FEMoctave, Finite Element Algorithms in Octave

Andreas Stahel, Bern University of Applied Sciences

Version 2.0.5 of 29th September 2022

Contents

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 and second order triangular elements in 2D for scalar boundary value problems. This leads to the *Octave* package **FEMoctave**.
  - Provide examples on how to solve steady state and dynamic heat equations and the wave equation, 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/v2.0.4.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 is included and find more information 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* by Springer.
- an introduction to FEM algorithms. The basic concept is not explained here, 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 is explained. Find the lecture notes for this class on my web site at https://andreasstahel.github.io/Notes/NumMethods.pdf.

The structure of this document:

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 **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 7. The instruction on how to use the commands are given in Section 4. Some typical examples are worked out in Section 3.

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

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

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

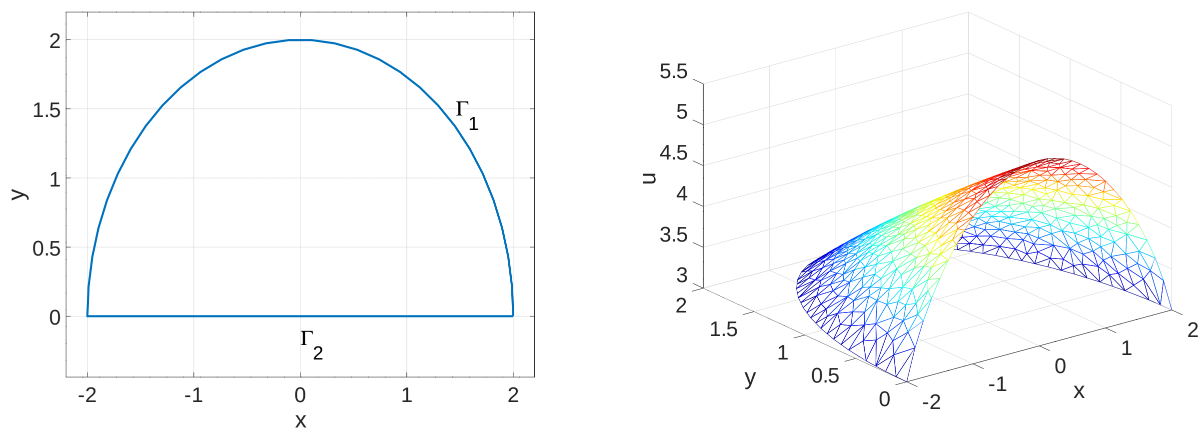


Figure 1: A semidisk as domain in $\mathbb{R}^2$ and a solution of a BVP

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)

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

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.

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

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

Using Calculus of Variations one can show that solving (2) is equivalent to minimizing 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 and 7.4.

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.

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

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.

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

This is an Initial Boundary Value Problem (IBVP). Mathematicians call this a parabolic problem, engineers think of dynamic heat equations. An example is shown in Section 3.3.

2.6 The symmetric parabolic problem

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

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

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.

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

Examples are shown in Sections 3.4 and 7.2. The effect of eigenvalues is described in Section 6.8.4.

command	type of problem	section
`BVP2Dsym()`	solve a symmetric elliptic BVP	4.3.1
`BVP2D()`	solve a general elliptic BVP	4.3.2
`BVP2Deig()`	solve a symmetric elliptic eigenvalue problem	4.4
`IBVP2D()`	solve a parabolic IBVP	4.5
`IBVP2Dsym()`	solve a symmetric parabolic IBVP	4.5
`I2BVP2D()`	solve a hyperbolic IBVP	4.6

Table 1: Commands to solve PDEs and IBVPs

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.
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 the 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 thus Neumann boundary Γ_2 at $x = 1$ and at $y = 2$ seek a solution of

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

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.

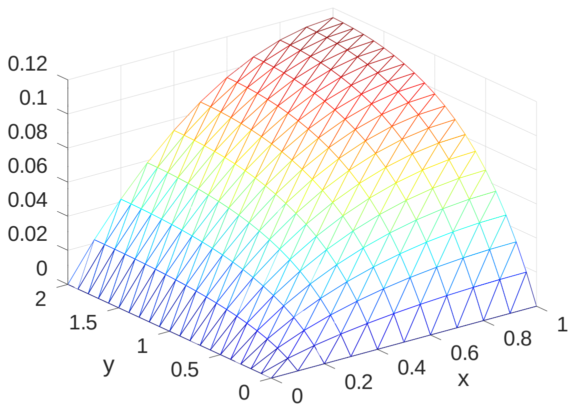


Figure 2: Solution of $-\Delta u = 0.25$ on a rectangle

LaplaceRectangle.m

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

u = BVP2Dsym(FEMmesh,1,0,0.25,0,0,0);

figure(1); FEMtrimesh(FEMmesh,u);
           xlabel('x'); ylabel('y');
```

Find the result in Figure 2. The above code is using linear elements. To use quadratic or cubic elements uncomment one of the lines with `MeshUpgrade()`.

3.1.2 Laplace equation in cylindrical coordinates

The Laplace operator in cylindrical coordinates is given by

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

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

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

Thus it is in the form of equation (2), with $x = \rho$ and $y = z$. As an example consider $b_0(\rho, z) = 10$ and $f(\rho, z) = 2\,z$. If 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 3.

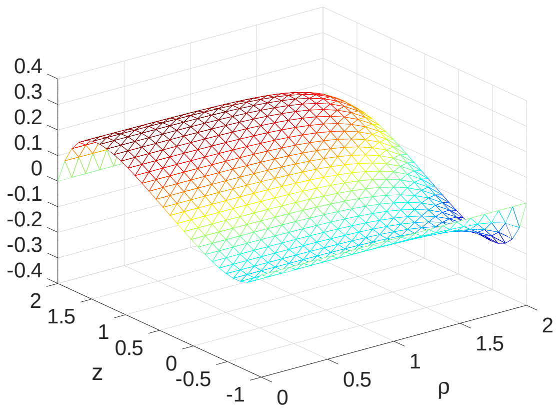


Figure 3: Solution of Lapalce equation in cylindrical coordinates

LaplaceCylindrical.m

```
FEMmesh = CreateMeshRect(linspace(0,2,20),linspace(-1,2,30),-1,-1,-2,-2);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment to use quadratic elements

function res = f(rz,dummy)  res = rz(:,1)*2.*rz(:,2); endfunction
function res = b0(rz,dummy) res = 10*rz(:,1);         endfunction
function res = a(rz,dummy)  res = rz(:,1);            endfunction
function res = g2(rz)       res = -1*rz(:,1)/2;       endfunction

u = BVP2Dsym(FEMmesh,'a','b0','f',0,'g2',0);

FEMtrimesh(FEMmesh,u);
xlabel('\rho'); ylabel('z');
```

3.1.3 Diffusion on an L-shaped domain

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

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

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

- Generate the L-shaped domain with the help of `CreateMeshTriangle()`.
- Solve the equations with `BVP2Dsym()`.
- Display the result with `FEMtrimesh()`.
- 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 4.

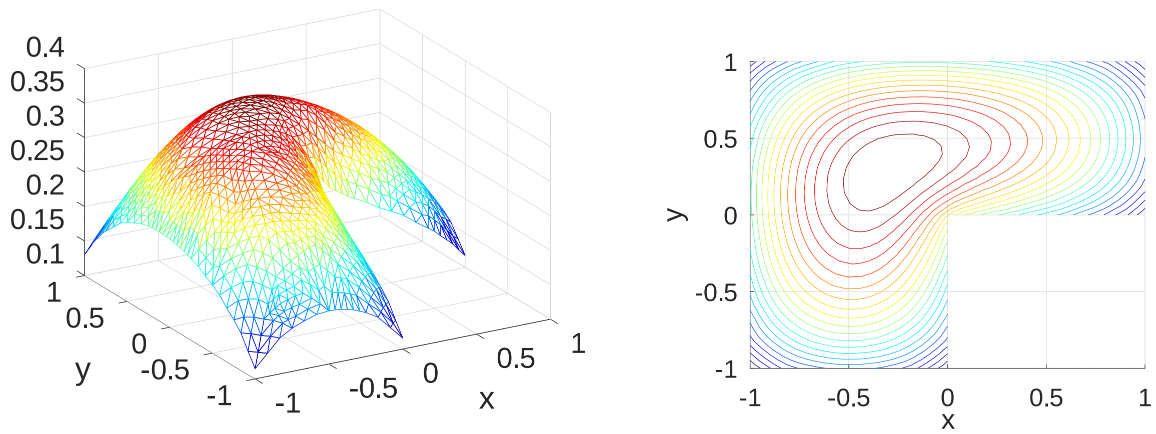


Figure 4: Solution of a diffusion problem on a L-shaped domain

DiffusionLshape.m

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

u = BVP2Dsym(FEMmesh,1,0,1,0,0,-2);

figure(1); FEMtrimesh(FEMmesh,u);
           xlabel('x'); ylabel('y'); view(-30,30)

figure(2); FEMtricontour(FEMmesh,u);
           xlabel('x'); ylabel('y');
```

3.1.4 A diffusion convection problem

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

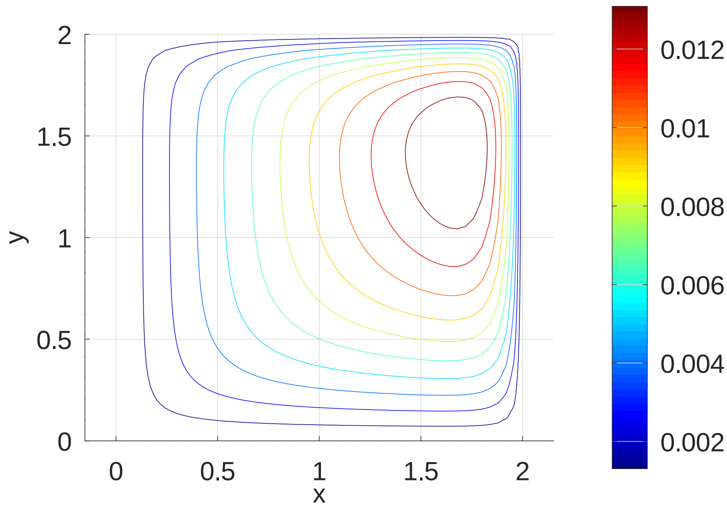


Figure 5: Solution of a diffusion convection problem

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 finer 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 6.
- To use second order element, use `MeshUpgrade()`.

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

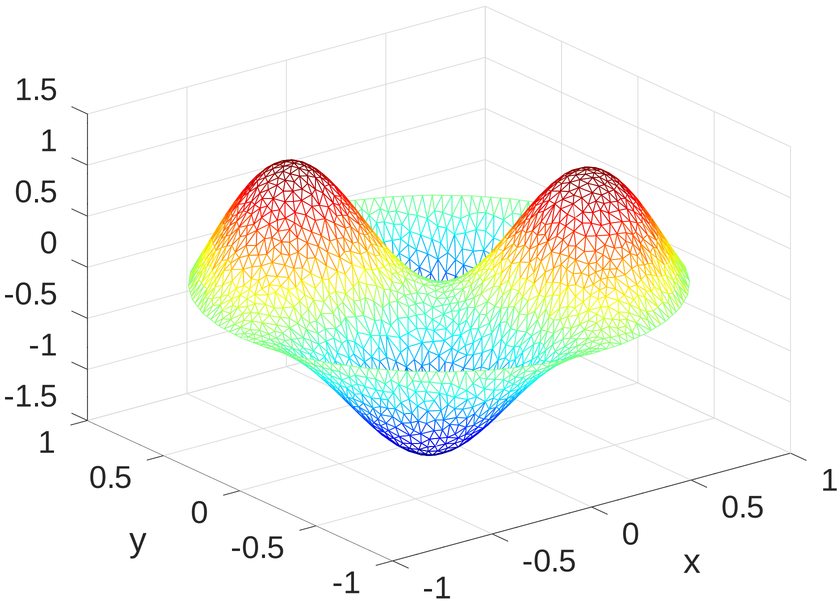


Figure 6: The fourth eigenfunction of $\Delta u = \lambda\, u$ on a disc

EigenvaluesDisc.m

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

FEMmesh = CreateMeshTriangle('circle',xy,0.0005);
%%FEMmesh = MeshUpgrade(FEMmesh,'quadratic');

