].

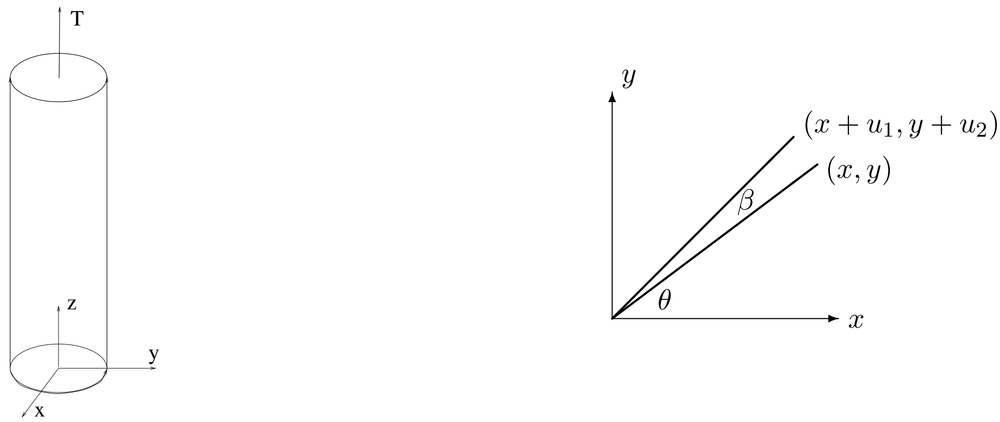


Figure 74: Torsion of a shaft

8.9.1 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 74. 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 74 and a linear approximation

$$\begin{array}{rcl} u_1(x,y) & = & r\,\cos(\beta+\theta) - r\,\cos(\theta) \approx -\beta\, r\,\sin\theta = -y\,\beta = -y\,z\,\alpha \\ u_2(x,y) & = & r\,\sin(\beta+\theta) - r\,\sin(\theta) \approx +\beta\, r\,\cos\theta = +x\,\beta = +x\,z\,\alpha \end{array} .$$

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

$$\begin{aligned} T &= \iint_\Omega x\,\tau_{yz} - y\,\tau_{yz}\;dA = \frac{E\,\alpha}{2\,(1+\nu)} \iint_\Omega x\,(x + \frac{\partial\,\phi}{\partial y}) - y\,(-y + \frac{\partial\,\phi}{\partial x})\;dA \\ &= \frac{E\,\alpha}{2\,(1+\nu)} \iint_\Omega x^2 + y^2 + x\,\frac{\partial\,\phi}{\partial y} - y\,\frac{\partial\,\phi}{\partial x}\;dA = \frac{E\,\alpha}{1+\nu}\;J\,. \end{aligned}$$

Using the **torsional rigidity** J with

$$J = \iint_\Omega x^2 + y^2 + x\,\frac{\partial\,\phi}{\partial y} - y\,\frac{\partial\,\phi}{\partial x}\;dA$$

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

$$\begin{aligned} E &= \iint_\Omega \sigma_{xz}\,\tau_{xz} + \sigma_{yz}\,\tau_{yz}\;dA = \frac{E\,\alpha^2}{4\,(1+\nu)} \iint_\Omega (-y + \frac{\partial\,\phi}{\partial x})^2 + (x + \frac{\partial\,\phi}{\partial y})^2\;dA \\ &= \frac{E\,\alpha^2}{4\,(1+\nu)} \iint_\Omega (\frac{\partial\,\phi}{\partial x})^2 + (\frac{\partial\,\phi}{\partial y})^2 - 2\,y\,\frac{\partial\,\phi}{\partial x} + 2\,x\,\frac{\partial\,\phi}{\partial y} + x^2 + y^2\;dA\,. \end{aligned}$$

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

$$\begin{array}{rcll} \operatorname{div}(\nabla\phi) = \Delta\phi &=& 0 & \text{in the cross section} \quad \Omega \\ \vec{n}\cdot\nabla\phi &=& \begin{pmatrix} y \\ -x \end{pmatrix}\cdot\vec{n} & \text{on the boundary} \quad \partial\Omega \end{array} \,. \tag{59}$$

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[22], 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

$$\begin{array}{rcll} -\Delta\chi & = & 2 & \text{in}\quad \Omega\\ \chi & = & 0 & \text{on}\quad \Gamma \end{array}\,. \tag{60}$$

The torsional rigidity is determined by

$$J = \iint\limits_{\Omega} x^2 + y^2 + x\,(-\frac{\partial\chi}{\partial x} - x) - y\,(+\frac{\partial\chi}{\partial y} + y)\;dA = -\iint\limits_{\Omega} x\,\frac{\partial\chi}{\partial x} + y\,\frac{\partial\chi}{\partial y}\;dA\,.$$

For ductile materials the von Mises stress indicates the possible fractures in the material. In this case it is given by

$$\sigma_{vM} = \sqrt{\frac{3}{2}\,(\tau_{xz}^2 + \tau_{yz}^2)} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\sqrt{(\frac{\partial\chi}{\partial x})^2 + (\frac{\partial\chi}{\partial y})^2} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\|\nabla\chi\|\,.$$

8.9.2 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

$$\tau_{xz} = +\frac{E\,\alpha}{2\,(1+\nu)}\,\frac{\partial\chi}{\partial y} = -\frac{E\,\alpha}{2\,(1+\nu)}\,y \quad \text{and} \quad \tau_{yz} = -\frac{E\,\alpha}{2\,(1+\nu)}\,\frac{\partial\chi}{\partial x} = +\frac{E\,\alpha}{2\,(1+\nu)}\,x\,.$$

The BVP (59) for the warp function ϕ is

$$\begin{array}{rcll} \operatorname{div}(\nabla\phi) = \Delta\phi & = & 0 & \text{in the cross section}\quad \Omega\\ \vec{n}\cdot\nabla\phi & = & \frac{1}{\sqrt{x^2+y^2}}\begin{pmatrix} y\\ -x\end{pmatrix}\cdot\begin{pmatrix} x\\ y\end{pmatrix} = 0 & \text{on the boundary}\quad \partial\Omega \end{array}$$

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

$$\sigma_{vM} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\sqrt{(\frac{\partial\chi}{\partial x})^2 + (\frac{\partial\chi}{\partial y})^2} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,\sqrt{x^2+y^2} = \frac{E\,\alpha}{2\,(1+\nu)}\,\sqrt{\frac{3}{2}}\,r\,.$$

[22] This restriction can be removed.

8.9.3 On a square

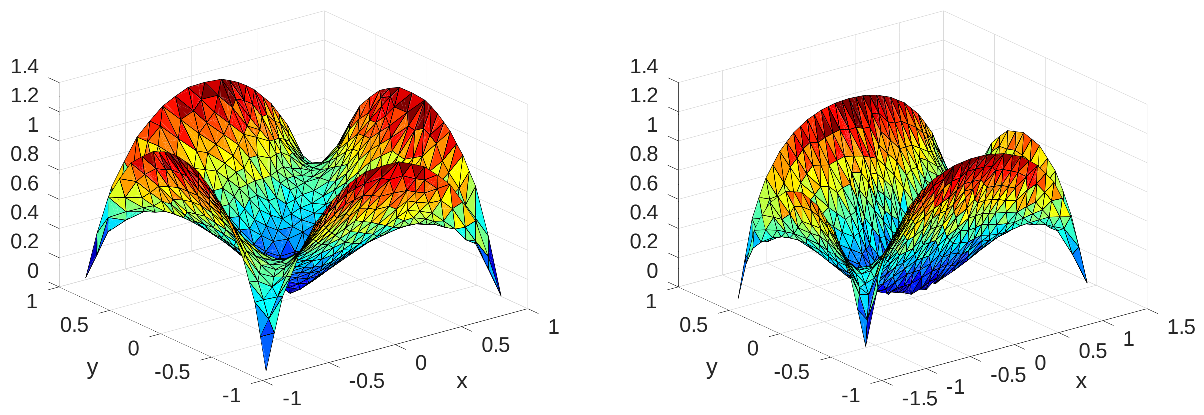
To examine the stiffness of a square cross section with a circular cross section examine a square with the same area as a circle with radius $R = 1$. Thus the length of a side is $\sqrt{\pi} \approx 1.77$. The code below solves the boundary value problem (60) and then computes the torsional rigidity by integrating

$$J = -\iint_\Omega x\,\frac{\partial\,\chi}{\partial x} + y\,\frac{\partial\,\chi}{\partial y}\;dA\,.$$

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 75(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 75: 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,'quadratic');