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

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

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 7. 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 one shown in Figure 5 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); tricontour(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
```

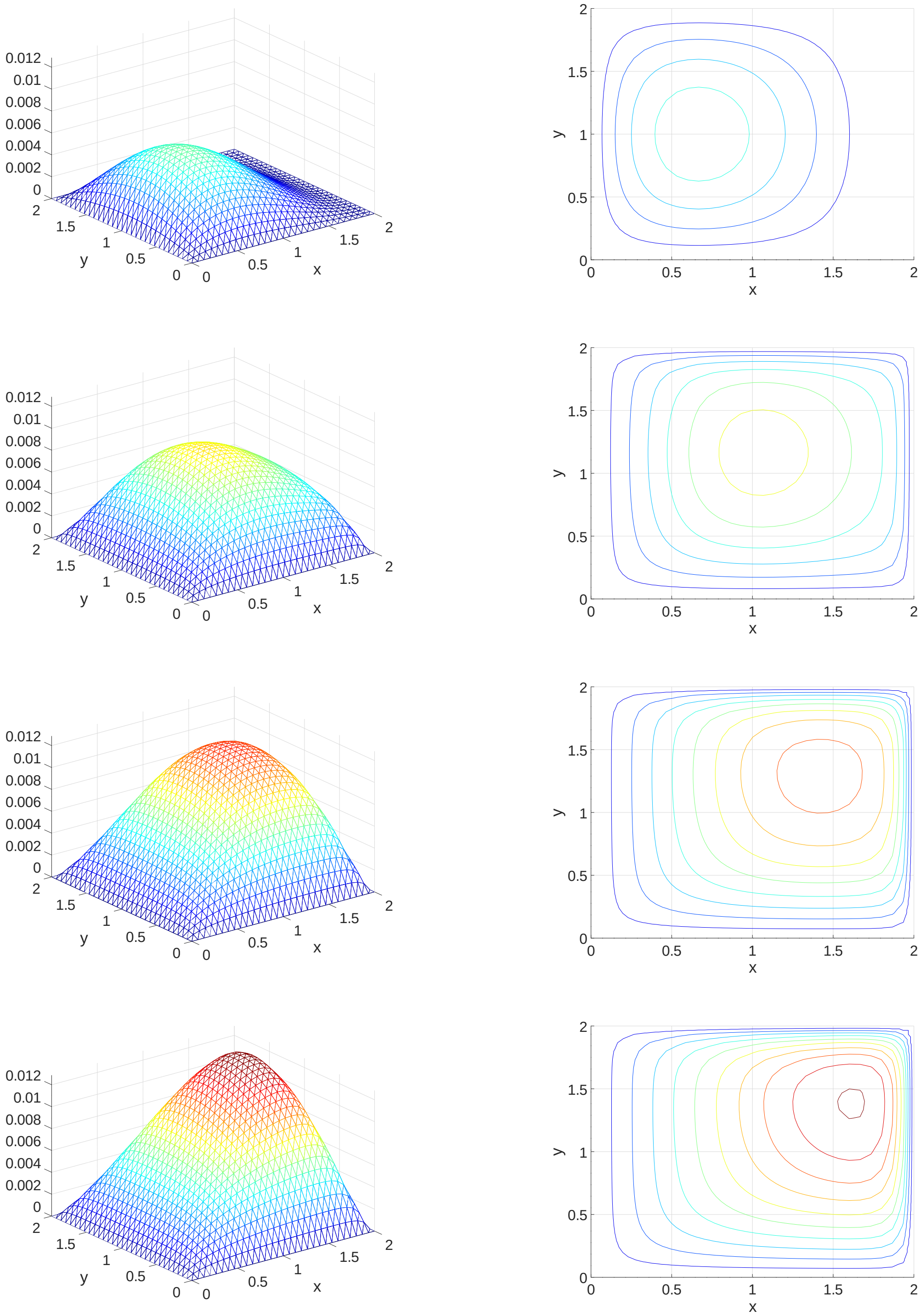


Figure 7: Solution of a dynamic heat equation

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 8. This initial hump is traveling towards to boundary of the circle with speed 1, where it is reflected. Another example is shown in Section 7.2. xs

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;
%% 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 Solving plane stress problems (later)

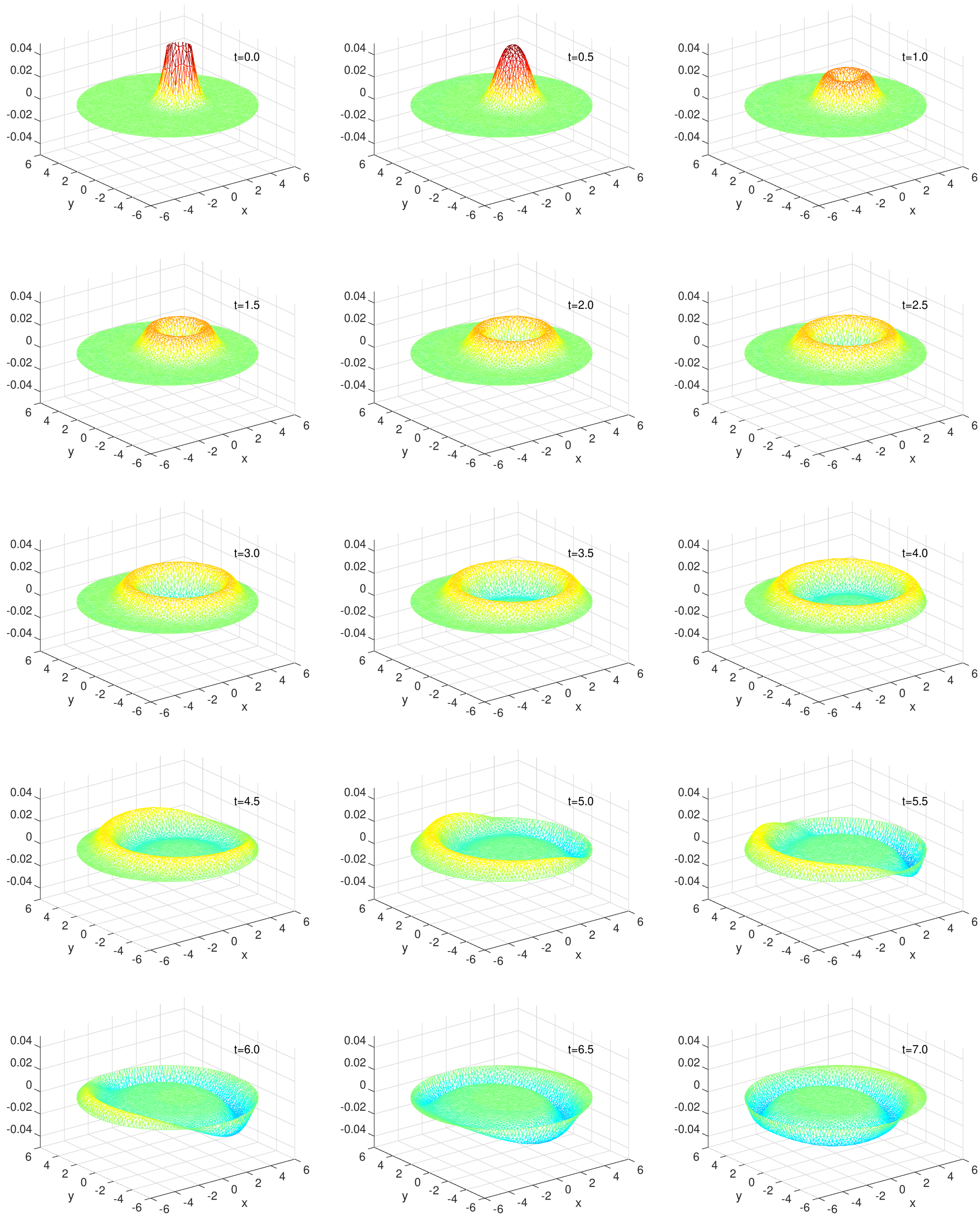


Figure 8: Solution of a wave equation

4 The Commands of FEMoctave

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

4.1 Commands for meshes: creation, evaluation, modification, integration

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 2. 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.
- `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.

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

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

Name	Size	Information
nodes	$Nn \times 2$	coordinates of nodes
nodes	$Ne \times 3$	for first order elements
	$Ne \times 6$	for second order elements
	$Ne \times 10$	for third order elements
nodesT	$Nn \times 1$	type of nodes, either 0 (free) or 1 (fixed)
elem		list of nodes that make up the triangles
elemT	$Ne \times 1$	type of elements
edges	$Nb \times \{2,3,4\}$	list of nodes that make up the boundary edges
edgesT	$Nb \times 1$	type of boundary edge, Dirichlet or Neumann
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	1×1	total number of DOF of the system
node2DOF	$Nn \times 1$	renumbering from nodes to DOF

Table 2: Elements of a mesh structure

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.
- 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.

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 mesh on a rectangle with nodes at (x_i,y_j)

   parameters:
     * X,Y are the vectors containing the coodinates of the mesh
       nodes to be generated.
     * BLOW, BUP, BLEFT, BRIGHT indicate the type of boundary
       condition at lower, upper, left and right edge of the rectangle
       B* = -1: Dirichlet boundary condition
       B* = -2: Neumann or Robin boundary condition

   return value
```

```
    * MESH is a a structure with the information about the mesh.
      The mesh consists of n_e linear elements, n_n nodes and n_ed edges.
         * 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 (not used)
         * 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 of integer, 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()` one can only generate meshes with elements of order 1. With the help of `MeshUpgrade()` (Section 4.1.4) 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 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.

CreateMeshTriangle()

```
 Mesh = CreateMeshTriangle(NAME,XY,AREA,OPTIONS)
    Generate files with a mesh with linear elements using the extenal 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
                * bi=-1 Dirichlet boundary condition
                * bi=-2 Neumann or Robin boundary condition
      * 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 or
     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 a structure with the information about the mesh.
        The mesh consists of n_e elements, n_n nodes and n_ed edges.
           * 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 (not used)
           * 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 of integer, mapping nodes to DOF

    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
```

- If a return argument for `MeshGenerateTriangle()` is provided, the mesh is returned.
- If no return argument is provided, the information is written to files. The generated mesh is the 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()`. 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 7.8.

With `CreateMeshTriange()` and `ReadMeshTriangle()` one can only generate meshes with elements of order 1. With the help of `MeshUpgrade()` (Section 4.1.4) you can upgrade to the same mesh with elements of order 2 or 3.

4.1.4 Converting meshes: upgrading and downgrading

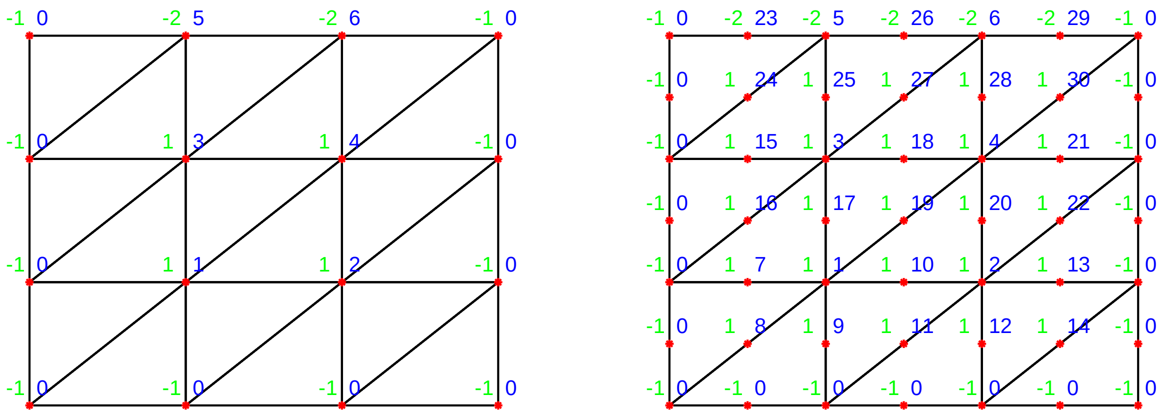
Given a mesh `MeshLin` with first order elements one can generate the same mesh with elements of order 2 by the command `MeshUpgrade(MeshLin,'quadratic')`. The numbering of the nodes of the linear elements is preserved in the mesh with the quadratic elements. The new nodes are placed at the midpoints of the edges of the triangles. With `MeshUpgrade(MeshLin,'cubic')` a mesh with 10 node cubic elements is generated. Examine Figure 26 on page 53 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 condition on three edges and a Neumann condition on the upper edge at $y = 2$. In Figure 9 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 9: 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
```

4.1.5 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 10. For almost all cases `triangle` will generate meshes of better quality. To generate the domain and the solution in Figure 10 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);

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');
```

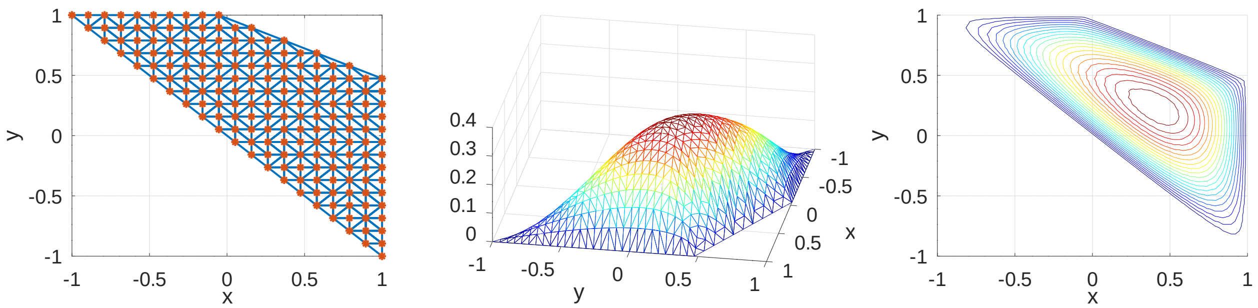


Figure 10: A mesh generated by a Delaunay triangulation and the solution of a BVP

4.1.6 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
```

To generate the quarter of a ring in Figure 24 on page 49 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 7.1.

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

To display the results of the computations elementary wrappers around `trimesh()`, `trisurf()` `tricontour()` are provided.[1]

[1] 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.

- 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.

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
```

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
```

4.1.8 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 element 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, 7.3, 7.4, 7.5 and 7.9. 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!

4.1.9 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 7.7 and 7.9.

4.1.10 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.
- 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 $0 \le x, y \le 1$. The exact integral equals $\frac{1}{8}$, but you have to subtract 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 integration's 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, 7.1, 7.7 and 7.9.

4.1.11 Evaluation at arbitrary points or along lines, 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 or pieceweise quadratic interpolation is used.
- If a point (x_i, y_i) is on the edge of a triangle is a a matter of rounding which of the neighboring triangles is used for the interpolation.
- 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 those curves. Find an example in Section 7.5.

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 11.

```
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));
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.
```

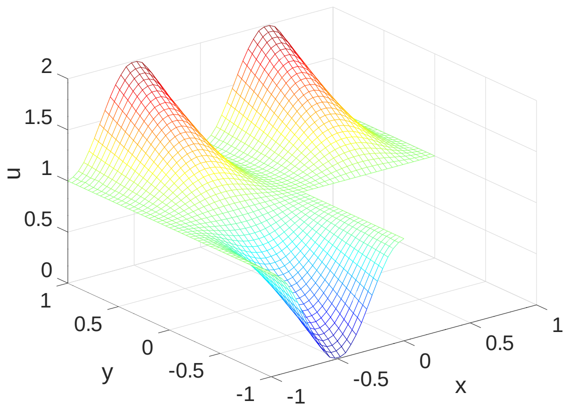


Figure 11: A function evaluated on a uniform grid

Examples are given in Sections 5.3, 7.3, 7.5 and 7.8.

4.2 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**[2]. Due to a recent change in *Octave* the script versions should use a dummy second argument[3]. The function can be implemented as a `name.m` *Octave* function or as dynamically linked function `name.oct`, written in C++.

[2]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.

[3]In the script files (`FEMEquationM()` 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()` and similar) the dummy argument is not required. I might remove this “feature” in a next release.

- 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 3 contains many examples or you may examine the examples below.

4.2.1 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 available. 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
   ```
   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.

4.2.2 Functions for dynamic problems

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

4.3 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 section the commands to solve a symmetric and a non-symmetric elliptic BVP are shown.

4.3.1 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 4.2.1, 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, 7.4, 7.5 and 7.7.

4.3.2 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 4.2.1, 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 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
```

4.4 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 4.2.1, 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.

4.5 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 Crank Nicolson 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
...
```

4.6 Solving hyperbolic problems: `I2BVP2D()`

Examine an IBVP (6) of hyperbolic type.

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

To solve this wave type equation use the command `I2BVP2D()`. Find an example in Section 7.2 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.
```

4.7 Internal commands in FEMoctave

4.7.1 Linear elements: `FEMEquation()` and `FEMEquationM()`

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,N2D] = 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
     * N2D is a vectors used to match nodes to degrees of freedom
       n2d(k)= 0 indicates that node k is a Dirichlet node n2d(k)=nn indicates
       that the value of the solution at node k is given by u(nn)
```

`FEMEquationM.m` is a simplified version, but written as an *Octave* script. Thus the code is easier to read and understand. The convection terms are not available in `FEMEquationM.m`.

4.7.2 Quadratic elements: `FEMEquationQuad()` and `FEMEquationQuadM()`

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,N2D] = 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
    * N2D is a vectors used to match nodes to degrees of freedom
      n2d(k)= 0 indicates that node k is a Dirichlet node n2d(k)=nn indicates
      that the value of the solution at node k is given by u(nn)
```

`FEMEquationQuadM.m` is a simplified version, but written as an *Octave* script. Thus the code is easier to read and understand. The convection terms are not available in `FEMEquationQuadM.m`.

4.7.3 Cubic elements: `FEMEquationCubic()` and `FEMEquationCubicM()`

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

4.7.4 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()`.

4.7.5 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
```

4.7.6 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()
```

4.8 External programs

To construct the meshes and display contour lines FEMoctave uses external programs.

- **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 used in the function `FEMtricontour()` and available at https://ch.mathworks.com/matlabcentral/fileexchange/38858-contour-plot-for-scattered-data . The current version of FEMoctave contains a simple implementation of `tricontour.m` and thus the code from `Matlab Central` is not required any more. 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 classes on 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 the function $u_e(x,y) = \sin(x) \cdot \sin(y)$ is solution of

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

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 12.

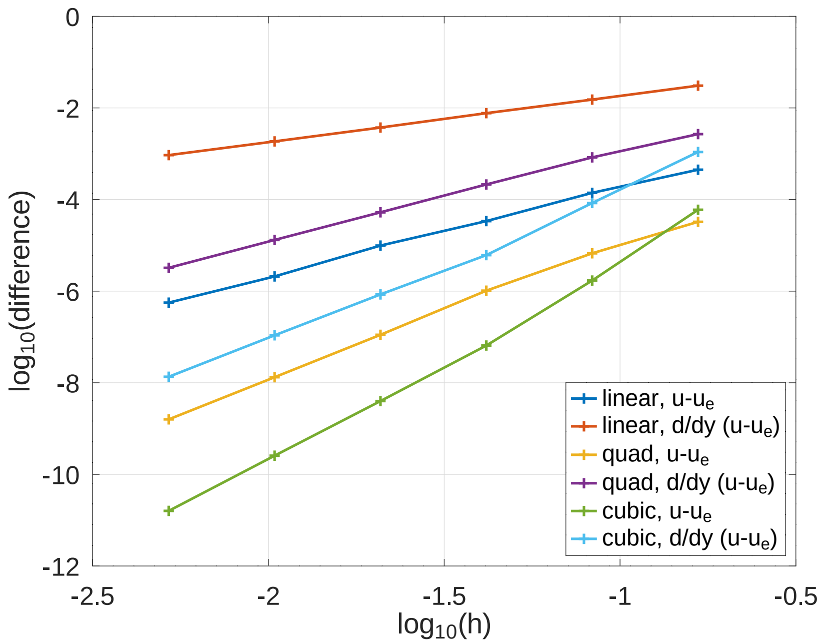


Figure 12: Convergence results for linear, quadratic and cubic elements

- 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 $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 1 and thus conclude that the error of the gradient is proportional to h.

- For quadratic elements:
  - The slope of the curve for the absolute values of $u(x,y) - u_e(x,y)$ is approximately 3 and thus conclude that the error is proportional to h^3.
  - The slope of the curve for the absolute values of $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 2 and thus conclude that the error of the gradient is proportional to h^2.
- For cubic elements:
  - The slope of the curve for the absolute values of $u(x,y) - u_e(x,y)$ is approximately 4 and thus conclude that the error is proportional to h^4.
  - The slope of the curve for the absolute values of $\frac{\partial}{\partial y}\left(u(x,y) - u_e(x,y)\right)$ is approximately 3 and thus conclude that the error of the gradient is proportional to h^3.

These observations confirm the theoretical error estimates in Section 6.7 on page 85. It is rather obvious from Figure 12 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)  res =   sin(xy(:,1)).*sin(xy(:,2));   endfunction
function res = f(xy)        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 `GenerateMeshTriangle`. The code in `CreateTriangle.m` generates the mesh and Figure 13.

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
```

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

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

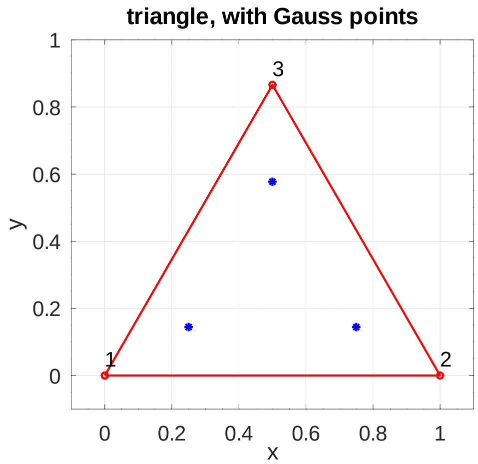


Figure 13: An linear, equilateral triangle, the Gauss integration points and the element stiffness matrix

For the PDE $-\Delta u = 1$ generate the element stiffness matrix $\mathbf{A}$ and the element vector $\vec{f}$ with he help of the commands `FEMEquation()` or `FEMEquationM()`.

```
[A2,f2] = FEMEquationM(mm,1,0,1,0,0);       %% using the script
[A,f]   = FEMEquation (mm,1,0,0,0,1,0,0,0); %% using compiled code
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 13.

Using the same idea one can examine the contributions of the different term 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 triangle, as shown in Figure 14. Then examine the linear equation corresponding to an interior point at (x_i, y_i).

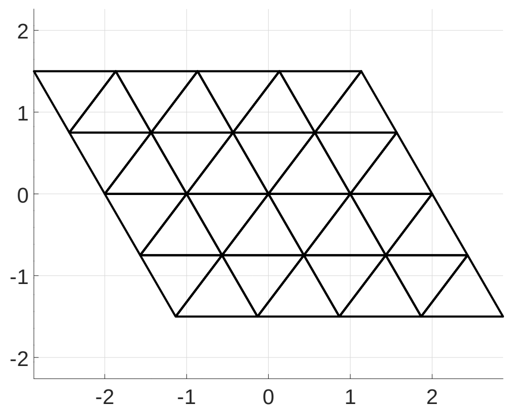


Figure 14: Uniform meshes consisting of equilateral triangles

- 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 13, 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 13, i.e. $a_{i,j} = 2\,\frac{-1}{2\sqrt{3}} = \frac{-1}{\sqrt{3}}$.
- If the function f in $-\nabla^2 u = f$ is constant, then there will be 6 contributions from the six neighboring triangle. If the length of one side of a triangle equals h, then the area is $\frac{\sqrt{3}}{4}\,h^2$. Thus find $b_i = 6\,\frac{\text{area of triangle}}{3}\,(-f) = -\frac{\sqrt{3}}{2}\,h^2\,f$.

As a result find the equation for the node at (x_i, y_i).

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

This is somewhat similar to a finite difference approximation. For each row of the global stiffness matrix the entry on the diagonal and 6 more will be different from 0.

One can examine second order elements and the resulting element stiffness matrix and vector for quadratic elements for the PDE $-\Delta u = 1$. The triangular, equilateral element and the matrix are shown in Figure 15. The

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

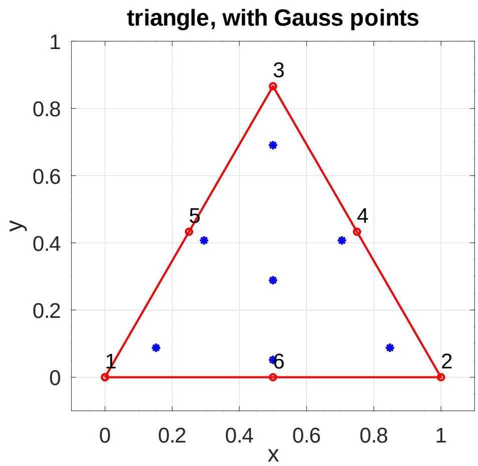


Figure 15: An equilateral, quadratic triangle, the Gauss integration points and the element stiffness matrix

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 a very regular mesh in Figure 14 on each row of the matrix 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.