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);
```

```
figure(1); FEMtrisurf(Mesh,sqrt(chi_x.^2 + chi_y.^2))
           xlabel('x'); ylabel('y');
-->
J =  1.3873
```

8.9.4 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.

8.10 Dynamic heat conduction problems

The dynamic heat equation with a thermal conductivity $a(x, y)$ is of the form given in equation (4). 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

$$\begin{array}{rcll} \frac{\partial}{\partial t}\,u - \nabla\cdot(a\,\nabla u) & = & 0 & \text{for} \quad (x,y,t)\in\Omega\times(0,T] \\ u & = & g & \text{for} \quad (x,y,t)\in\Gamma_1\times(0,T] \\ \vec{n}\cdot(a\,\nabla u) & = & 0 & \text{for} \quad (x,y,t)\in\Gamma_2\times(0,T] \\ u & = & u_0 & \text{on} \quad \Omega \quad \text{at} \quad t=0 \end{array} . \tag{61}$$

In Figure 76 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.

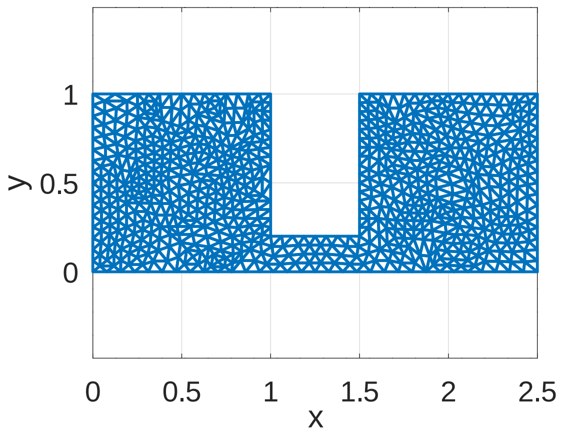


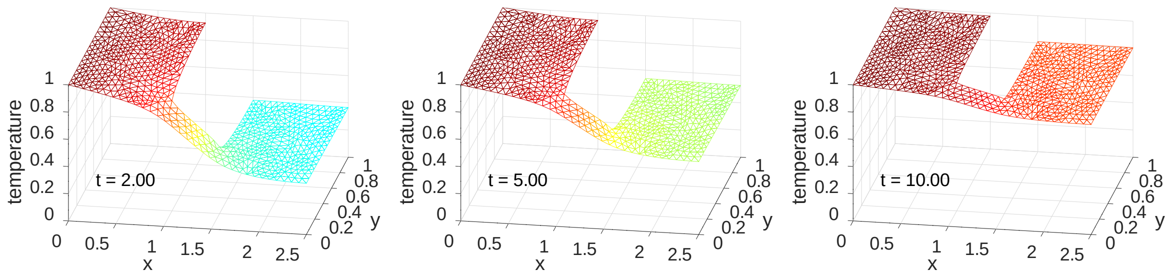
Figure 76: The mesh for a dynamic heat problem

8.10.1 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 $l = 0.5$.

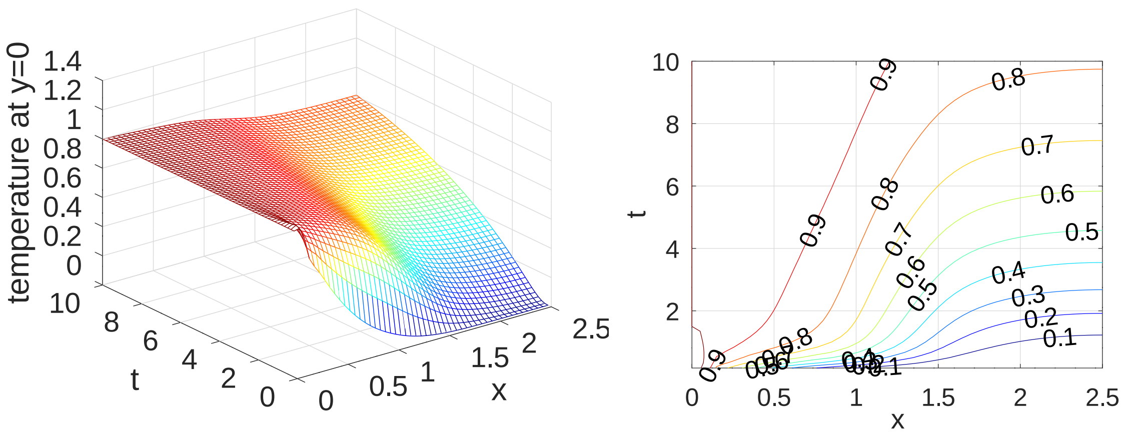
- In Figure 77 observed the delayed heating of the section on the right.

- In Figure 78 the temperature along the edge $y = 0$ for $0 \leq x \leq 2.5$ and $0 \leq t \leq 10$ is shown, as surface and contour lines.
- In Figure 79 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 77: The evolution of the temperature surface at different times



(a) surface along the edge $y = 0$ (b) contours along the edge $y = 0$

Figure 78: 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(2); FEMtrimesh(FEMmesh,u(:,end))
           xlabel('x'); ylabel('y'); zlim([0,1]); view([10 30]); caxis([0,1]);
```

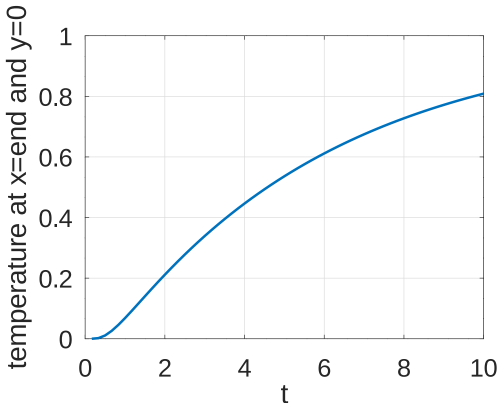


Figure 79: The temperature as function of time at the endpoint $(2.5\,,\,0)$

```
             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+1,51);  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')
```

8.10.2 With a section with lower thermal conductivity

On the modified domain visible in Figure 80 in the middle section the conductivity is considerably smaller than in the two side section, i.e.

$$a(x,y) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1 \text{ and } x \geq 1.5 \\ \frac{1}{6} & \text{for } 1 < x < 1.5 \end{cases}$$

- In Figure 81 observed the delayed heating of the section on the right.
- In Figure 82 the temperature along the edge $y = 0$ for $0 \leq x \leq 2.5$ and $0 \leq t \leq 10$ is shown, as surface and contour lines.
- In Figure 83 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.

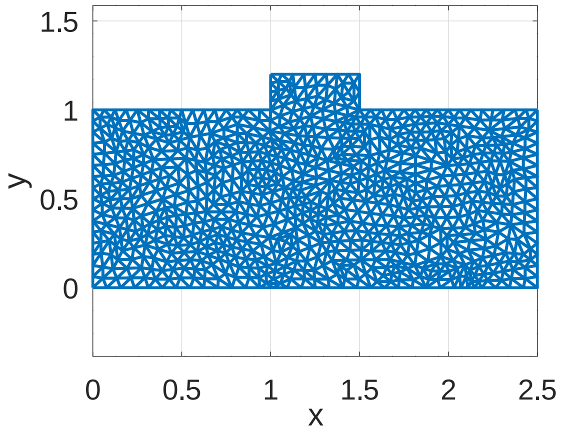
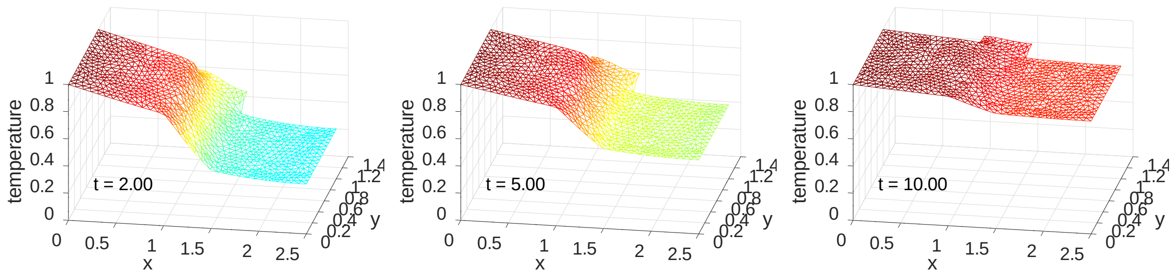
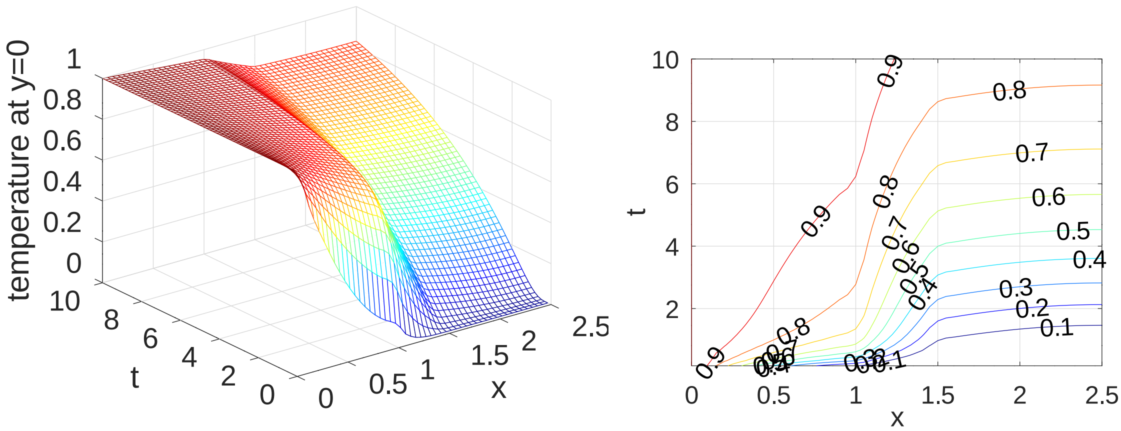


Figure 80: The mesh for a dynamic heat problem



(a) at $t = 2$ (b) at $t = 5$ (c) at $t = 10$

Figure 81: The evolution of the temperature surface at different times



(a) surface along the edge $y = 0$ (b) contours along the edge $y = 0$

Figure 82: The temperature surface at different times along $y = 0$

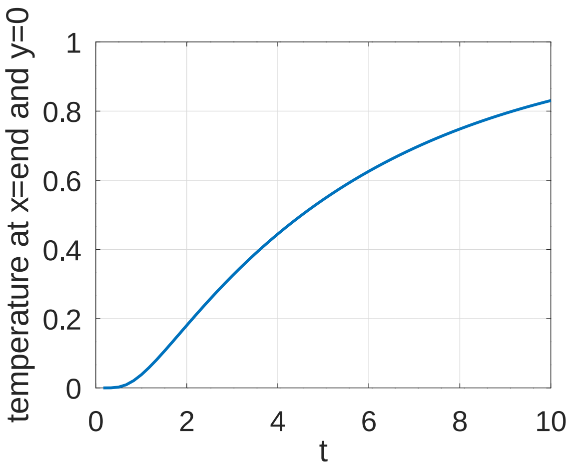


Figure 83: The temperature as function of time at the endpoint $(2.5\,,0)$

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);
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')
```

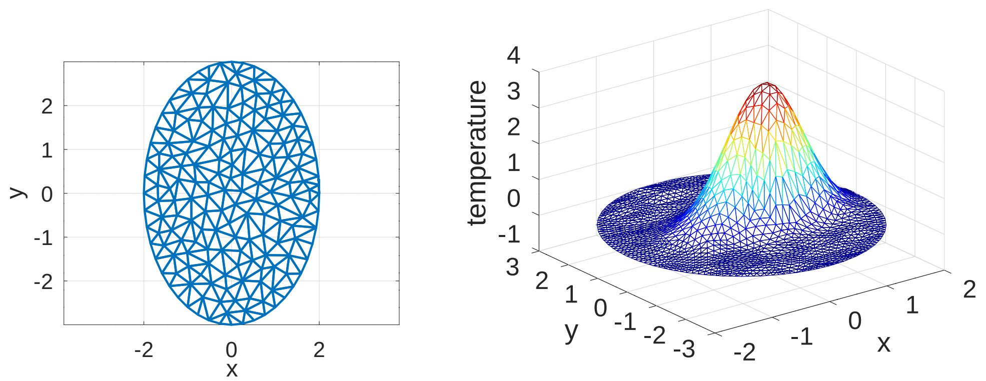
```
figure(11); [c,h] = contour(x,t(2:end),u_int,[0:0.1:1]);
            clabel(c,h);
            xlabel('x'); ylabel('t');