5.2.2 From FEM to a finite difference approximation

Generate the trivial mesh consisting of a single equilateral triangle with the help of `GenerateMeshTriangle`. The code in `CreateTriangle.m` generates the mesh and Figure 16. 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
```

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

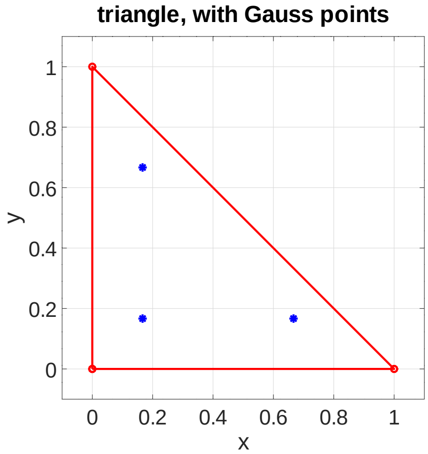
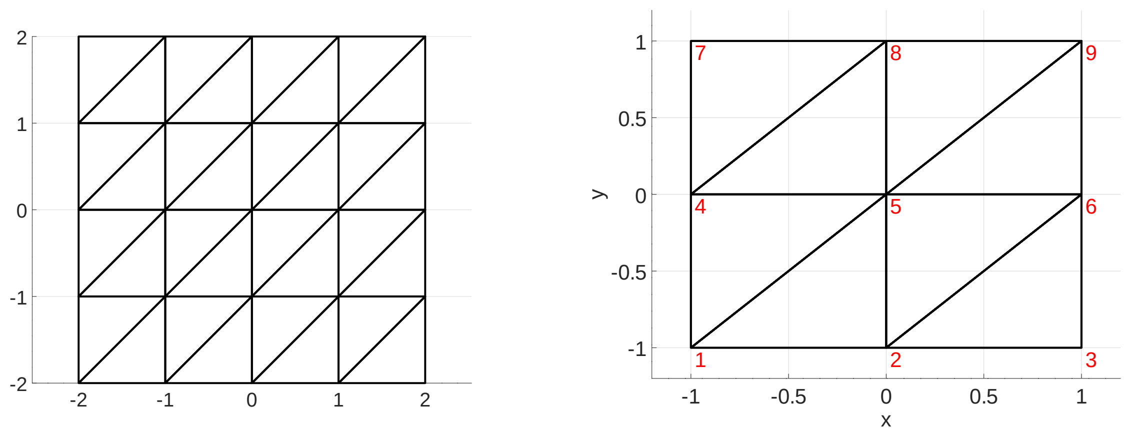


Figure 16: A right triangle, the Gauss integration points and the element stiffness matrix



(a) a section

(b) a small section, numbered

Figure 17: Uniform meshes consisting of rectangular triangles

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 17. 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=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 18. The vector is given by

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

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

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

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

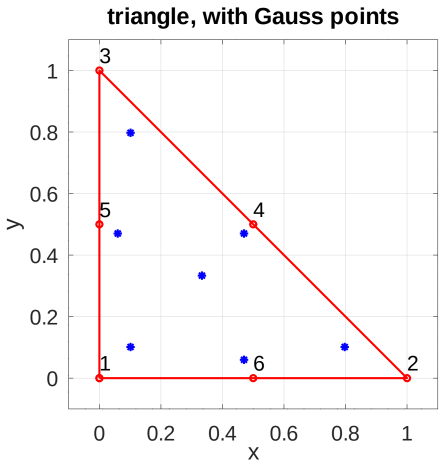
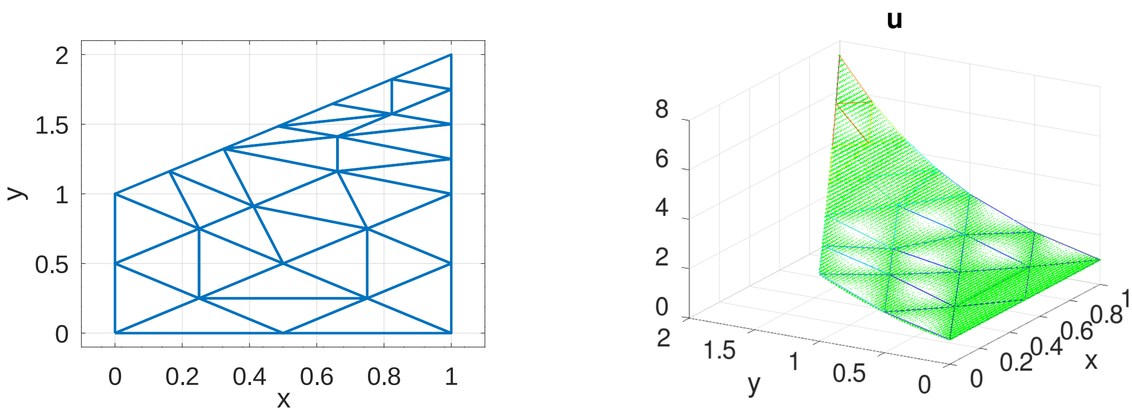


Figure 18: A right angle triangle, the Gauss integration points and the element stiffness matrix

on the domain Ω displayed in Figure 19(a). The exact solution is given by $u(x, y) = \exp(y)$, shown in Figure 19(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 19(a) for the linear elements correspond to one of the eight triangles for the quadratic elements.

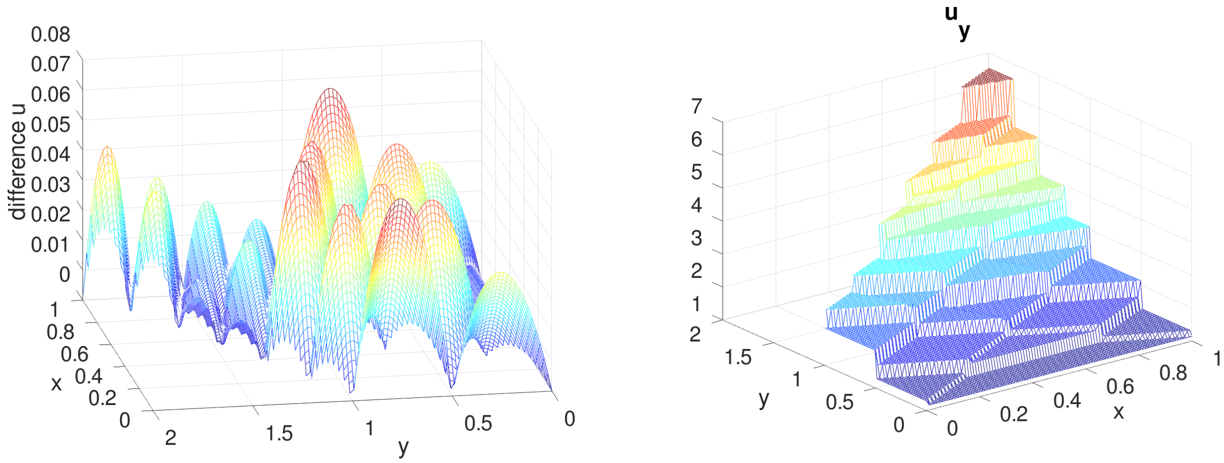


(a) the mesh (b) the solution

Figure 19: The mesh and the solution for a BVP

Figure 20(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 20(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.

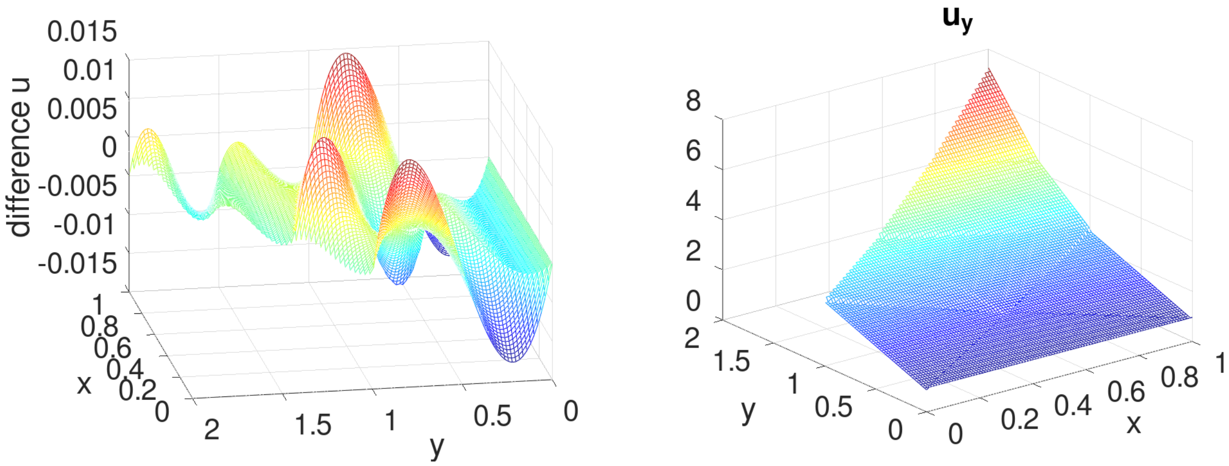
Figure 21(a) shows the difference of the computed solution with second order elements to the exact solution. The error is considerably smaller than for linear elements, using identical degrees of freedom. Within each of the 8 elements the difference does not show a simple structure. Figure 21(b) shows 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



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

Figure 20: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a first order mesh

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 22 shows the errors for the partial derivative $\frac{\partial u}{\partial y}$ and confirms this observation.



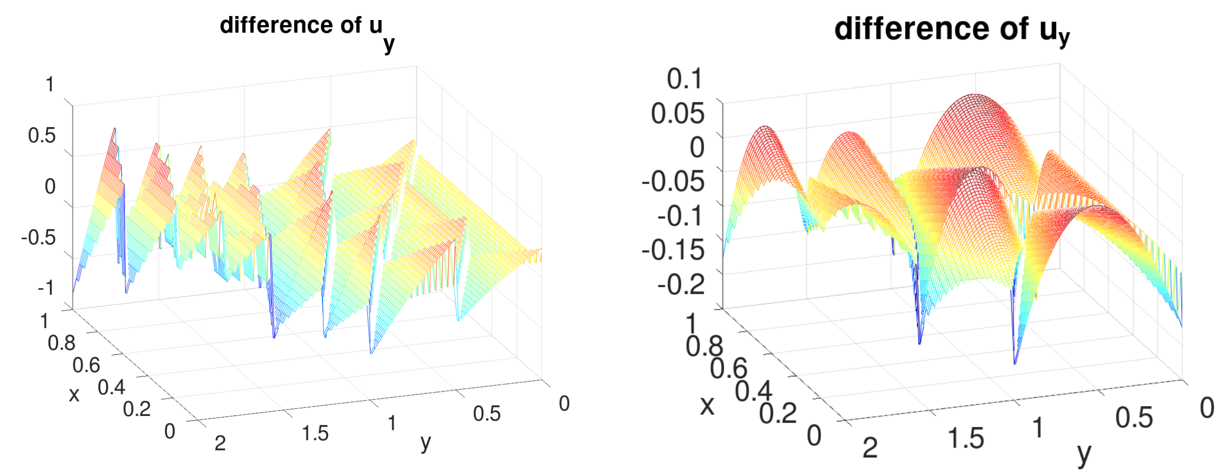
(a) the difference to the exact solution (b) the values of $\frac{\partial u}{\partial y}$

Figure 21: Difference to the exact solution and values of $\frac{\partial u}{\partial y}$, using a second order mesh

In Figure 23 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.

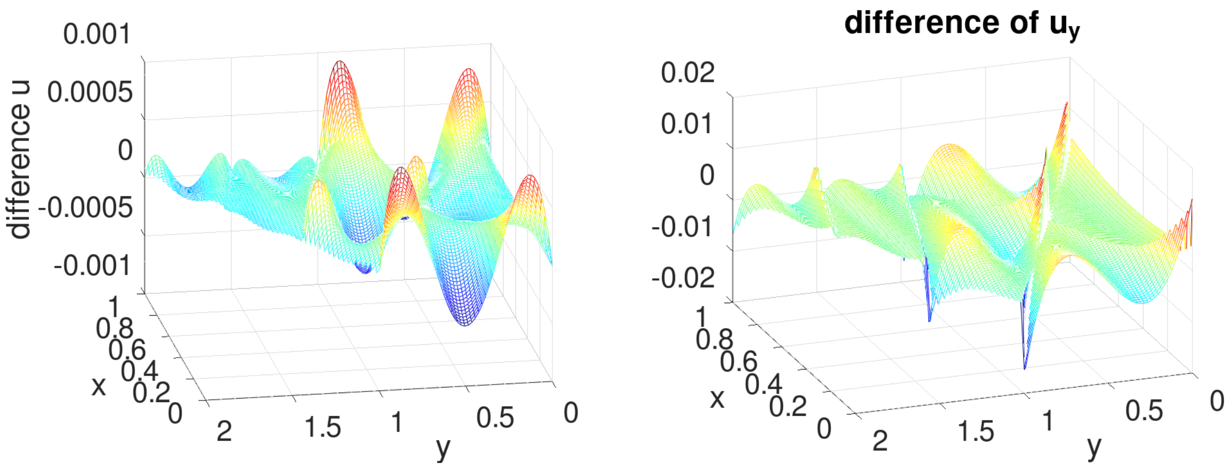
FEMInsideElement

```
N = 2; MeshType = 'quadratic'  %% use 'linear', 'quadratic' or 'cubic'
Mesh = CreateMeshTriangle('test',[0 0 -1;1 0 -1;1 2 -1; 0 1 -1],1/N^2);
switch MeshType
```



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

Figure 22: Difference of the approximate values of $\frac{\partial u}{\partial y}$ to the exact values



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

Figure 23: Difference of the approximate values of u and $\frac{\partial u}{\partial y}$ to the exact values for cubic elements

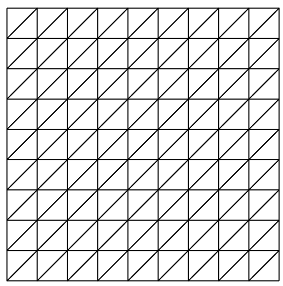
```
  case 'quadratic'
    Mesh = MeshUpgrade(Mesh,'quadratic');
  case 'cubic'
    Mesh = MeshUpgrade(Mesh,'cubic');
endswitch

x = Mesh.nodes(:,1);  y = Mesh.nodes(:,2);
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.

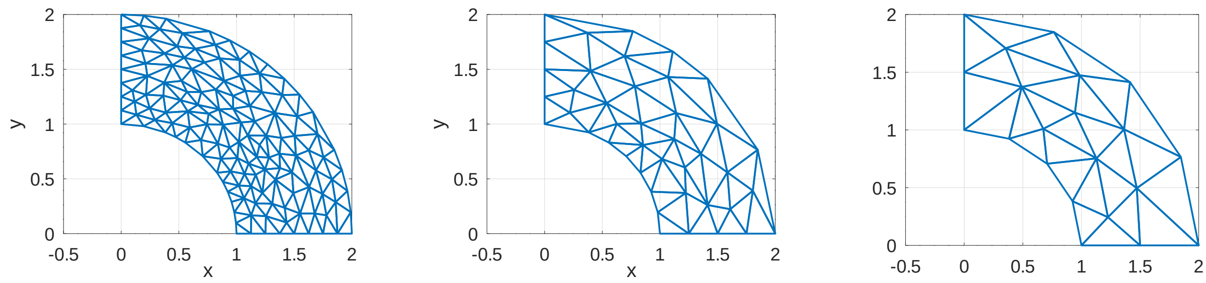
This points to about a factor of $\frac{11.5}{7} \approx 1.6$ more nonzero entries in the matrix for quadratic elements for the same number of degrees of freedom. For cubic elements expect a factor of $\frac{17}{7} \approx 2.4$. This implies that the computational effort is larger, the actual effect 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 24. 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 24: 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 24. 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 85.
- 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 3 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 3: Results for elements of order 1, 2 and 3

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

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

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

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

$$\begin{aligned} 0 &= -\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\,u - f \\ 0 &= \iint\limits_{\Omega} \phi\,\left(-\nabla \cdot (a\,\nabla u - u\,\vec{b}) + b_0\,u - f\right)\,dA \\ &= \iint\limits_{\Omega} \nabla\phi \cdot (a\,\nabla u - u\,\vec{b}) + \phi\,(b_0\,u - f)\,dA - \int_{\Gamma} \phi\,\left(a\,\nabla u - u\,\vec{b}\right) \cdot \vec{n}\,ds \\ &= \iint\limits_{\Omega} \nabla\phi \cdot (a\,\nabla u - u\,\vec{b}) + \phi\,(b_0\,u - f)\,dA - \int_{\Gamma_2} \phi\,(g_2 + g_3\,u)\,ds\,. \end{aligned} \tag{7}$$

If a function u satisfies (7) 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 25 shows connections between classical solutions, weak solutions and the resulting system of (linear) equations for the finite element approach. The left branch in Figure 25 illustrates the usage of minimization and calculus of variations in the context of FEM algorithms.

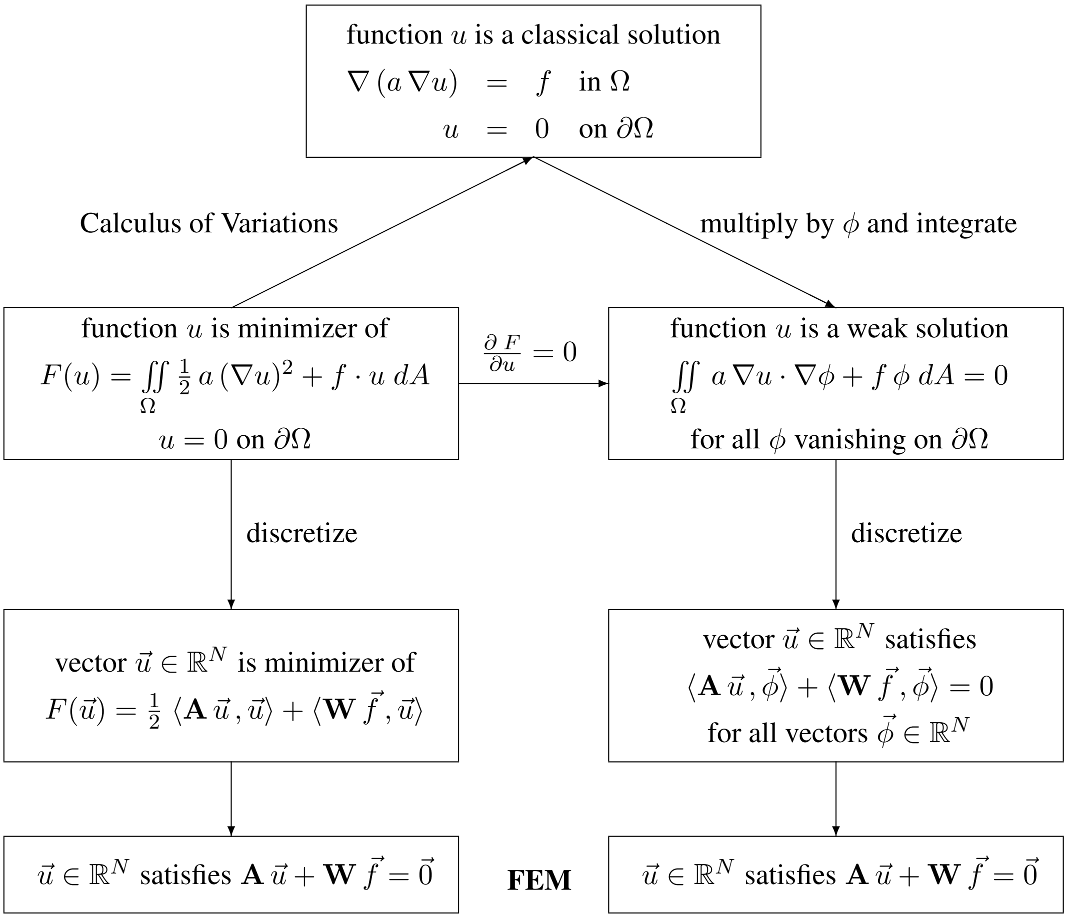
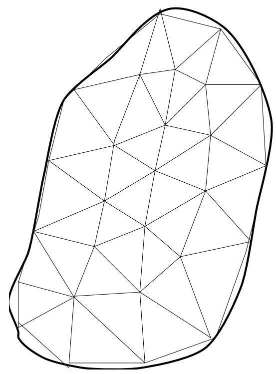


Figure 25: Classical and weak solutions, minimizers and FEM

In the above equation integrals over the domain $\Omega \subset \mathbb{R}^2$ have to be computed. To discretize this process use a triangularization of the domain, 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. At the end each contribution is 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, 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 26 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.
- A double circle around a solid dot at a position indicates that the values of the second order partial derivatives are used as DOFs.
- A short line at a position indicates that the value of the normal derivative at this point is used as a DOF.

	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 4: Properties of triangular elements

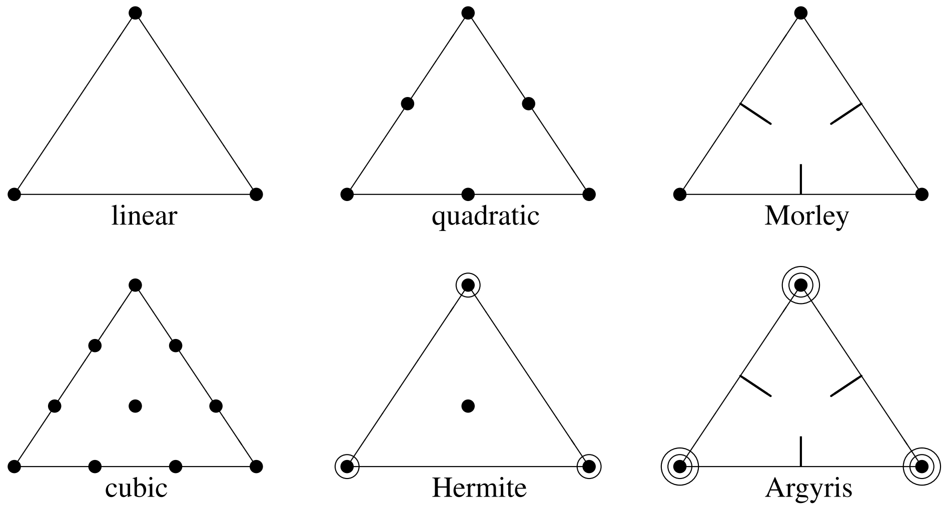


Figure 26: A few triangular elements

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 an 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 27. The transformation is given by

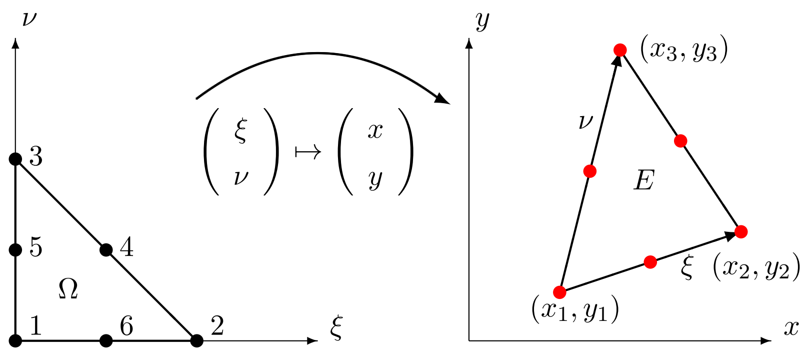


Figure 27: Transformation of standard triangle Ω to a general triangle E

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

where

$$\mathbf{T} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} .$$

By using $0 < \xi, \nu < 1$ with $\xi + \nu < 1$ the standard triangle Ω is mapped onto the general triangle $E \subset \mathbb{R}^2$. If the coordinates (x, y) are given find the values of (ξ, ν) with the help of

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

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

$$\iint\limits_E f \, dA = \iint\limits_\Omega f(\vec{x}(\xi,\nu)) \left| \det \left(\frac{\partial(x,y)}{\partial(\xi,\nu)} \right) \right| \, d\xi \, d\nu = |\det(\mathbf{T})| \int_0^1 \left(\int_0^\nu f(\vec{x}(\xi,\nu)) \, d\xi \right) d\nu \, . \tag{8}$$

The Jaccobi determinant is given by

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

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

$$\iint\limits_\Omega f(\vec{\xi}) \, dA \approx \sum_{j=1}^m w_j \, f(\vec{g}_j) \, . \tag{9}$$

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 28 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

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

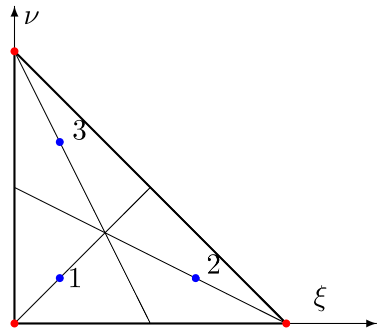


Figure 28: Gauss integration of order 2 on the standard triangle, using 3 integration points

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.[4]

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 29. 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 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 $0 \leq k \leq 5$ 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.

[4] 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.

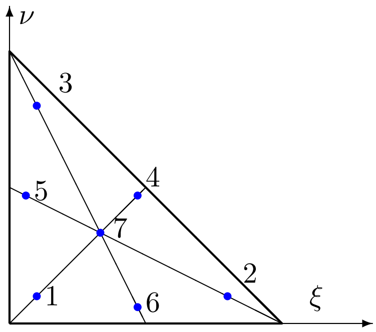


Figure 29: Gauss integration of order 5 on the standard triangle, using 7 integration points

The optimal choice of Gauss points and integration weights is given by[5]

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

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

$$\mathbf{X}_G = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \mathbf{G}^T . \tag{10}$$

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 (7) 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 (7)

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

have to be transformed into

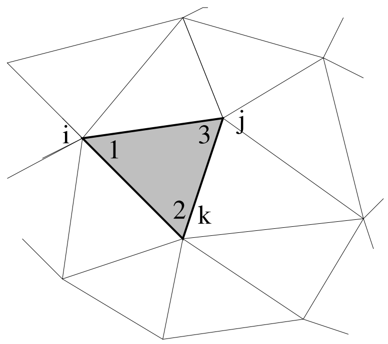
$$0 = \langle \vec{\phi}\,,\,\mathbf{A}\vec{u} + \mathbf{W}\vec{f}\rangle\,. \tag{11}$$

By integration over one triangle E find

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

[5] 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 $w_7 = 9/80$.

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

Figure 30: Local and global numbering of nodes

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

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

$$\begin{aligned} \iint\limits_E f\,\phi\,dA &\approx w\;2\,\text{area}(E)\;(f_1\,\phi(\vec{g}_1) + f_2\,\phi(\vec{g}_2) + f_3\,\phi(\vec{g}_3)) \\ &= \frac{2\,\text{area}(E)}{6}\,\langle \mathbf{M}\,\vec{\phi}\,,\,\vec{f}\rangle = \frac{\text{area}(E)}{3}\,\langle\vec{\phi}\,,\,\mathbf{M}^T\vec{f}\rangle\,. \end{aligned}$$

  Thus find one contribution to (11).

- 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 (11).

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 (11).

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[6] 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

$$\nabla u = \frac{-1}{2\,\text{area}(E)} \begin{bmatrix} (y_3 - y_2) & (y_1 - y_3) & (y_2 - y_1) \\ (x_2 - x_3) & (x_3 - x_1) & (x_1 - x_2)\end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3\end{pmatrix} = \frac{-1}{2\,\text{area}(E)}\, \mathbf{G} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3\end{pmatrix}. \tag{12}$$

[6] 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

$$\mathbf{G}_\mathbf{x} = \begin{bmatrix} y_3-y_2 & y_1-y_3 & y_2-y_1\end{bmatrix} \quad\text{and}\quad \mathbf{G}_\mathbf{y} = \begin{bmatrix} x_2-x_3 & x_3-x_1 & x_1-x_2\end{bmatrix}\,.$$

Let $b_{1,i}$ be the values of the first component of $\vec{b}$ at the Gauss nodes and find

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

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

$$\iint\limits_E u\,b_2\,\phi_y\;dA \approx \frac{-\text{area}(E)}{3}\sum_{i=1}^{3} u(\vec{g}_i)\,b_{2,i}\,\phi_{y,i}$$

$$= \frac{-1}{6}\langle \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix}, \begin{bmatrix} b_{2,1}(x_2-x_3) & b_{2,2}(x_2-x_3) & b_{2,3}(x_2-x_3) \\ b_{2,1}(x_3-x_1) & b_{2,2}(x_3-x_1) & b_{2,3}(x_3-x_1) \\ b_{2,1}(x_1-x_2) & b_{2,2}(x_1-x_2) & b_{2,3}(x_1-x_2) \end{bmatrix} \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}\rangle .$$

This leads to two more contributions to (11).

6.4.6 Integration over boundary segments

In expression (7) 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[7]

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

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

$$\begin{aligned} \int \phi\, g_2\, ds &\approx \frac{L}{2}\langle \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}, \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \end{pmatrix}\rangle \\ &= \frac{L}{2}\langle \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}, \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} g_2(\vec{p}_1) \\ g_2(\vec{p}_2) \end{pmatrix}\rangle \\ \int \phi\, g_3\, u\, ds &\approx \frac{L}{2}\langle \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}, \begin{bmatrix} g_3(\vec{p}_1) & 0 \\ 0 & g_3(\vec{p}_2) \end{bmatrix} \begin{bmatrix} (1-\alpha) & \alpha \\ \alpha & (1-\alpha) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}\rangle \end{aligned}$$

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

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

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

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

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

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

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 (7)

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

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 27 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 27. The parameters ξ and ν at the nodes are given by Table 5. 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 5: Coordinates of the nodes in the standard quadratic triangle

$$\vec{\Phi}(\xi,\nu)=\begin{pmatrix}\Phi_1(\xi,\nu)\\ \Phi_2(\xi,\nu)\\ \Phi_3(\xi,\nu)\\ \Phi_4(\xi,\nu)\\ \Phi_5(\xi,\nu)\\ \Phi_6(\xi,\nu)\end{pmatrix}=\begin{pmatrix}(1-\xi-\nu)\,(1-2\,\xi-2\,\nu)\\ \xi\;(2\,\xi-1)\\ \nu\;(2\,\nu-1)\\ 4\,\xi\,\nu\\ 4\,\nu\;(1-\xi-\nu)\\ 4\,\xi\;(1-\xi-\nu)\end{pmatrix}\tag{13}$$

and find their graphs in Figure 31.

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

$$f(\xi,\nu)=\sum_{i=1}^{6} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu)\,.=\sum_{i=1}^{6} f_i\,\Phi_i(\xi,\nu)\,.\tag{14}$$

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

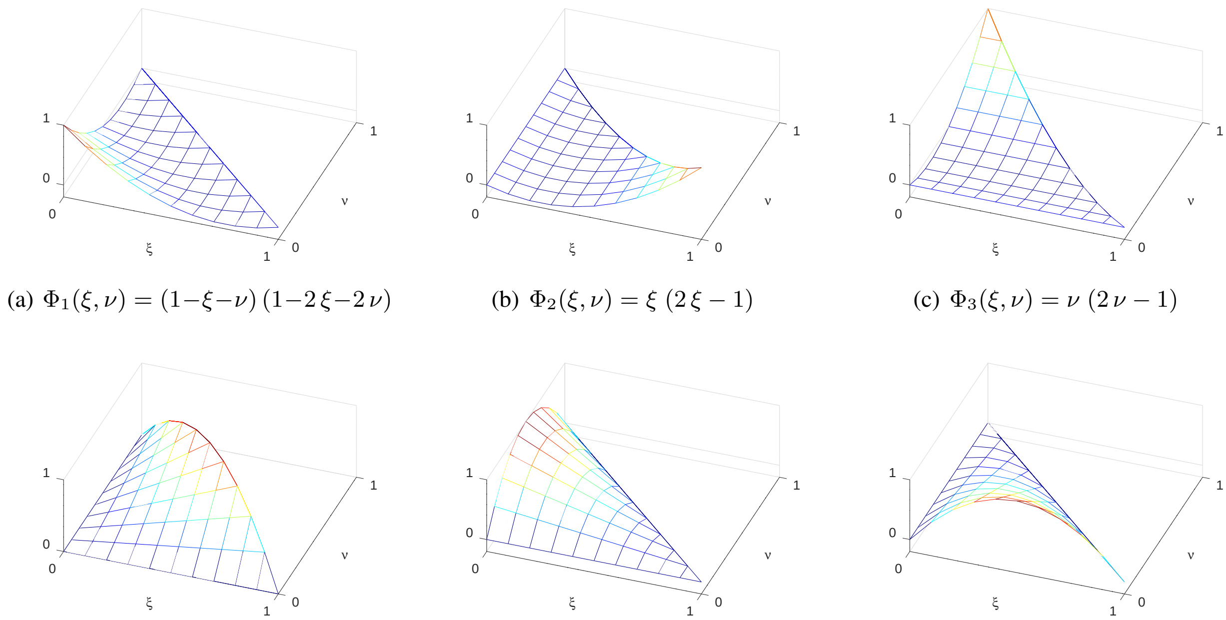
$$\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}x_1\\ y_1\end{pmatrix}+\xi\begin{pmatrix}x_2-x_1\\ y_2-y_1\end{pmatrix}+\nu\begin{pmatrix}x_3-x_1\\ y_3-y_1\end{pmatrix}.$$

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 31: Basis functions for second order triangular elements

6.5.2 Determine values at the Gauss points and apply Gauss integration

Use equation (10) 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

$$\begin{aligned}
\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} &= \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,6} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,6} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_6 \end{pmatrix} = \mathbf{M}\cdot\vec{f} \\
&\approx \begin{bmatrix}
+0.474353 & -0.080769 & -0.080769 & 0.041036 & 0.323074 & 0.323074 \\
-0.080769 & +0.474353 & -0.080769 & 0.323074 & 0.041036 & 0.323074 \\
-0.080769 & -0.080769 & +0.474353 & 0.323074 & 0.323074 & 0.041036 \\
-0.052584 & -0.028075 & -0.028075 & 0.884134 & 0.112300 & 0.112300 \\
-0.028075 & -0.052584 & -0.028075 & 0.112300 & 0.884134 & 0.112300 \\
-0.028075 & -0.028075 & -0.052584 & 0.112300 & 0.112300 & 0.884134 \\
-0.111111 & -0.111111 & -0.111111 & 0.444444 & 0.444444 & 0.444444
\end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix}
\end{aligned}$$

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 (8), 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.