figure(12); plot(t(2:end),u_int(:,end))
            xlabel('t'); ylabel('temperature at x=end and y=0')
```

8.10.3 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 initial temperature profile are visible in Figure 84. The selected, nonsymmetric initial temperature is

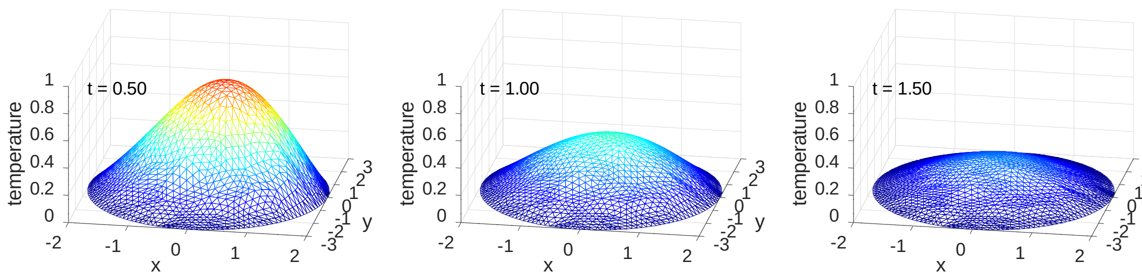
$$u_0(x,y) = \exp(-(x-0.5)^2 - 2\,y^2) \cdot (4 - x^2 - y^2) \ .$$

The initial boundary value problem is solved for times $0 \leq t \leq 2$. A few snapshots are visible in Figure 85. By looking at different time slices an animation can be generated.



(a) the elliptical domain (b) the initial temperature

Figure 84: The domain and initial temperature



(a) at $t = 0.5$ (b) at $t = 1.0$ (c) at $t = 1.5$

Figure 85: The temperature at different times

CylinderCooling.m

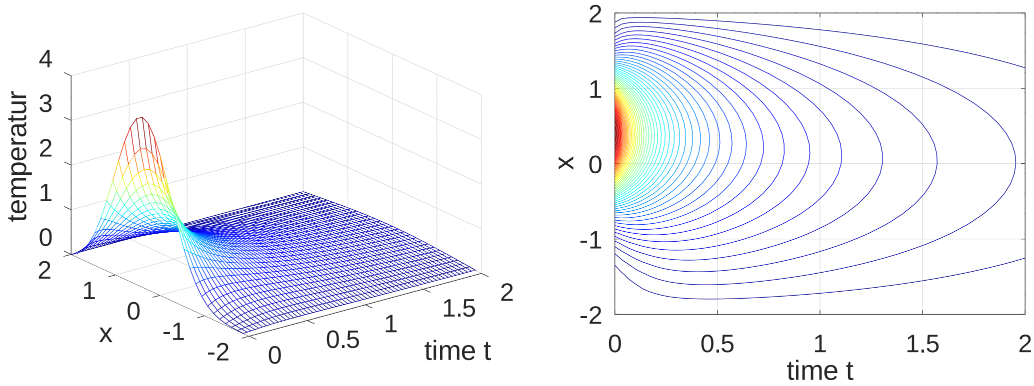
```
R = 2;  N = 50; alpha = linspace(0,2*pi*N/(N-1),N)';
Tend = 2; Nt = 60; %% number of shown time steps
FEMmesh = CreateMeshTriangle('circle',[R*cos(alpha),1.5*R*sin(alpha),...
                             -ones(size(alpha))],0.1);

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 86.

```
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 87, with linear and logarithmic scale. The exponential decay is clearly displayed in the logarithmic scale. This is consistent with the theoretical



(a) surface (b) contour lines

Figure 86: The temperature at different times along $y = 0$

result

$$u(t,x,y) = \sum_{n=1}^{\infty} c_n\, e^{-\lambda_n t}\, u_n(x,y) \tag{62}$$

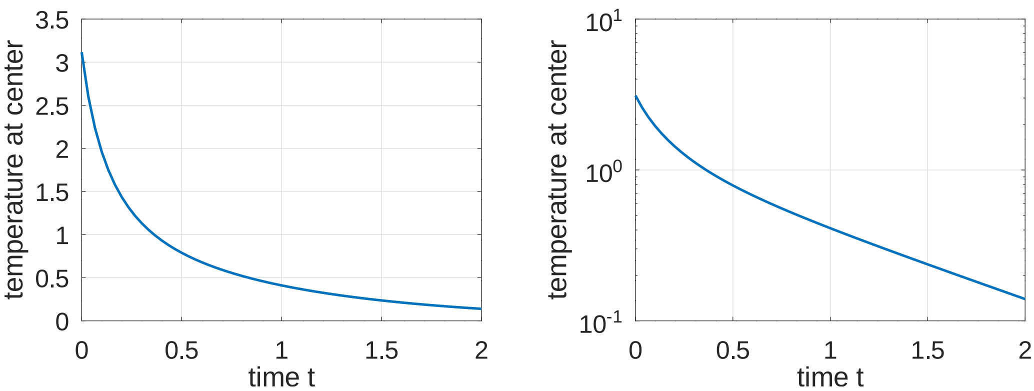
where $\lambda_1 < \lambda_2 \le \lambda_3 \le \lambda_4 \ldots$ and $u_n(x,y)$ are the eigenvalues and eigenfunctions of the boundary value problem

$$\begin{array}{rcl} -\nabla\cdot\nabla u_n &=& \lambda_n\, u \quad \text{for} \quad (x,y)\in\Omega \\ u &=& 0 \quad \text{for} \quad (x,y)\in\Gamma \end{array} .$$

For large times t in equation (62) 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 87



(a) linear scale (b) logarithmic scale

Figure 87: 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.

8.10.4 Heat waves

In Figure 88 a domain $\Omega \subset \mathbb{R}^2$ is visible. The heat equation (a special case of the IBVP (4)) to be solved is

$$\begin{array}{rcll} \frac{\partial}{\partial t}\,u(x,y,t) - \Delta\,u(x,y,t) & = & f(x,y,t) & \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\ \frac{\partial}{\partial n}\,u(x,y,t) & = & 0 & \text{for} \quad (x,y,t) \in \Gamma \times (0,T] \\ u(x,y,0) & = & 0 & \text{on} \quad \Omega \end{array} .$$

The function $f(x,y,t)$ equals $\cos(0.5\,\pi\,t)$ for $x \leq -0.9$ and zero otherwise. Thus there a periodic excitation with period 5 at the very left end of the appendix for $-1 \leq x \leq -0.9$.

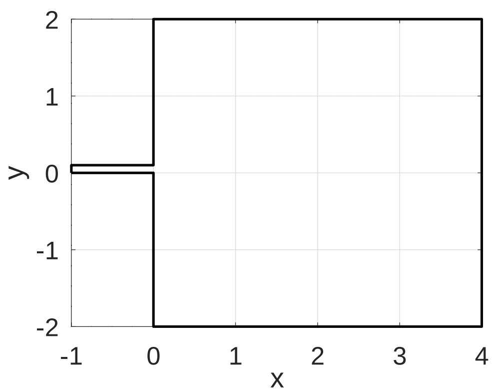
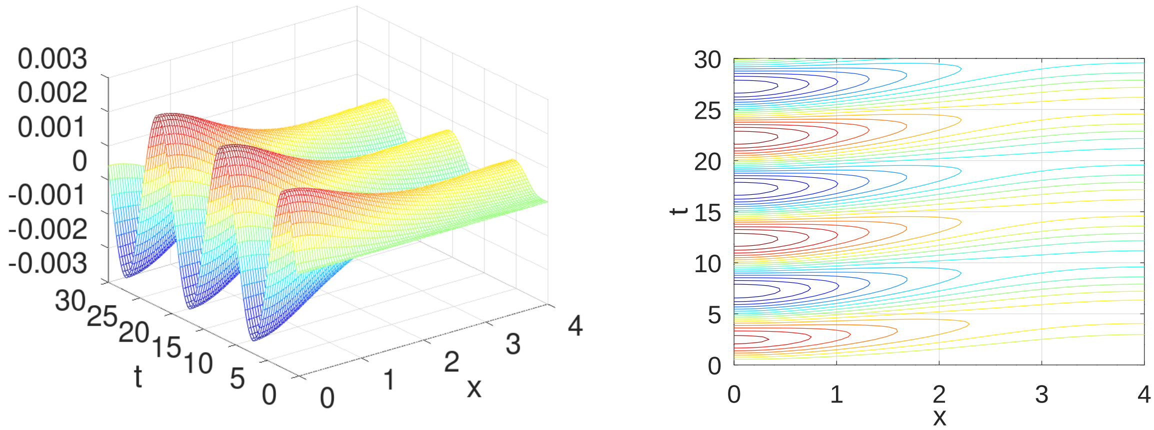


Figure 88: The domain for a heat wave propagation

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 89.

- In Figure 89(a) the periodic behavior of the temperature is clearly visible.
- In Figure 89(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 8.11, even is the setup is comparable.



(a) the surface

(b) the contour lines

Figure 89: 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');
```

8.11 Wave propagation, Kirchhoff diffraction

8.11.1 A dynamic solution

In Figure 90 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 case of the IBVP (6)) to be solved is

$$\begin{array}{rcll} \frac{\partial^2}{\partial t^2} u(x,y,t) - \Delta\, u(x,y,t) & = & f(x,y,t) & \text{for} \quad (x,y,t) \in \Omega \times (0,T] \\ \frac{\partial}{\partial n} u(x,y,t) & = & 0 & \text{for} \quad (x,y,t) \in \Gamma \times (0,T] \\ u(x,y,0) = \frac{\partial}{\partial t} u(x,y,0) & = & 0 & \text{on} \quad \Omega \end{array} . \tag{63}$$

The function $f(x,y,t)$ equals $\sin(3\,\pi\,t)$ for $x \leq -0.9$ and zero otherwise. Thus there a periodic excitation at the very left end of the appendix for $-1 \leq x \leq -0.9$. The wave speed equals 1 and the appendix (more precise: the two appendices) is a source of waves.

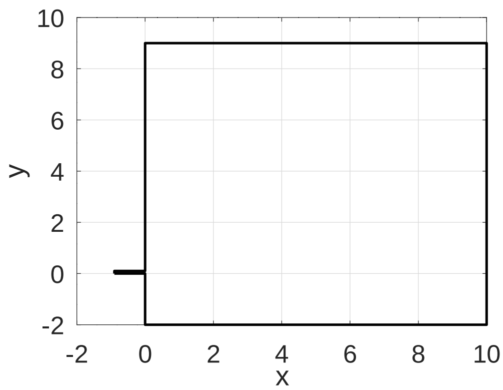


Figure 90: The domain for the wave propagation

Figure 91 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 diffracting pattern by Kirchhoff, see e.g. en.wikipedia.org/wiki/Double-slit_experiment.