6.5.3 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

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

  Thus find one contribution to (11).

- 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

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

The matrices $\mathbf{M}^T\,\mathrm{diag}(\vec{w})$ and $\mathbf{M}^T\,\mathrm{diag}(\vec{w})\,\mathbf{M}$ are independent on the triangle E.

6.5.4 Integration of $b_0\,u\,\phi$

Since the values of the functions u and ϕ are known at the nodes use an interpolation and then the function $b_0(x,y)$ at the Gauss nodes to find

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

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

6.5.5 Transformation of the gradient to the standard triangle

To examine the contributions containing ∇u or $\nabla \phi$ requires considerably more tools than the ones used in Section 6.4.4 for linear elements. For linear elements the gradients are constant on each of the triangles. For quadratic elements the gradients are linear functions and thus not constant. First examine how the gradient behave under the transformation to the standard triangle, only then use the above integration methods.

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

$$\left(\frac{\partial\, g}{\partial \xi} \,,\, \frac{\partial\, g}{\partial \nu} \right) = \left(\frac{\partial\, f}{\partial x} \,,\, \frac{\partial\, f}{\partial y} \right) \cdot \mathbf{T} . \tag{15}$$

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

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

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 (13). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (13). The result is

$$\text{grad}\,\vec{\Phi} = \begin{bmatrix} -3+4\,\xi+4\,\nu & -3+4\,\xi+4\,\nu \\ 4\,\xi-1 & 0 \\ 0 & 4\,\nu-1 \\ 4\,\nu & 4\,\xi \\ -4\,\nu & 4-4\,\xi-8\,\nu \\ 4-8\,\xi-4\,\nu & -4\,\xi \end{bmatrix} = \begin{bmatrix} \vec{\Phi}_\xi(\xi,\nu) & \vec{\Phi}_\nu(\xi,\nu) \end{bmatrix}. \tag{16}$$

Thus find on the standard triangle Ω

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

If the function $\varphi(x,y)$ is given on the general triangle E as linear combination of the basis functions on E find

$$\varphi(x,y) = \sum_{i=1}^{6}\varphi_i\ \Phi_i(\xi(x,y)\,,\,\nu(x,y))\,.$$

Now combine the results in this section to conclude

$$\left(\frac{\partial\,\varphi}{\partial x}\,,\,\frac{\partial\,\varphi}{\partial y}\right) = \left(\frac{\partial\,\varphi}{\partial\xi}\,,\,\frac{\partial\,\varphi}{\partial\nu}\right)\cdot\mathbf{T}^{-1} = \vec{\varphi}^T\cdot\begin{bmatrix} \vec{\Phi}_\xi & \vec{\Phi}_\nu \end{bmatrix}\cdot\mathbf{T}^{-1}$$

or by transposition

$$\begin{pmatrix} \frac{\partial\varphi}{\partial x} \\ \frac{\partial\varphi}{\partial y} \end{pmatrix} = \left(\mathbf{T}^{-1}\right)^T\cdot\begin{bmatrix} \vec{\Phi}_\xi^T \\ \vec{\Phi}_\nu^T \end{bmatrix}\cdot\vec{\varphi} = \frac{1}{\det(\mathbf{T})}\begin{bmatrix} +y_3-y_1 & -y_2+y_1 \\ -x_3+x_1 & +x_2-x_1 \end{bmatrix}\cdot\begin{bmatrix} \vec{\Phi}_\xi^T \\ \vec{\Phi}_\nu^T \end{bmatrix}\cdot\vec{\varphi}$$

and the same identities can be spelled out for the two components independently.

$$\frac{\partial\,\varphi}{\partial x} = \frac{1}{\det(\mathbf{T})}\left[\,(+y_3-y_1)\,\vec{\Phi}_\xi^T + (-y_2+y_1)\,\vec{\Phi}_\nu^T\,\right]\cdot\vec{\varphi}\,, \tag{17}$$

$$\frac{\partial\,\varphi}{\partial y} = \frac{1}{\det(\mathbf{T})}\left[\,(-x_3+x_1)\,\vec{\Phi}_\xi^T + (+x_2-x_1)\,\vec{\Phi}_\nu^T\,\right]\cdot\vec{\varphi} \tag{18}$$

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 (16) to find them at the nodes, given by the (ξ, ν) coordinates in Table 5 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 (17) and (18) 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

$$\begin{aligned}
\begin{pmatrix} \varphi_x(x_1,y_1) \\ \varphi_x(x_2,y_2) \\ \vdots \\ \varphi_x(x_6,y_6) \end{pmatrix} &= \frac{1}{\det(\mathbf{T})} \left[\; (+y_3-y_1)\,\mathbf{N}_\xi^T + (-y_2+y_1)\,\mathbf{N}_\nu^T \; \right] \cdot \vec{\varphi}\,, \\
\begin{pmatrix} \varphi_y(x_1,y_1) \\ \varphi_y(x_2,y_2) \\ \vdots \\ \varphi_y(x_6,y_6) \end{pmatrix} &= \frac{1}{\det(\mathbf{T})} \left[\; (-x_3+x_1)\,\mathbf{N}_\xi^T + (+x_2-x_1)\,\mathbf{N}_\nu^T \; \right] \cdot \vec{\varphi}\,.
\end{aligned}$$

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 (7) 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 we 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[8]

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

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[9]

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

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

$$f(\pm\alpha\, h) = f_0 \pm (f_+ - f_-)\,\alpha + 2\,(f_- - 2\, f_0 + f_+)\,\alpha^2$$

where $\alpha = \frac{\sqrt{3}}{2\sqrt{5}}$. If a function u is given at the two endpoints by u_1 and u_3 and at the midpoint by u_2 obtain

$$\begin{pmatrix} u(\vec{p}_1)\\ u(\vec{p}_2)\\ u(\vec{p}_3)\end{pmatrix} = \begin{bmatrix} +\alpha + 2\,\alpha^2 & 1 - 4\,\alpha^2 & -\alpha + 2\,\alpha^2\\ 0 & 1 & 0\\ -\alpha + 2\,\alpha^2 & 1 - 4\,\alpha^2 & +\alpha + 2\,\alpha^2\end{bmatrix}\begin{pmatrix} u_1\\ u_2\\ u_3\end{pmatrix}$$

---

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

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

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

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

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

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

$$= \mathbf{M}_B \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \approx \begin{bmatrix} +0.68730 & 0.4 & -0.08730 \\ 0 & 1 & 0 \\ -0.08730 & 0.4 & +0.68730 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

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 (7)

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

have to be transformed into

$$0 = \langle \vec{\phi}\,,\ \mathbf{A}\vec{u} + \mathbf{W}\vec{f}\rangle\,. \tag{19}$$

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.

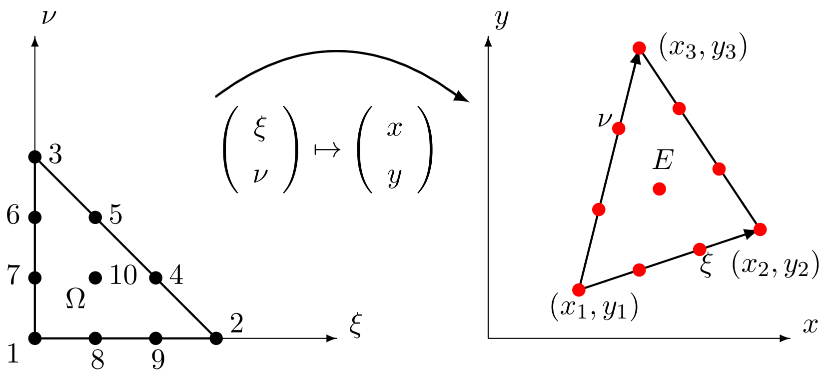


Figure 32: Transformation of cubic standard triangle Ω to a general triangle E

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

Examine the standard triangle Ω in Figure 32 with the values of a function $f(\xi,\nu)$ at the corners and at the points on the edges. Use the numbering as shown in Figure 32. The parameters ξ and ν at the nodes are given by Table 6. 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[10]

[10] 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 6: Coordinates of the nodes in the standard cubic triangle

$$\vec{\Phi}(\xi,\nu) = \begin{pmatrix} \Phi_1(\xi,\nu) \\ \Phi_2(\xi,\nu) \\ \Phi_3(\xi,\nu) \\ \Phi_4(\xi,\nu) \\ \Phi_5(\xi,\nu) \\ \Phi_6(\xi,\nu) \\ \Phi_7(\xi,\nu) \\ \Phi_8(\xi,\nu) \\ \Phi_9(\xi,\nu) \\ \Phi_{10}(\xi,\nu) \end{pmatrix} = \begin{pmatrix} (1-(\xi+\nu))\,(1-3\,(\xi+\nu))\,(1-\frac{3}{2}\,(\xi+\nu)) \\ \xi\,(3\,\xi-1)\,(\frac{3}{2}\,\xi-1) \\ \nu\,(3\,\nu-1)\,(\frac{3}{2}\,\nu-1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\xi-1) \\ \frac{9}{2}\,\xi\,\nu\,(3\,\nu-1) \\ \frac{9}{2}\,\nu\,(1-(\xi+\nu))\,(3\,\nu-1) \\ 9\,\nu\,(1-(\xi+\nu))\,(1-\frac{3}{2}\,(\xi+\nu)) \\ 9\,\xi\,(1-\frac{3}{2}\,(\xi+\nu))\,(1-(\xi+\nu)) \\ \frac{9}{2}\,\xi\,(3\,\xi-1)\,(1-(\xi+\nu)) \\ 27\,\xi\,\nu\,(1-(\xi+\nu)) \end{pmatrix} \tag{20}$$

$$= \begin{pmatrix} 1-\frac{11}{2}\,\xi-\frac{11}{2}\,\nu+9\,\xi^2+18\,\xi\,\nu+9\,\nu^2-\frac{9}{2}\,\xi^3-\frac{27}{2}\,\xi^2\,\nu-\frac{27}{2}\,\xi\,\nu^2-\frac{9}{2}\,\nu^3 \\ \xi-\frac{9}{2}\,\xi^2+\frac{9}{2}\,\xi^3 \\ \nu-\frac{9}{2}\,\nu^2+\frac{9}{2}\,\nu^3 \\ -\frac{9}{2}\,\xi\,\nu+\frac{27}{2}\,\xi^2\,\nu \\ -\frac{9}{2}\,\xi\,\nu+\frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\nu+\frac{9}{2}\,\xi\,\nu+18\,\nu^2-\frac{27}{2}\,\xi\,\nu^2-\frac{27}{2}\,\nu^3 \\ 9\,\nu-\frac{45}{2}\,\xi\,\nu-\frac{45}{2}\,\nu^2+\frac{27}{2}\,\xi^2\,\nu+27\,\xi\,\nu^2+\frac{27}{2}\,\nu^3 \\ 9\,\xi-\frac{45}{2}\,\xi^2-\frac{45}{2}\,\xi\,\nu+\frac{27}{2}\,\xi^3+27\,\xi^2\,\nu+\frac{27}{2}\,\xi\,\nu^2 \\ -\frac{9}{2}\,\xi+18\,\xi^2+\frac{9}{2}\,\xi\,\nu-\frac{27}{2}\,\xi^3-\frac{27}{2}\,\xi^2\,\nu \\ 27\,\xi\,\nu-27\,\xi^2\,\nu-27\,\xi\,\nu^2 \end{pmatrix} \tag{21}$$

and find their graphs in Figure 33.

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

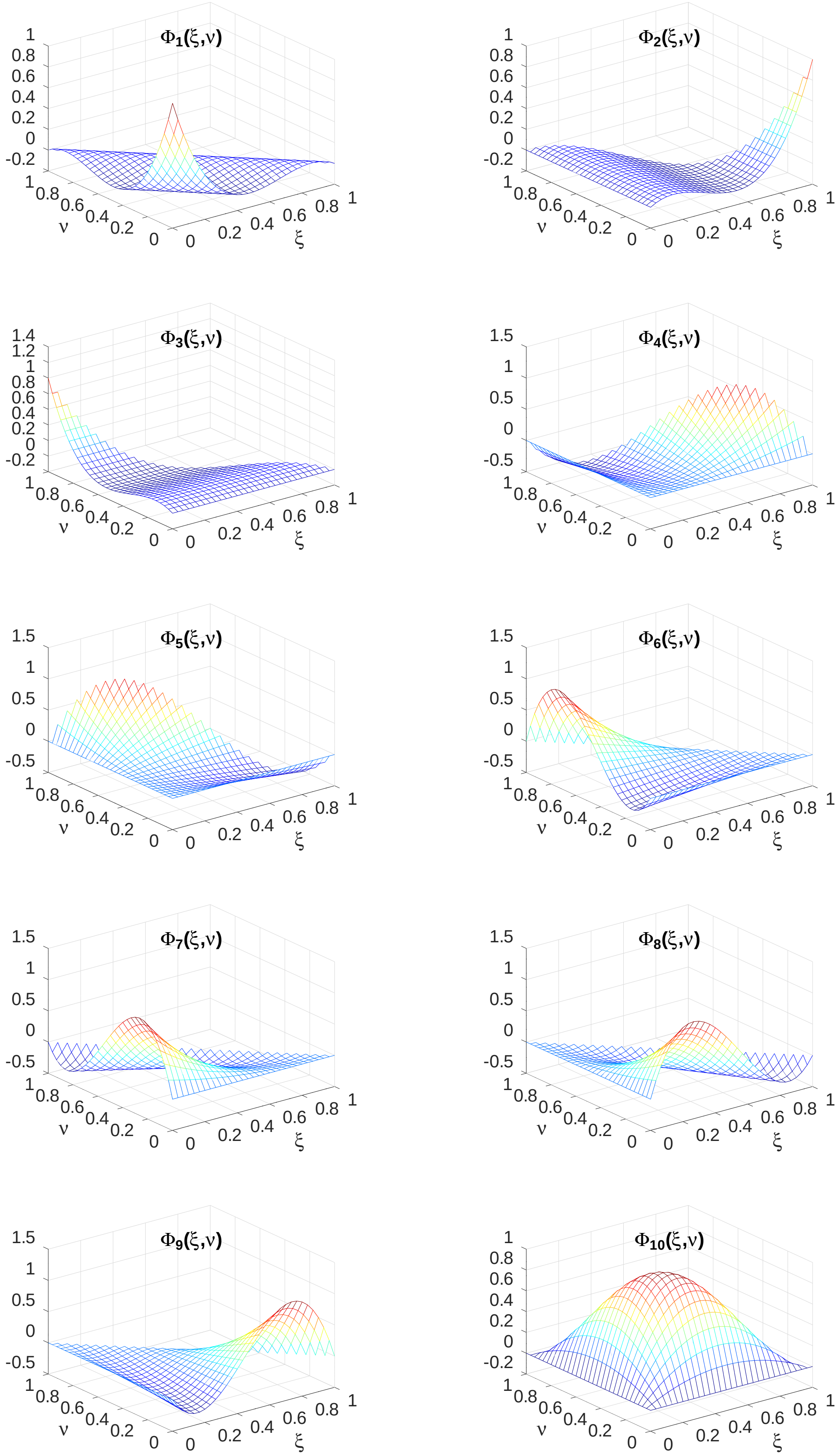
$$f(\xi,\nu) = \sum_{i=1}^{10} f(\xi_i,\nu_i)\,\Phi_i(\xi,\nu) = \sum_{i=1}^{10} f_i\,\Phi_i(\xi,\nu)\,. \tag{22}$$

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 32. 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 29-9-22

Figure 33: The 10 basis functions for third order triangular elements

6.6.2 Determine values at the Gauss points and apply Gauss integration

Use equation (10) (page 56) 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

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

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

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 (8), i.e.

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

Now all the tools to approximate the integrals required for the element stiffness matrix are available.

6.6.3 Integration of $f\,\phi$ and $b_0\,u\,\phi$

These integrations are identical to the case of quadratic elements The test function ϕ is given by its values $\vec{\phi}$ at the nodes, i.e. the corners of the triangle and the two points on each side.

- If the values of the function f at the Gauss points $\vec{g}_i$ are denoted by f_i then this integral is approximated by

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

Thus find one contribution to (19).

- 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

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

- Since the values of the functions u and ϕ are known at the nodes use an interpolation and then the function $b_0(x,y)$ at the Gauss nodes to find

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

The matrices $\mathbf{M}^T\,\mathrm{diag}(\vec{w})$ and $\mathbf{M}^T\,\mathrm{diag}(\vec{w})\,\mathbf{M}$ are again independent on the triangle E, but different from the case of quadratic elements.

6.6.4 Transformation of the gradient to the standard triangle

Computing the partial derivatives is again very similar to the case of quadratic elements. If a function $f(x,y)$ is given on the general triangle E can pull it back to the standard triangle by

$$g(\xi,\nu) = f(x(\xi,\nu)\,,\,y(\xi,\nu))$$

and then compute the gradient of $g(\xi,\nu)$ with respect to its independent variables ξ and ν. The result is This can be written with the help of matrices in the form

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

or equivalently

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

or

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

Let u be a function on the standard triangle Ω given as a linear combination of the basis functions, i.e. $u(\xi,\nu) = \sum_{i=1}^{10} u_i\;\Phi_i(\xi,\nu)$, where the basis function $\Phi_i(\xi,\nu)$ are given by (21). Then its gradient with respect to ξ and ν can be determined with the help of elementary partial derivatives applied to the expressions in (21).

The results are

$$\vec{\Phi}_\xi(\xi,\nu) = \frac{\partial}{\partial \xi}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 1 - 9\,\xi + \frac{27}{2}\,\xi^2 \\ 0 \\ -\frac{9}{2}\,\nu + 27\,\xi\,\nu \\ -\frac{9}{2}\,\nu + \frac{27}{2}\,\nu^2 \\ \frac{9}{2}\,\nu - \frac{27}{2}\,\nu^2 \\ -\frac{45}{2}\,\nu + 27\,\xi\,\nu + 27\,\nu^2 \\ 9 - 45\,\xi - \frac{45}{2}\,\nu + \frac{81}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2} + 36\,\xi + \frac{9}{2}\,\nu - \frac{81}{2}\,\xi^2 - 27\,\xi\,\nu \\ 27\,\nu - 54\,\xi\,\nu - 27\,\nu^2 \end{pmatrix} \tag{23}$$

and

$$\vec{\Phi}_\nu(\xi,\nu) = \frac{\partial}{\partial \nu}\vec{\Phi}(\xi,\nu) = \begin{pmatrix} -\frac{11}{2} + 18\,\xi + 18\,\nu - \frac{27}{2}\,\xi^2 - 27\,\xi\,\nu - \frac{27}{2}\,\nu^2 \\ 0 \\ 1 - 9\,\nu + \frac{27}{2}\,\nu^2 \\ -\frac{9}{2}\,\xi + \frac{27}{2}\,\xi^2 \\ -\frac{9}{2}\,\xi + 27\,\xi\,\nu \\ -\frac{9}{2} + \frac{9}{2}\,\xi + 36\,\nu - 27\,\xi\,\nu - \frac{81}{2}\,\nu^2 \\ 9 - \frac{45}{2}\,\xi - 45\,\nu + \frac{27}{2}\,\xi^2 + 54\,\xi\,\nu + \frac{81}{2}\,\nu^2 \\ -\frac{45}{2}\,\xi + 27\,\xi^2 + 27\,\xi\,\nu \\ +\frac{9}{2}\,\xi - \frac{27}{2}\,\xi^2 \\ 27\,\xi - 27\xi^2 - 54\,\xi\,\nu \end{pmatrix} . \tag{24}$$

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 (21) 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 (23) and (24), define the interpolation matrices for the partial derivatives.

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

Approximate values are

$$\mathbf{M}_\xi \approx \begin{bmatrix} -2.408 & 0.227 & 0 & -0.179 & -0.317 & 0.317 & -1.725 & 3.271 & -1.090 & 1.904 \\ -0.227 & 2.408 & 0 & 1.725 & -0.317 & 0.317 & 0.179 & 1.090 & -3.271 & -1.904 \\ -0.227 & 0.227 & 0 & -1.408 & 4.996 & -4.996 & 1.408 & -0.138 & 0.138 & 0 \\ -0.511 & -0.247 & 0 & 3.852 & 0.868 & -0.868 & 1.358 & 1.137 & -0.379 & -5.210 \\ 0.247 & 0.511 & 0 & -1.358 & 0.868 & -0.868 & -3.852 & 0.379 & -1.137 & 5.210 \\ 0.247 & -0.247 & 0 & 0.489 & -0.221 & 0.221 & -0.489 & -2.984 & 2.984 & 0 \\ 0.500 & -0.500 & 0 & 1.500 & 0 & 0 & -1.500 & -1.500 & 1.500 & 0 \end{bmatrix}$$

and

$$\mathbf{M}_\nu \approx \begin{bmatrix} -2.269 & 0 & 0.227 & -0.317 & -0.179 & -1.090 & 3.271 & -1.725 & 0.317 & 1.904 \\ 0.863 & 0 & 0.227 & 4.996 & -1.408 & 0.138 & -0.138 & 1.408 & -4.996 & 0 \\ 0.863 & 0 & 2.408 & -0.317 & 1.725 & -3.271 & 1.090 & 0.179 & 0.317 & -1.904 \\ 2.473 & 0 & -0.247 & 0.868 & 3.852 & -0.379 & 1.137 & 1.358 & -0.868 & -5.210 \\ 0.626 & 0 & -0.247 & -0.221 & 0.489 & 2.984 & -2.984 & -0.489 & 0.221 & 0 \\ 0.626 & 0 & 0.511 & 0.868 & -1.358 & -1.137 & 0.379 & -3.852 & -0.868 & 5.210 \\ 2.000 & 0 & -0.500 & 0 & 1.500 & 1.500 & -1.500 & -1.500 & 0 & 0 \end{bmatrix} .$$

The matrices $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ allow to compute the values of the partial derivatives at the Gauss points in the standard triangle Ω and they are independent on the general triangle E.

Combining the above two computations use the notation

$$\vec{x}_i = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi_i \\ \nu_i \end{pmatrix} \quad \text{for} \quad i = 1, 2, 3, \ldots, 7$$

and find for the first component $\varphi_x = \frac{\partial \varphi}{\partial x}$ of the gradient at the Gauss points

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

and for the second component of the gradient

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

The above results for $\mathbf{M}_\xi$ and $\mathbf{M}_\nu$ can be coded in *Octave* and then used to compute the element stiffness matrix.

6.6.5 Integration of $u\,\vec{b} \cdot \nabla\phi$ and $a\,\nabla u \cdot \nabla\phi$

The vector function $\vec{b}(\vec{x})$ has to be evaluated at the Gauss integration points $\vec{g}_j$. Then the integration of

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

is approximated by

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

The function $a\,\nabla u \cdot \nabla\phi = a\,(\frac{\partial\,u}{\partial x}\frac{\partial\,\phi}{\partial x} + \frac{\partial\,u}{\partial y}\frac{\partial\,\phi}{\partial y})$ has to be evaluated at the Gauss integration points $\vec{g}_j$, then multiplied by the Gauss weights w_i and added up. Use the vector $\overrightarrow{wa}$ with the values of the function $a(x_i, y_i)$ and the weights w_i at the Gauss points to obtain

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

where

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

6.6.6 Partial derivatives at the nodes

For post processing one also needs the partial derivatives of the function at the nodes. On the standard triangle Ω use the formulas for the partial derivatives of the basis functions in expressions (23) and (24) to find them at the nodes, given by the (ξ, ν) coordinates in Table 6 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 (17) and (18) 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 (7) 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

$$\int_{-h/2}^{h/2} f(x)\,dx \approx \frac{h}{18}\left(5\,f(-\frac{\sqrt{3}}{2\sqrt{5}}\,h) + 8\,f(0) + 5\,f(\frac{\sqrt{3}}{2\sqrt{5}}\,h)\right) = \frac{h}{18}\left(5\,f(-\frac{\sqrt{15}}{10}\,h) + 8\,f(0) + 5\,f(\frac{\sqrt{15}}{10}\,h)\right)$$

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

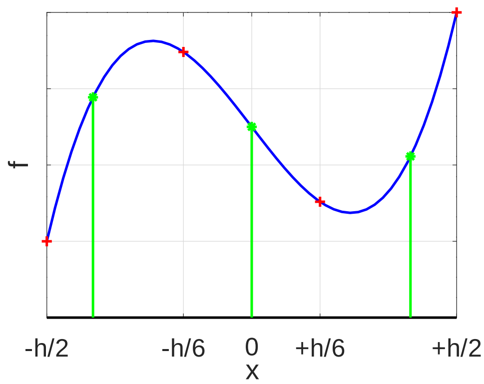


Figure 34: The interpolation from four nodes to three Gauss points on an interval $[-\frac{h}{2}, +\frac{h}{2}]$

computations are tedious[11] and lead to

$$\begin{pmatrix} u(\vec{p}_1) \\ u(\vec{p}_2) \\ u(\vec{p}_3) \end{pmatrix} = \mathbf{M}_B \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix} \approx \begin{bmatrix} 0.4880 & 0.7479 & -0.2979 & 0.06199 \\ -0.0625 & 0.5625 & 0.5625 & -0.0625 \\ 0.06199 & -0.2979 & 0.7479 & 0.4880 \end{bmatrix} \begin{pmatrix} f_{-2} \\ f_{-1} \\ f_{+1} \\ f_{+2} \end{pmatrix}$$

[11]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 the second and 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 converge 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

$$\begin{aligned} \|u\|_2^2 &= \iint_\Omega u^2(x,y)\,dA \\ \|u\|_V^2 &= \iint_\Omega u^2(x,y) + \|\nabla u(x,y)\|^2\,dA\,. \end{aligned}$$

The 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 7.4.

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.

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

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$. Solve the static problem with nonhomogeneous boundary conditions.

$$\begin{aligned}
-\nabla\cdot(a\,\nabla u_B - u_B\,\vec{b}) + b_0\,u_B &= 0 && \text{for} \quad (x,y,t)\in\Omega \\
u_B &= g_1 && \text{for} \quad (x,y)\in\Gamma_1 \\
\vec{n}\cdot(a\,\nabla u_B + u_B\,\vec{b}) &= g_2 + g_3\,u_B && \text{for} \quad (x,y,t)\in\Gamma_2
\end{aligned} \tag{25}$$

Then the new function $v(x,y,t) = u(x,y,t) - u_B(x,y)$ is a solution of an initial boundary value problem with no constant boundary contributions, i.e. $g_1 = g_2 = 0$.

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

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

$$\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{v}_0\,. \tag{26}$$

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[12] 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

[12] 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.

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

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!

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

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

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

6.8.2 Using eigenvalues for dynamic problems of the heat equation type

With equation (26) 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 $u(t) = c\,\exp(-\lambda\,t)\,\vec{v}$, 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.

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

First the problem is reduced to a new problem with homogeneous boundary conditions, i.e $g_1 = g_2 = 0$, using (25). 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. $g_1 = g_2 = 0$.

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

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

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t) + 2\,\mathbf{D}\,\frac{d}{dt}\,\vec{v}(t) + \mathbf{A}\,\vec{v}(t) = \vec{f}(t) \quad \text{with} \quad \vec{v}(0) = \vec{u}_0 - \vec{u}_B\,, \quad \frac{d}{dt}\,\vec{v}(0) = \vec{v}_0 \tag{27}$$

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

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

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[14].