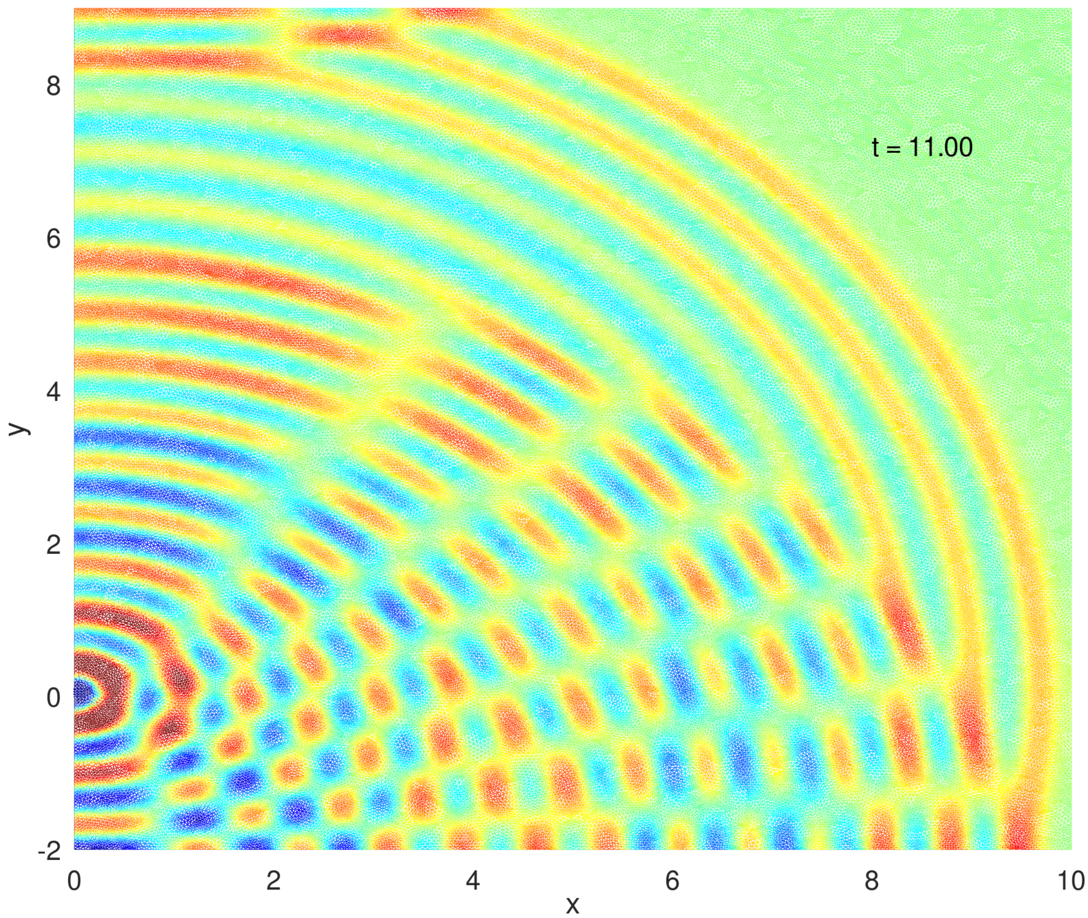


Figure 91: Wave propagation, leading to a Kirchhoff diffraction pattern

Observe that the behavior of the solution is very different from a heat equation in 8.10.4, 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 of the final snapshot.

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

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
```

8.12 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.

8.12.1 A sound wave in $\mathbb{R}^3$ with cylindrical coordinates

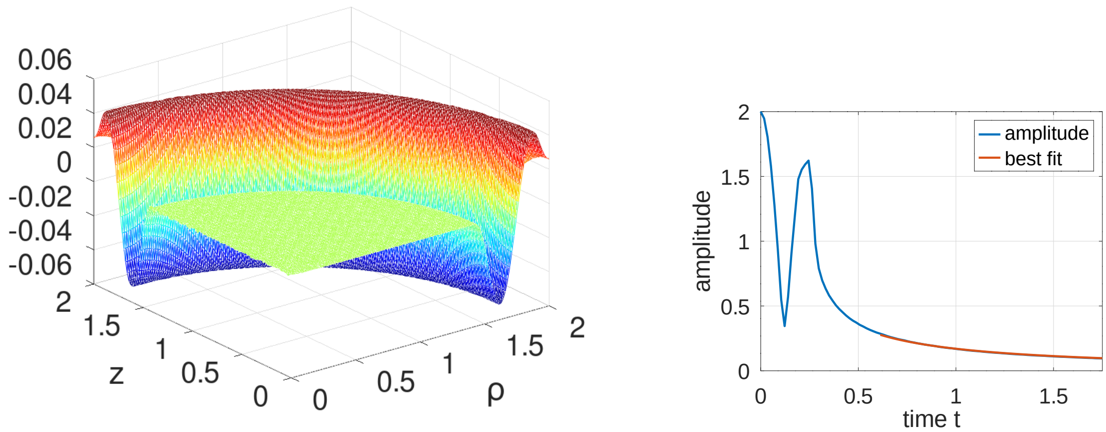
Assuming that the solution $u(\rho,z,t)$ is independent on ϕ and multiplying the wave equation by ρ arrive at

$$\rho\,\frac{\partial^2}{\partial t^2}\,u(\rho,z,t) - \frac{\partial}{\partial\rho}(\rho\,\frac{\partial u}{\partial\rho}) - \frac{\partial}{\partial z}\,(\rho\,\frac{\partial\,u}{\partial z}) = 0 \tag{64}$$

and thus it is in the from of the general hyperbolic equation (6) 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 92. Using an energy argument the amplitude of the wave front is expected to decay like $c\,\frac{1}{t}$. Using linear regression this is confirmed in Figure 92.



(a) the spherical wave (b) decay of the ampltude

Figure 92: 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')
```

8.12.2 A sound wave in $\mathbb{R}^2$

In a rectangle $0 \leq x, y \leq R$ solve the standard wave equation

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0$$

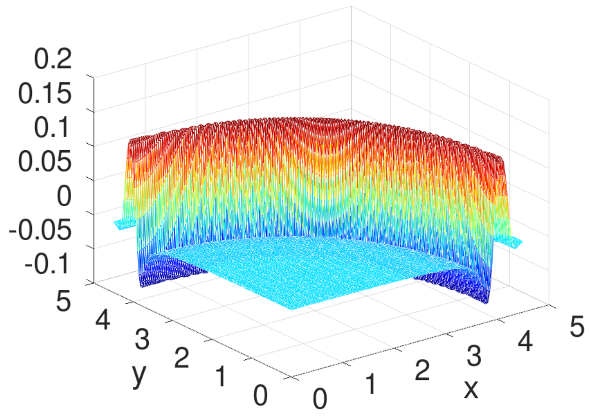
with Neumann boundary conditions $\frac{\partial u}{\partial n} = 0$, initial zero velocity and initial displacement

$$u(x,y,0) = \begin{cases} 1 + \cos(10\,r) & \text{for } 0 \leq r \leq \frac{\pi}{10} \\ 0 & \text{for } \frac{\pi}{10} \leq r \end{cases} .$$

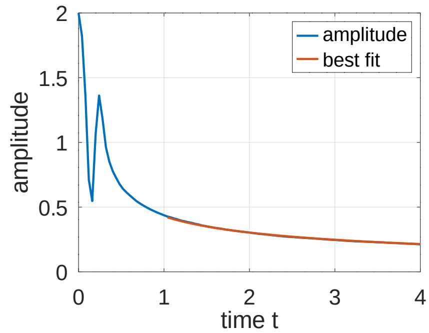
where we use $r = \sqrt{x^2+y^2}$. The result of solving this initial boundary value problem will be a circular wave moving with speed 1 and a decaying amplitude. Find the result at time $t = 4$ in Figure 93. Using an energy argument the amplitude of the wave front is expected to decay like $c\,\frac{1}{\sqrt{t}}$. Using linear regression this is confirmed in Figure 93.

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 93: 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')
```

8.13 The EIT forward problem

For a conductivity σ on a bounded domain $\Omega \subset \mathbb{R}^2$ consider the PDE

$$\nabla \cdot (\sigma \nabla u) = 0 \quad \text{in} \quad \Omega \subset \mathbb{R}^2 \tag{65}$$

- 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

$$\Lambda_\sigma \; : \; u \to \sigma \frac{\partial u}{\partial n} \quad \text{on} \quad \partial\Omega \tag{66}$$

  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

$$\mathcal{R}_\sigma \;:\; \sigma\, \frac{\partial\, u}{\partial n} \to u \quad \text{on} \quad \partial\Omega \tag{67}$$

also called current density to voltage map.

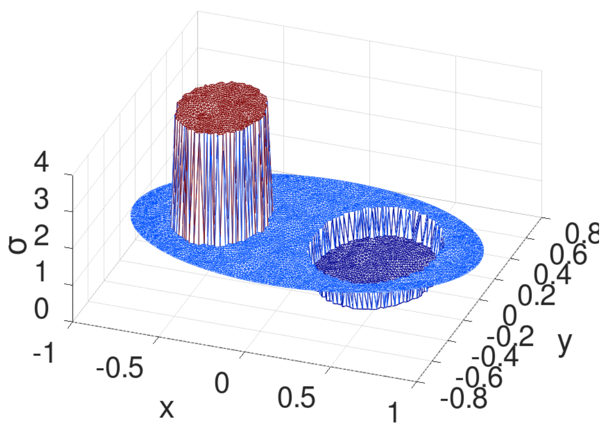


Figure 94: The conductivity with the conducting “heart” on the left and the insulating “lung” on the right

From either one of these maps it is possible to determine the conductivity σ in the domain. This is called Electrical Impedance Tomography, or short EIT. For a good, readable description 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 chest cross section) in Figure 94 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 95 and a matching current outlet 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 outlet 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

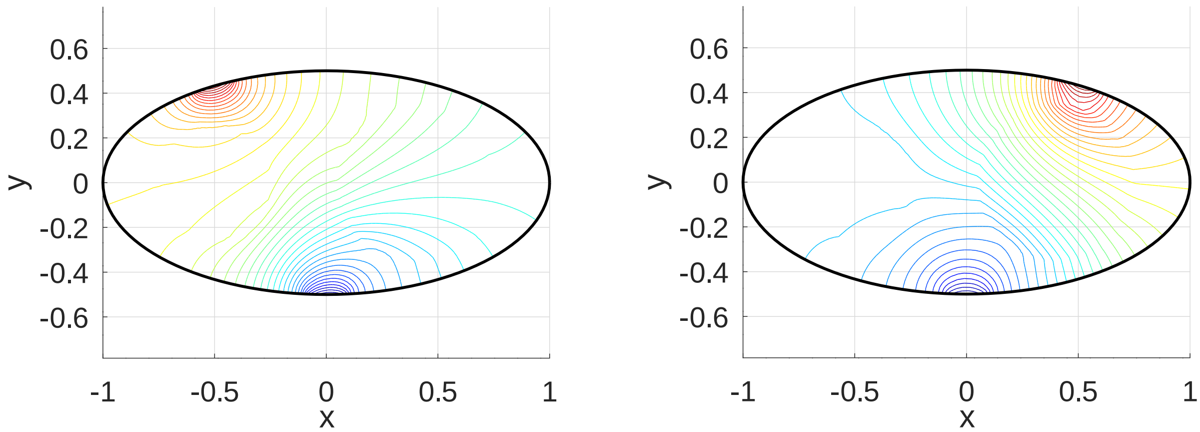
$$ds = \sqrt{R_x^2\, \sin^2\alpha + R_y^2\, \cos^2\alpha}\; d\alpha\;.$$

The "heart" is given by

$$(x+0.5)^2 + y^2 \leq 0.25^2 \quad \text{with conductivity} \quad \sigma = 4$$

and the "lung" is given by

$$(x-0.4)^2 + y^2 \leq 0.35^2 \quad \text{with conductivity} \quad \sigma = \frac{1}{4}\,.$$



(a) at angle 120°

(b) at angle 60°

Figure 95: Contours of the voltages

FEMoctave is used twice to determine the voltage u in the domain, leading to the level curves in Figure 95. 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

$$u_{mean} = \frac{1}{\text{area}(\Omega)} \iint_\Omega u \; dA = 0$$

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

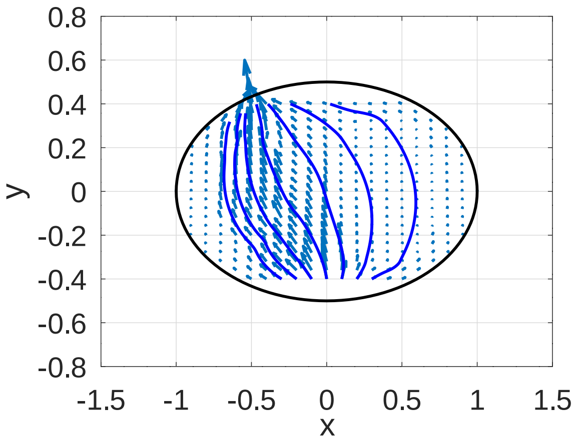
```
function u = BVP2DsymMean(Mesh,a,b0,f,gD,gN1,gN2)
  if nargin ~= 7
    print_usage();
  endif
  switch Mesh.type
    case 'linear'  %% first order elements
      [A,b] = FEMEquation (Mesh,a,b0,0,0,f,gD,gN1,gN2); % compute with compiled code
```

```
    case 'quadratic' %% second order elements
      [A,b] = FEMEquationQuad(Mesh,a,b0,0,0,f,gD,gN1,gN2);
    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
```

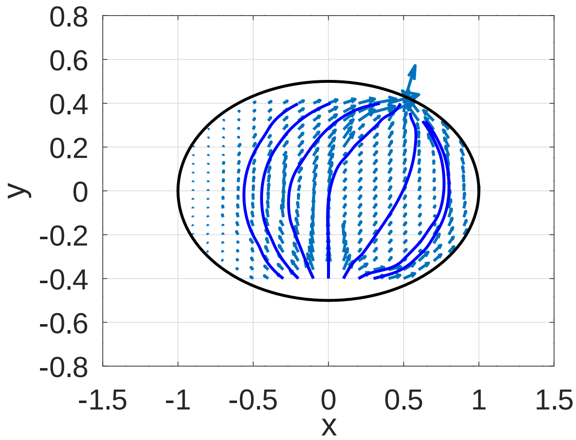
Using the current density $\vec{J} = -\sigma\,\nabla u$ the vector fields in Figure 96 can be determined. With `FEMgriddata()` determine ∇u and then multiply by the conductivity σ to obtain the current density $\vec{J}$. Using the same starting points along $y = -0.4$ a few streamlines are shown.

- In Figure 96(a) the current takes the path of least resistance and is attracted by the highly conducting "heart".
- In Figure 96(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 96 would be perfectly symmetric.



(a) at angle 120°



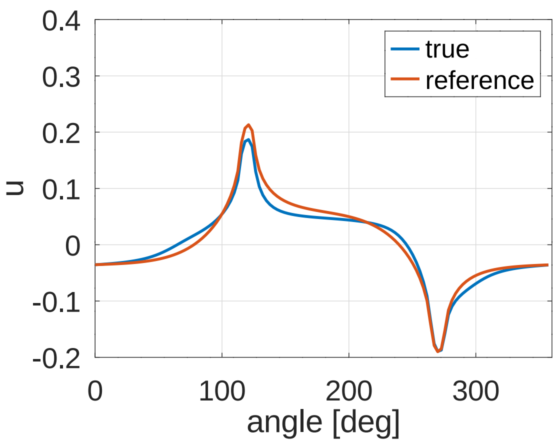
(b) at angle 60°

Figure 96: The vector field for the current density $\vec{J}$ and a few streamlines

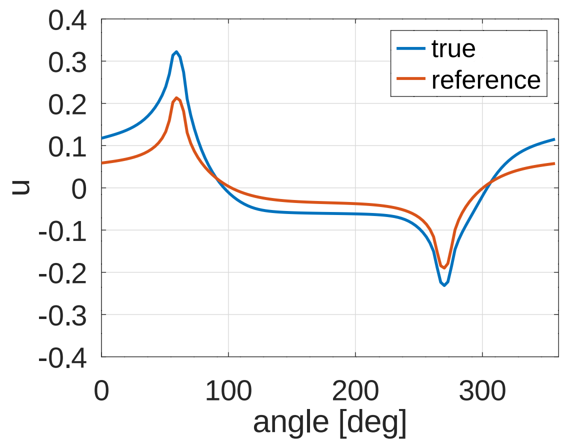
As a reference the situation of constant $\sigma = 1$ is computed too and the resulting voltages on the boundary are shown in Figure 97. The deviations from the reference on the boundary Γ contain information about the conductivity inside of the domain Ω. The deviations from the reference are shown in Figure 98. 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));
```

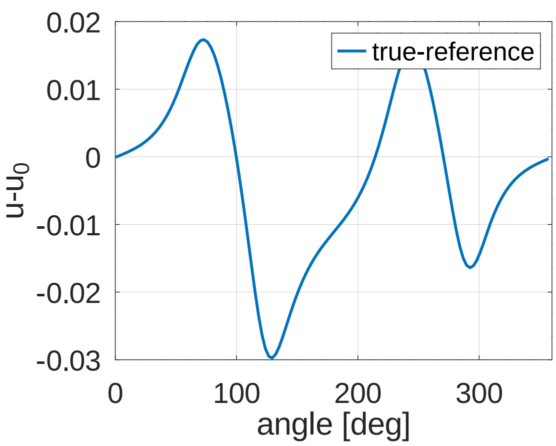


(a) at angle 120°

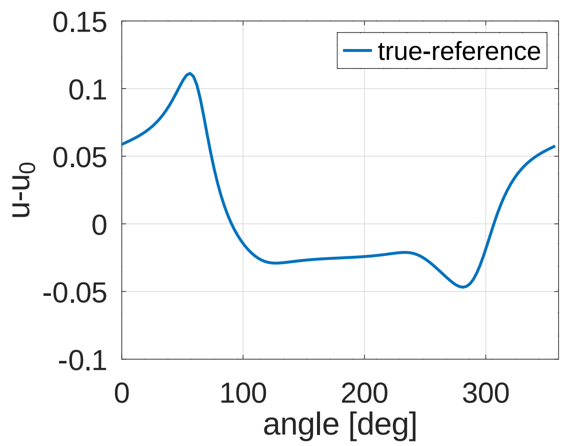


(b) at angle 60°

Figure 97: Voltage along the boundary



(a) at angle 120°



(b) at angle 60°

Figure 98: Differences of the voltage and the reference voltage

```
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

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');  %% add the boundary
           hold off
%% 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

$$\vec{n} = \frac{1}{\sqrt{R_x^2 \sin^2\alpha + R_y^2 \cos^2\alpha}} \begin{pmatrix} R_y\, \cos\alpha \\ R_x\, \sin\alpha \end{pmatrix}$$

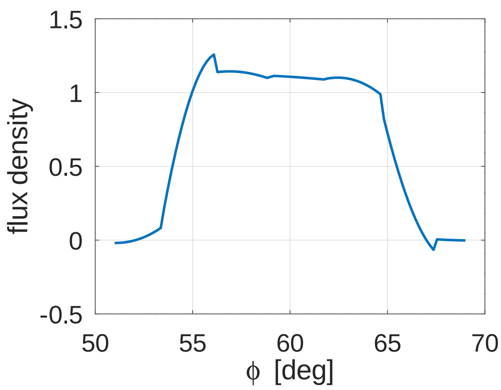
and then integrate over the segements where the flux is not zero

$$\int_{\text{section}} \langle \vec{n}, \nabla u\rangle\, ds\ .$$

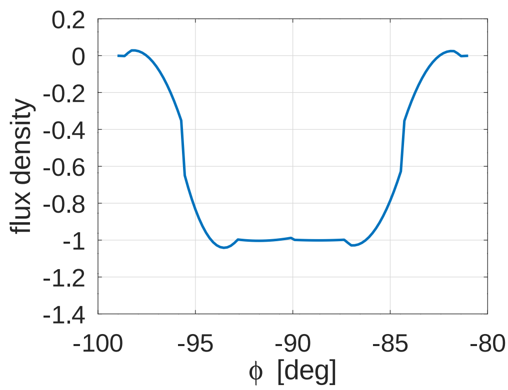
To evaluate this numerically use `FEMgriddata()` to determine the values of the gradient $(\frac{\partial\, u}{\partial x}, \frac{\partial\, u}{\partial y})$ and then `trapz()` to perform a numerical integration. Observe that along the boundary the length segment is given by

$$ds = \sqrt{R_x^2 \sin^2\alpha + R_y^2 \cos^2\alpha}\; d\phi\ .$$

The code below leads to in inlet flux of ≈ 0.1975 and to outlet fluxes at either ≈ 0.1980 at 60° or ≈ 0.1964 at 120°.



(a) outlet at angle $+60^\circ$



(b) inlet at angle -90°

Figure 99: 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)
```

8.14 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.

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 100.
- a pressure of $P = 10$ atm $= 10^6$ Pa $= 10^6\ \frac{\text{N}}{\text{m}^2}$.
- with a copper pipe, i.e. a yield strength of ≈ 33 MPa or a steel pipe, i.e. a yield strength of ≈ 350 MPa.

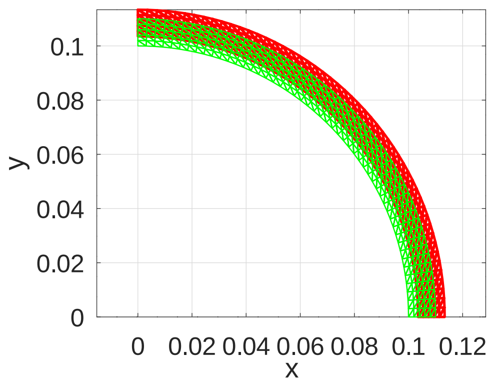


Figure 100: One quarter of a section through the pipe, the original domain (green) and the deformed domain (red)

8.14.1 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 $R \leq r \leq R + \Delta R$ and $0 \leq \phi \leq \frac{\pi}{2}$. Use `CreateMeshRect()` to create a rectangular mesh and then `MeshDeform()` to create the domain in Figure 100.
- 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 5 on page 37 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 the 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
```

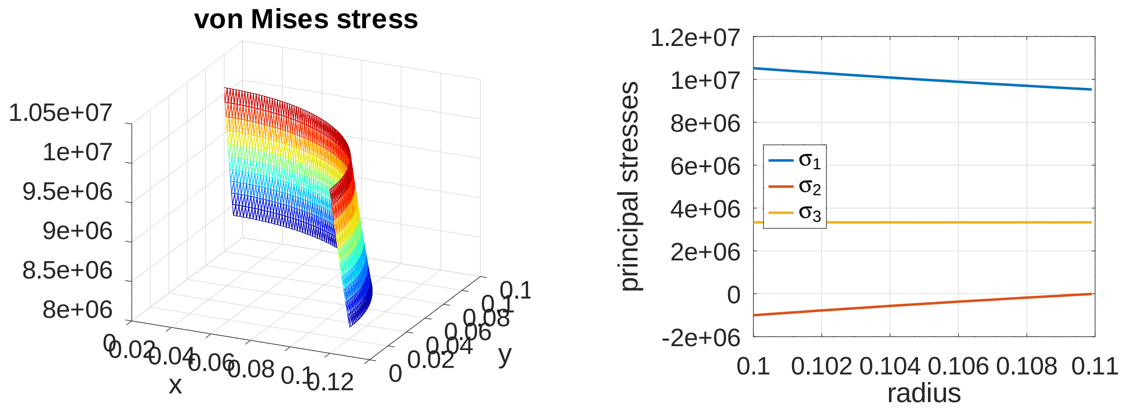
```
  angle = atan2(xy(:,2),xy(:,1));  res   = P*sin(angle);
endfunction

[u1,u2] = PlaneStrain(FEMmesh,E,nu,{0,0},{0,0},{'gN1','gN2'});

factor = 400;
figure(111); 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);
                     hold off; axis equal; xlabel('x'); ylabel('y');
```

The last few lines in the above code generate the domain visible in Figure 100.

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 101(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 101: 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. $\alpha = \frac{\pi}{4} = 90^\circ$, 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 101(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 100 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.

8.14.2 As an axisymmetric problem

The above problem can be examined as an axially symmetric problem. The domain is given by $R \le r \le R+dR$ 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 $x = r = R + dR$ 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 script functions `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')
```

8.14.3 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 $u(r) = u_r(r)$ the expression to be minimized is

$$\begin{aligned}
U_r(u) &= \int_R^{R+\Delta R} \frac{r\,k}{2}\left((1-\nu)\,((u'(r))^2 + \frac{1}{r^2}\,u^2(r)) + \frac{2\,\nu}{r}\,u(r)\,u'(r)\right)\,dr - R\,P\,u(R)\\
&= \int_R^{R+\Delta R} \frac{k}{2}\left((1-\nu)\,(r\,(u'(r))^2 + \frac{1}{r}\,u^2(r))\right)\,dr + \frac{k\,\nu}{2}\,u^2(r)\Big|_{r=R}^{R+\Delta R} - R\,P\,u(R)\\
U_r(u+\phi) &= U_r(u) + \int_R^{R+\Delta R} k\,(1-\nu)\,(r\,u'\phi' + \frac{1}{r}\,u\,\phi)\,dr + k\,\nu\,u(r)\,\phi(r)\Big|_{r=R}^{R+\Delta R} - R\,P\,\phi(R) + O(\phi^2)\\
&= U_r(u) + \int_R^{R+\Delta R} k\,(1-\nu)\,(-(r\,u')' + \frac{1}{r}\,u)\,\phi\,dr +\\
&\quad +k\,\left((1-\nu)\,r\,u'(r)\,\phi(r) + \nu\,u(r)\,\phi(r)\right)\Big|_{r=R}^{R+\Delta R} - R\,P\,\phi(R) + O(\phi^2)\;.
\end{aligned}$$

Use the Euler–Lagrange equation of 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 4.9.2 (page 63) all stresses and strain 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;
```

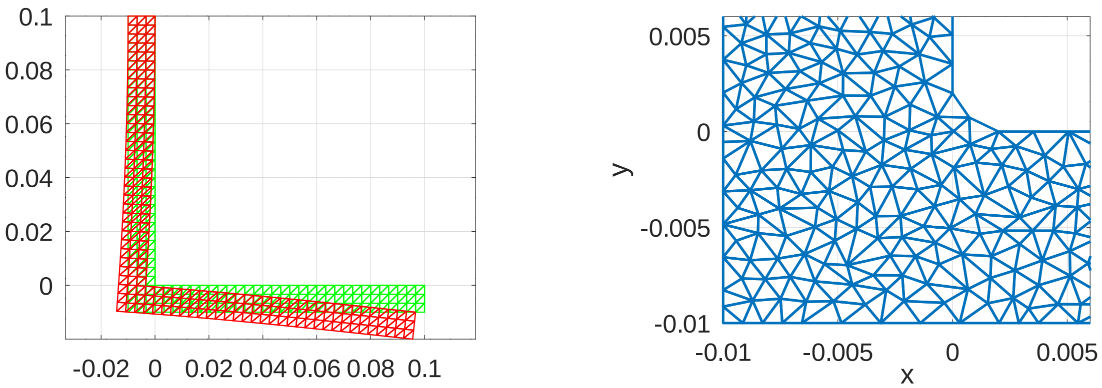
8.15 A crook with a weight attached

Examine the two L–shaped steel beams in Figure 102(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 102(b). 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()`. Since second order elements do not suffer from 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 102: Original and deformed domain for the hook with attached weight at the right edge

Then find the approximate displacements u_1 and u_2 by calling `PlaneStress()`. The code segement belows estimates the maximal vertical displacement by $-8.96 \cdot 10^{-4}$ m, i.e. approximately 0.9 mm. To verify the order of magnitude one may use two 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 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
$$\frac{\partial^2\,u_1(y)}{\partial y^2} = \frac{-F\,(H + W/2)}{E\,I} = \frac{-F\,(H + W/2)}{E\,\frac{1}{12}\,W^3\,W} \approx 3.1504 \cdot 10^{-2}\,.$$

Then use the contitions $u_1(1) = \frac{\partial\,u_1(1)}{\partial y} = 0$ at the top edge to estimate $\frac{\partial\,u_1(-W/2)}{\partial y} \approx 6.613 \cdot 10^{-3}$, 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 105(a).

To generate Figure 102(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. Is 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 103(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.

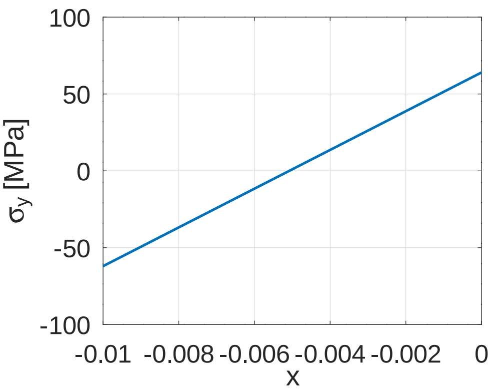
- At $x = \frac{H}{2} = 0.05$ examine the normal stress σ_x in x–direction along a vertical slice. The result in Figure 103(b) shows a compression in the lower segment and traction in the upper segment. This corresponds to the downward bending on the horizontal arm. For the moment with respect to the point $(\frac{H}{2}, 0)$ we expect

$$W \int_{-W}^{0} y\, \sigma_x(0.05, y)\; dy \approx \frac{1}{2}\,\text{H} \cdot \text{Force} = 5\;.$$

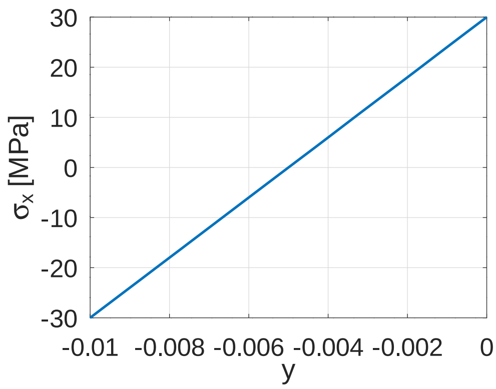
  The values of the normal stress σ_y are approximately zero. By integrating the shearing stress τ_{xy} obtain again the applied force, i.e.

$$W \int_{-W}^{0} \tau_{xy}(0.05, y)\; dy \approx \text{Force} = -100\;.$$

- Observe that the stress values in the vertical arm are considerably larger than in the horizontal arm. The values in Figure 103(b) are at $x = 0.05$. For larger values of x the strains σ_x will be even smaller.



(a) a horizontal slice at $y = 0.05$

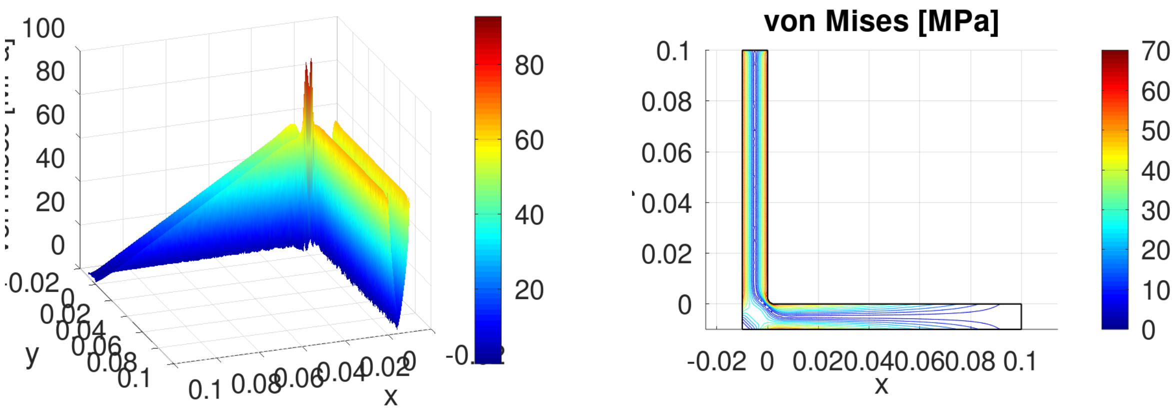


(b) a vertical slice at $x = 0.05$

Figure 103: 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
```

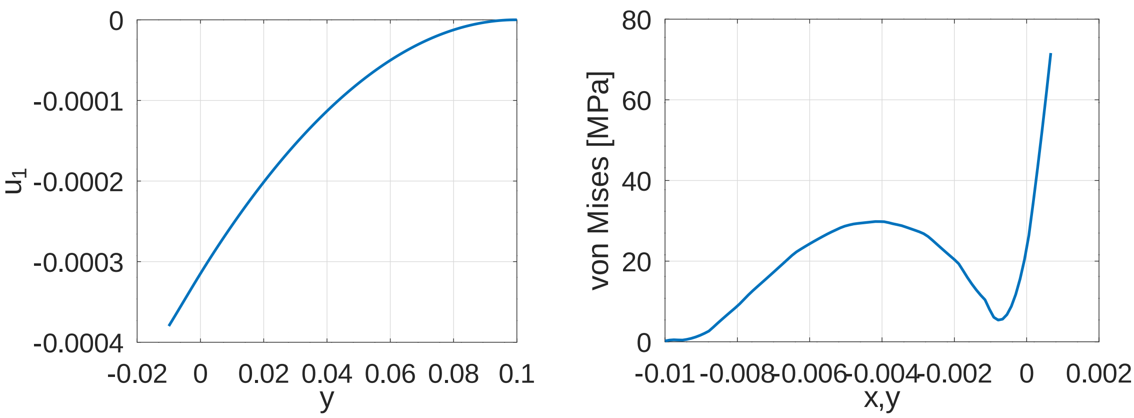
Since steel is a ductile material one can use the von Mises stress to decide whether the crook will withstand the force of 100 N. Use `EvaluateVonMises()` to find the values of the von Mises stress at the nodes and then `FEMtrisurf()` and `FEMtricontour()` to generate Figure 104. The contour lines in Figure 104 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 102(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 105(b). Since the yield strength of steel is ≈ 330 MPa the crook should be able to support the force of 100 N .



(a) the surface, in MPa

(b) the contour lines

Figure 104: The von Mises stress on the crook, as surface and level curves



(a) bending in vertical beam (b) von Mises stress at corner

Figure 105: 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]')
```

8.16 A wrench

A classic example application for mechanical FEM is a wrench. With the image of a typical wrench use the tool `xinput()` 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 106(a). Then setup an appropriate configuration of the wrench.

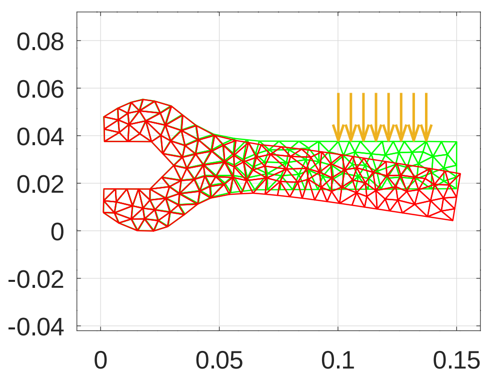
- The material is steel with the parameters $E = 200$ GPa and $\nu = 0.25$.
- Most of the boundary is force free, thus with the code -22 according to Table 5 on page 37.
- Along the two horizontal sections on the very left the displacements are zero, modeling the screw head in the wrench. A closer look 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 $\frac{100}{0.05 \cdot 0.005} = 4 \cdot 10^5\ \frac{\text{N}}{\text{m}^2}$. 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.

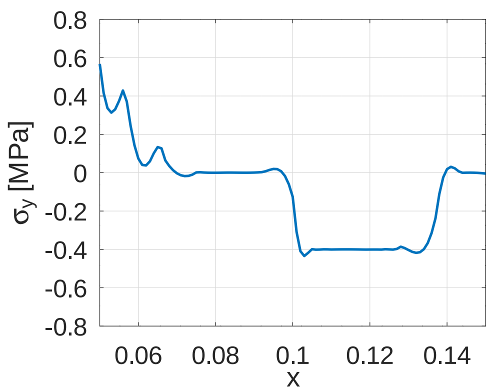
With this data the mesh is generated by calling `CreateMeshTriangle()`. Then the mesh of linear elements should be upgraded to quadratic or cubic elements with the help of `MeshUpgrade()`. Then use `PlaneStress()` to solve for the displacments u_1 and u_2.

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
```



(a) the wrench, original and deformed



(b) σ_y along upper edge

Figure 106: The deformed wrench and the stress σ_y along upper edge with the applied load

With the solution the original and deformed shape can be displayed with the applied load visualized by a few vectors, see Figure 106(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])
```

Then 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 106(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 107(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 107(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
```

8.17 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 the 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.

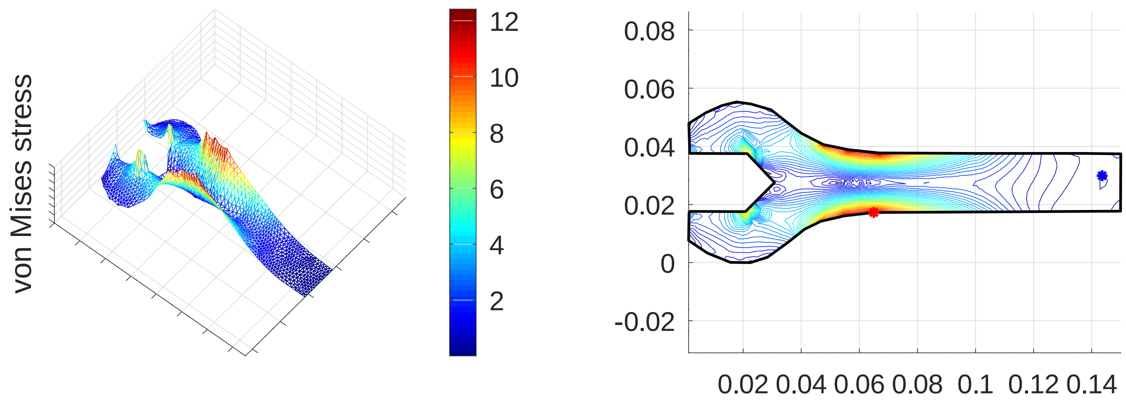


Figure 107: Surface and contour plot of the von Mises stress in [MPa]

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()`.

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});
```

Then display the original and the deformed domain in Figure 108 and the displacements in Figure 109.

RubberBox.m

```
factor = 1;
figure(10); trimesh(Mesh.elem,Mesh.nodes(:,1)+factor*ur,...
                    Mesh.nodes(:,2)+factor*uz,'color','red','linewidth',2);
hold on;trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),'color','green','linewidth',1);
           hold off; 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')
```

As last step evaluate the stresses and then the von Mises stress, leading to the surface and contour plots in Figure 110.

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)
```

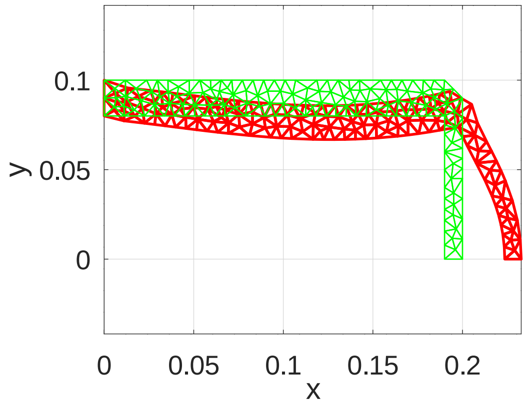
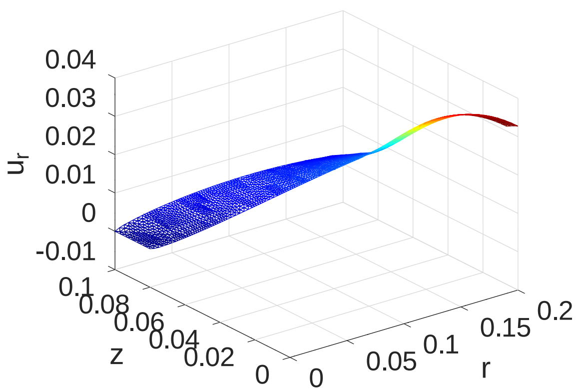
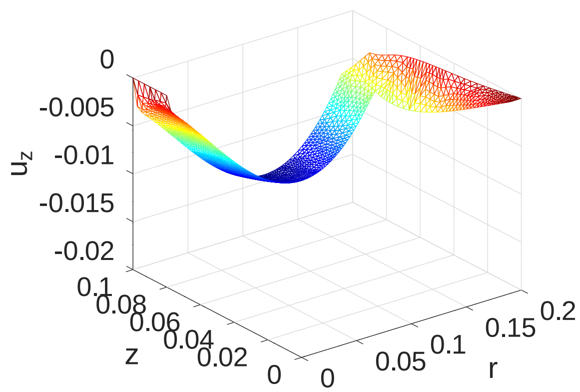


Figure 108: The upper half of the original and deformed domain for the rotating rubber box



(a) radial displacement u_r



(b) z–displacement u_z

Figure 109: The displacements u_r and u_z for the rotating rubber box

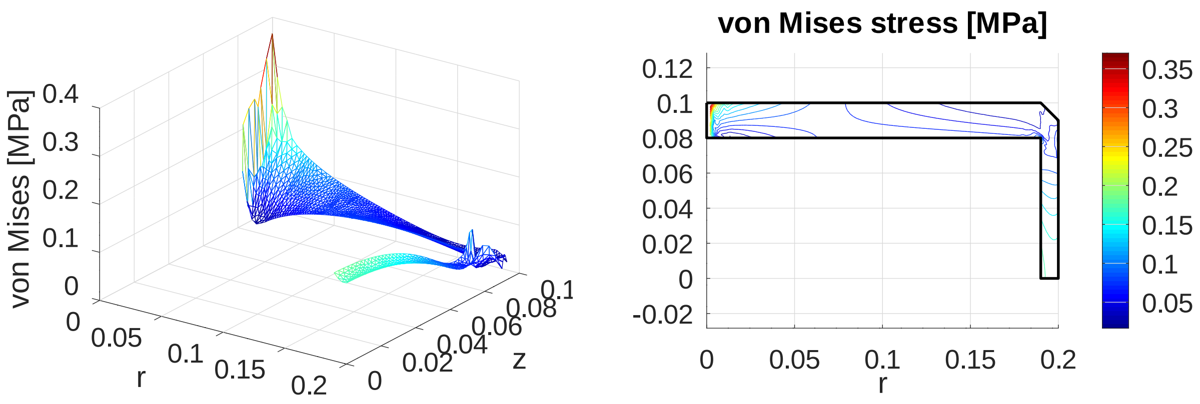


Figure 110: The von Mises stress for the rotating rubber box

```
              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; %%shading interp
              title('von Mises stress [MPa]')
```

8.18 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.

8.18.1 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 111. 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 $\mathbb{R}^3$.

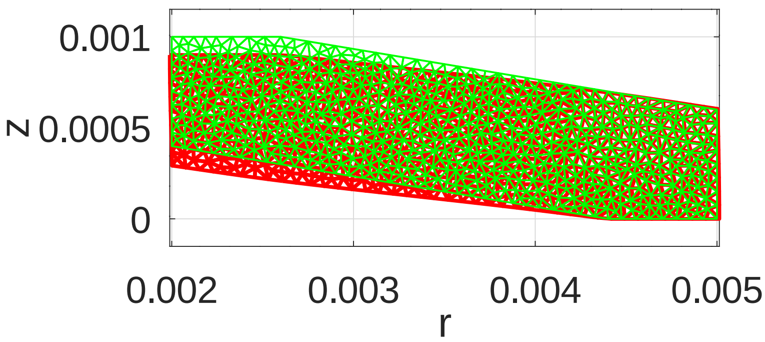


Figure 111: The original and deformed domain

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.

WasherSpring.m
```
pkg load femoctave
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});
factor = 1;
figure(10);clf; trimesh(Mesh.elem,Mesh.nodes(:,1)+factor*ur,...
           Mesh.nodes(:,2)+factor*uz,'color','red','linewidth',2);
           hold on; trimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2),...
           'color','green','linewidth',1);
           hold off; axis equal; xlabel('r'); ylabel('z');
           xticks([2:5]/1000); yticks([0:0.5:1]/1000)
```

Display the radial displacement u_r in Figure 112 and the height displacement u_z in Figure 113.

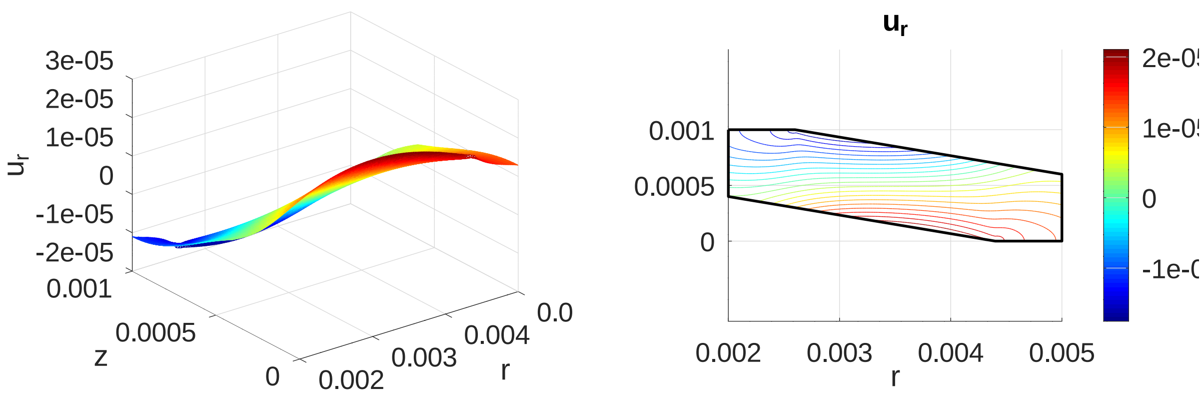


Figure 112: The radial displacement u_r

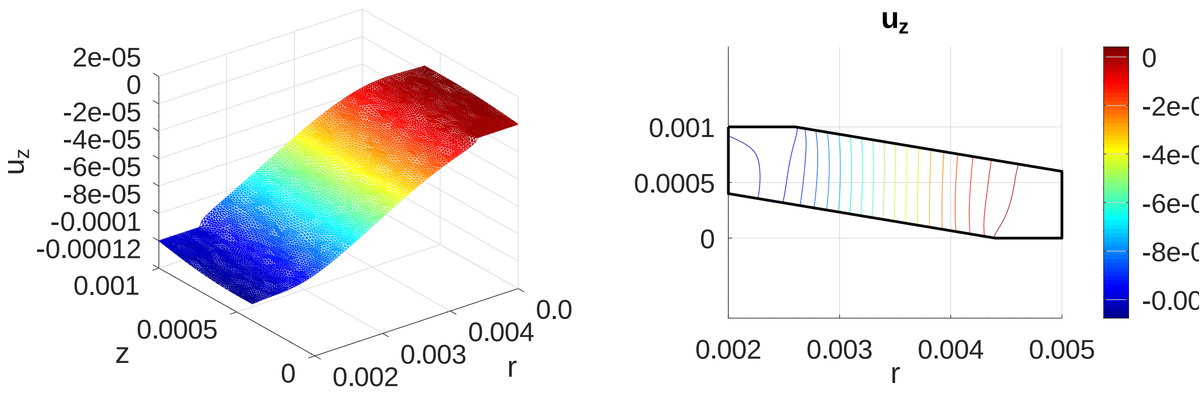


Figure 113: The height displacement u_z

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)
```

8.18.2 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)
```

At any height $0 \leq h \leq 0.001$ examine the slice $a \leq r \leq b$ in the domain visible in Figure 111 and perform an integration to determine F.

$$F \;=\; 2\,\pi \int_{r=a}^{b} r\,\sigma_z(r,h)\;dr$$

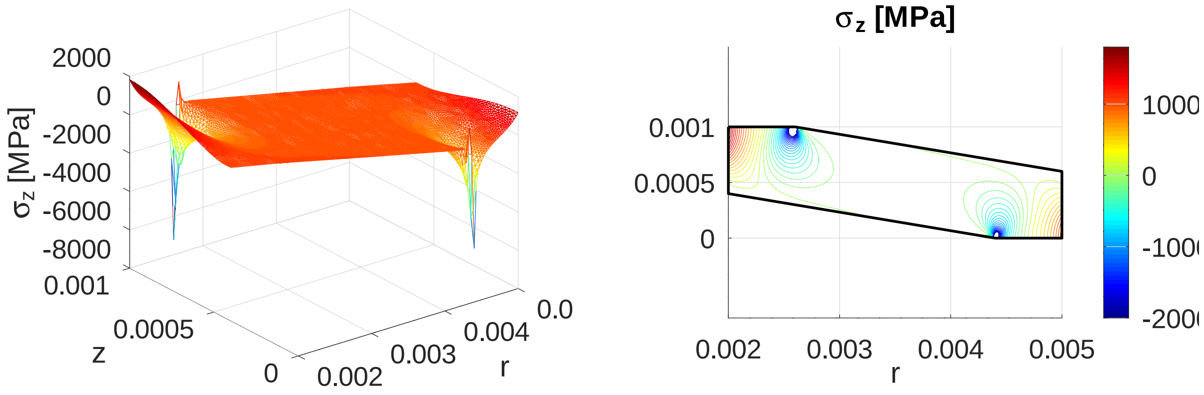


Figure 114: The normal stress σ_z

Examine the graph of the normal stress σ_z in Figure 114 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[23]

$$\begin{aligned} F_{up} &\approx 1415.1 \text{ N} \\ F_{middle} &\approx 2401.6 \text{ N} \\ F_{low} &\approx 1085.8 \text{ N} \end{aligned}$$

and the graphs in Figure 115. 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.

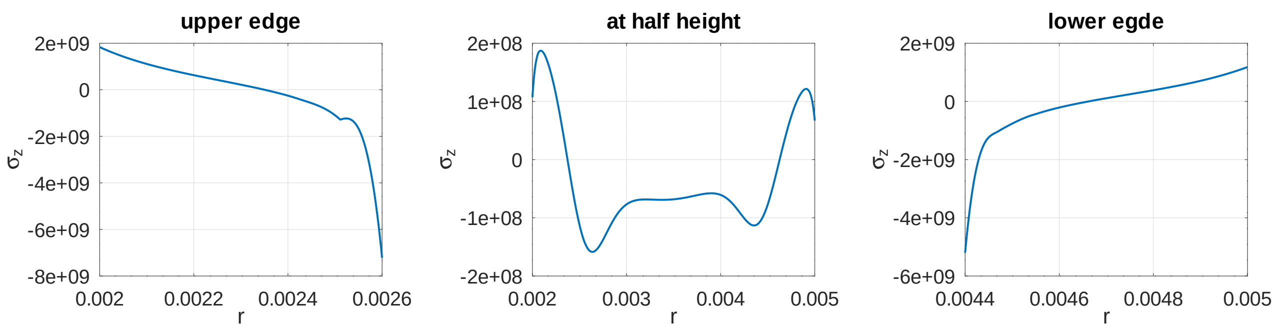


Figure 115: The normal pressures σ_z along upper and lower edge and at half height

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')
```

[23] A computation with Comsol Multiphysics lead 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 115 is confirmed.

```
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)
```

8.18.3 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 $0 \leq s \leq d$ obtain $F(s) = k\,s = \frac{s}{d}\,F(d) = \frac{s}{d}\,F$. An elementary integration leads to the energy U in the system.

$$U = \int_0^d F(s)\;ds \;=\; \int_0^d \frac{s}{d}\,F\;ds \;=\; \frac{1}{2}\,d\,F$$

Thus find the force $F = \frac{2\,U}{d}$. Using the results in Section 7.6 (starting on page 146) the elastic energy U is given by

$$\begin{aligned} U(\vec u) &= 2\,\pi \iint\limits_{\Omega} \frac{r\,E}{2\,(1+\nu)\,(1-2\,\nu)} \left((1-\nu)\,((\frac{\partial\,u_r}{\partial r})^2 + (\frac{\partial\,u_z}{\partial z})^2 + \frac{1}{r^2}\,u_r^2) + \right. \\ &\qquad \left. +2\,\nu\,((\frac{\partial\,u_r}{\partial r})(\frac{\partial\,u_z}{\partial z}) + \frac{1}{r}\,u_r\,(\frac{\partial\,u_r}{\partial r} + \frac{\partial\,u_z}{\partial z}))\right)\,dA + \\ &\quad +2\,\pi \iint\limits_{\Omega} \frac{r\,E}{1+\nu}\,\frac{1}{4}\,(\frac{\partial\,u_r}{\partial z} + \frac{\partial\,u_z}{\partial r})^2\;dA\,. \end{aligned}$$

Since the displacements u_r and u_z are available use the function `FEMIntegrate()`. It is best to evaluate the displacements and their partial derivatives at the Gauss nodes 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
```

8.18.4 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 triangle, but linear or quadratic functions. Find the results in Table 14.

- 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 $F = \frac{2U}{d}$.
- 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 1. The force F evaluated with elements of order 3 will be the smallest. This is confirmed in Table 14.
- Using finer meshes will lead to smaller minimal energies U and thus smaller forces F.
- The estimates in Section 5.4 on page 81 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 14.

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 14: Comparison of different elements for the washer fastener example

8.18.5 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.
- The third setup prevents both sides to move in radial direction.

This is implemented by switches at in the first section of the code. for `Contour` are 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

$$\begin{aligned} F &= 2402.1\ \text{N} && \text{with both sides free} \\ F &= 2880.4\ \text{N} && \text{with inner side free and outer side fixed} \\ F &= 3809.1\ \text{N} && \text{with both sides fixed} \end{aligned}$$

As expected the additional constraints lead to a stiffer system.

Bibliography

[AxelBark84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Values Problems*. Academic Press, 1984.

[Demm97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[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.

[www:triangle] J. R. Shewchuk. https://www.cs.cmu.edu/~quake/triangle.html.

[Sout73] R. W. Soutas-Little. *Elasticity*. Prentice–Hall, 1973.

[VarFEM] A. Stahel. Calculus of Variations and Finite Elements. Lecture Notes used at HTA Biel, 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.

[TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.

[Wein74] R. Weinstock. *Calculus of Variations*. Dover, New York, 1974.

[Zien13] O. Zienkiewicz, R. Taylor, and J. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7 edition, 2013.

List of Figures

List of Tables

Index