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

[14] 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 (27) for $\vec{f}=\vec{0}$ and a damping factor $D\,\mathbf{W}$ with a constant $D\geq 0$ (instead of the matrix $\mathbf{D}$)

$$\mathbf{W}\,\frac{d^2}{dt^2}\,\vec{v}(t)+2\,D\,\mathbf{W}\,\frac{d}{dt}\,\vec{v}(t)+\mathbf{A}\,\vec{v}(t)=\vec{0} \tag{28}$$

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 $u(t)=\exp(\mu\,t)\,\vec{v}$ 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

  $$u(t) = c_1\,\exp(-D + \sqrt{D^2 - \lambda}\,t)\,\vec{v}_1 + c_2\,\exp(-D - \sqrt{D^2 - \lambda}\,t)\,\vec{v}_2\,.$$

- 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 (28) 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

$$\min_{\lambda_j \in \sigma(\mathbf{A})} |\beta_i - \lambda_j| \leq \sqrt{\langle \vec{r}\,,\, \mathbf{B}^{-1}\vec{r} \rangle} \quad \text{and} \quad |\beta_i - \lambda_j| \leq \frac{\langle \vec{r}\,,\, \mathbf{B}^{-1}\vec{r} \rangle}{\text{gap}} \tag{29}$$

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 Examples, Examples, Examples

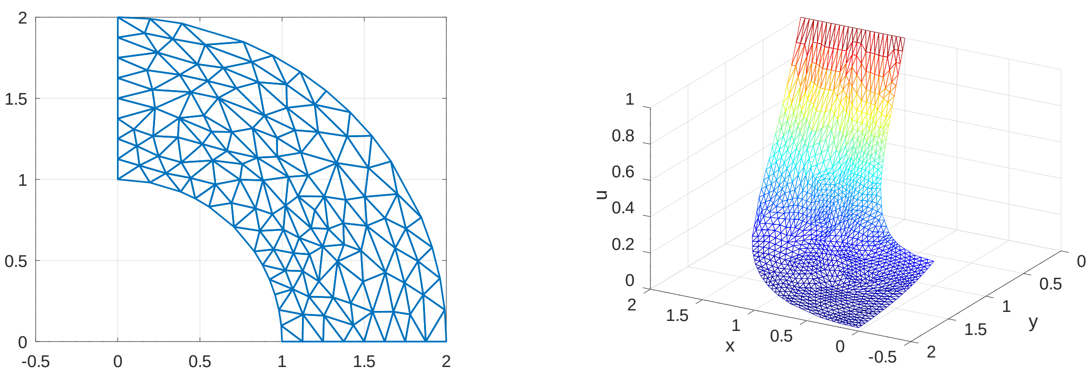
7.1 An elliptic problem with variable coefficients

The elliptic BVP in Section 5.5 is

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

on the domain shown in Figure 35(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 $1 \leq r \leq 2$ and $0 \leq \varphi \leq \pi/2$.
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 35(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 35(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 35: 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
```

7.2 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 36 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
```

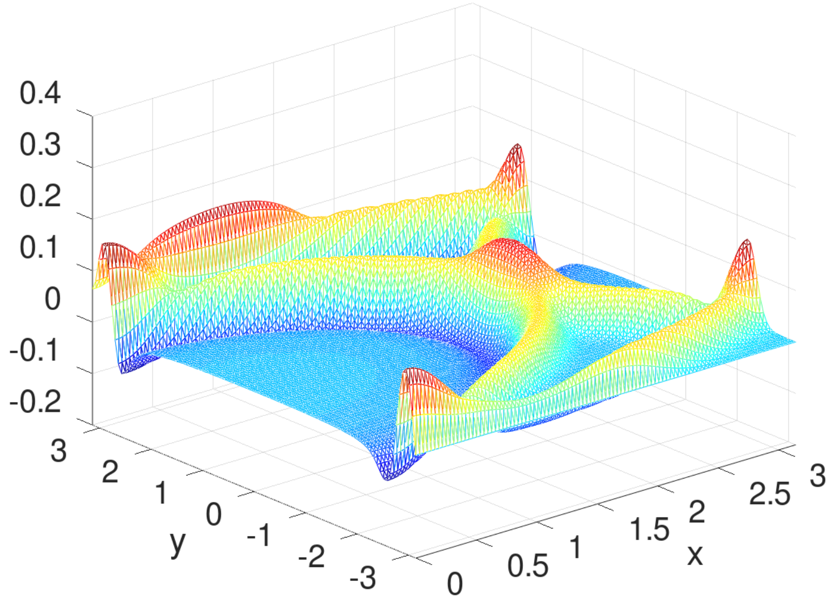


Figure 36: Traveling waves on a rectangle

7.3 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

$$\begin{array}{rcll} -\Delta u + u & = & 2\,f & \text{for} \quad 0 < x, y < 1 \\ u & = & f & \text{for } (1,y) \text{ and } (x,1) \\ \frac{\partial\, u}{\partial n} & = & 0 & \text{for } (0,y) \text{ and } (x,0) \end{array} \quad .$$

A solution is shown in Figure 37. This BVP is solved by two slightly different approaches, and then the difference to the known exact solution displayed in Figure 38. 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 38(a) shows the effect of super-convergence. Caused by the extremely regular structure of the grid points the differences are smaller than can reasonably be expected.

2. Use a non-uniform mesh generated by `CreateMeshTriangle`, leading to 432 degrees of freedom. Thus one expects to obtain similar accuracy. The result in Figure 38(b) confirms this.

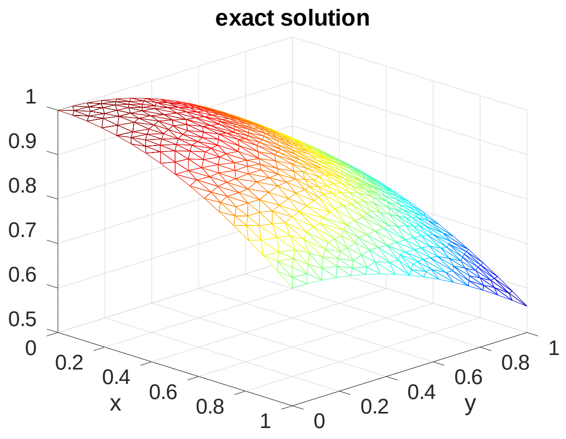
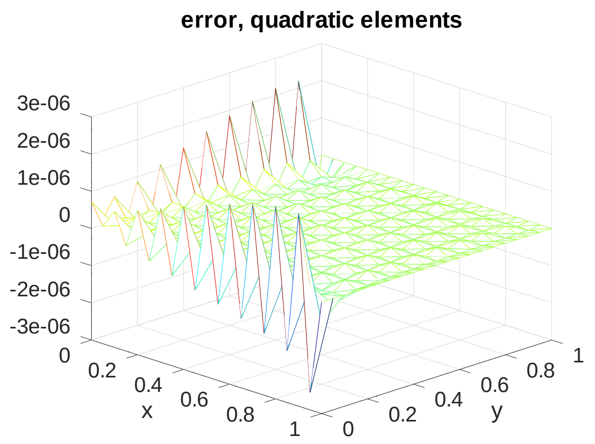
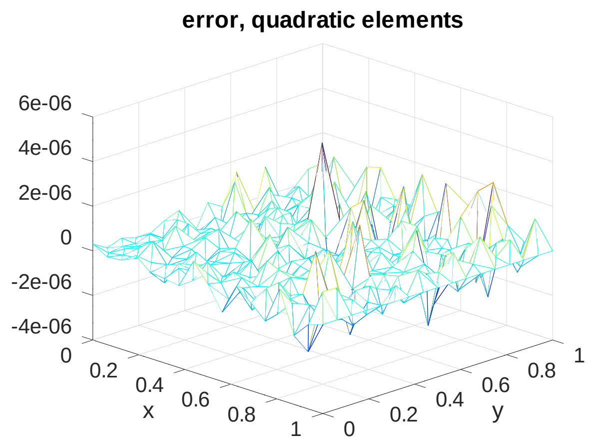


Figure 37: The radial Bessel function as solution of a BVP

```
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);
  FEMmesh1 = MeshQuad2Linear(FEMmesh);
  nDOFRect = [FEMmesh.nDOF, FEMmesh1.nDOF]
endif
```



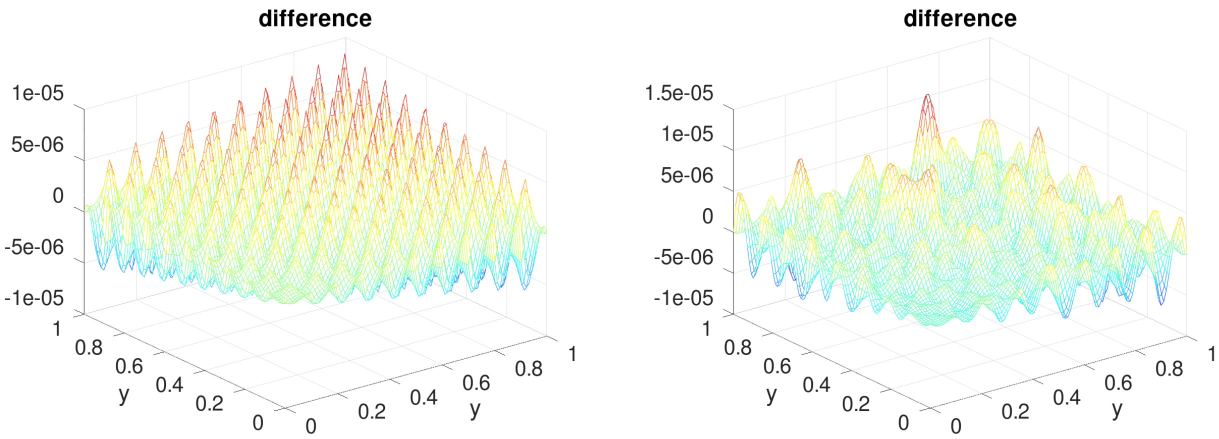
(a) uniform grid



(b) nonumiform grid

Figure 38: Difference to the exact solution of a BVP

To generate Figure 39 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.



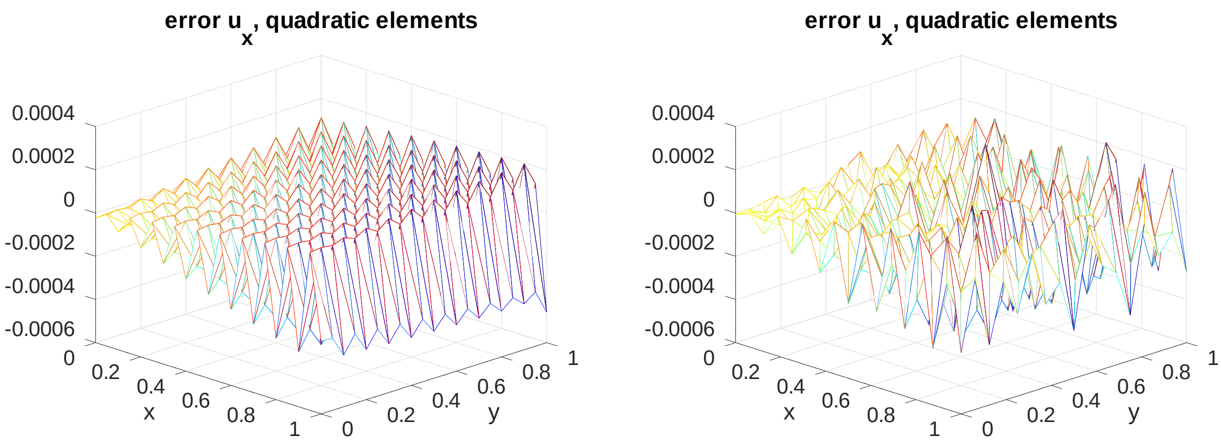
(a) uniform grid (b) nonumiform grid

Figure 39: Difference to the exact solution of a BVP, using quadratic elements and interpolation to a finer grid.

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

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

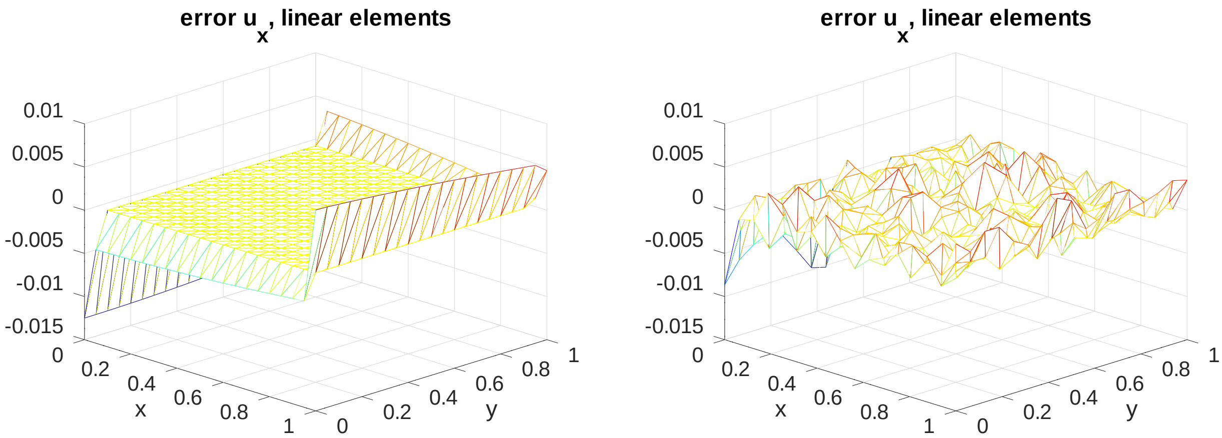
Using the above FEM results compare the true partial derivative $\frac{\partial u}{\partial x}$ with the one obtained by FEM with second order elements. Find the result in Figure 40. Observe the structure of the difference for the uniform mesh.



(a) uniform grid (b) nonumiform grid

Figure 40: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using second order elements

The above can be repeated using first order elements, leading to Figure 41. The size of the elements was set such that the same number of degrees of freedom are used. Observe that superconvergence strikes again. In this case I have a solid argument for the structural difference along the border.



(a) uniform grid (b) nonumiform grid

Figure 41: Difference of $\frac{\partial u}{\partial x}$ to the exact solution, using first order elements

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

7.4 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

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

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 42. This example illustrates that non-convex domains with sharp corners might cause convergence problems.

SingularDisc.m

```
x_p = [0;1;1;-1;-1;0]; y_p = [0;0;1;1;-1;-1];

FEMmesh = CreateMeshTriangle("circle34",[x_p,y_p,-ones(size(x_p))], 0.01);
FEMmesh = MeshUpgrade(FEMmesh);

function res = gD(xy)
  phi = mod(atan2(xy(:,2),xy(:,1)),2*pi);
  res = (xy(:,1).^2+ xy(:,2).^2).^(1/3).*sin(2/3*phi);
endfunction

u = BVP2Dsym(FEMmesh,1,0,0,'gD',0,0);
figure(1); FEMtrimesh(FEMmesh,u);
           xlabel("x"); ylabel("y"); title('FEM solution'); view([30,30])

u_exact = gD(FEMmesh.nodes);
figure(2); FEMtrimesh(FEMmesh,-u+u_exact);
xlabel("x"); ylabel("y"); title('Error of FEM solution'); view([30,30])
```

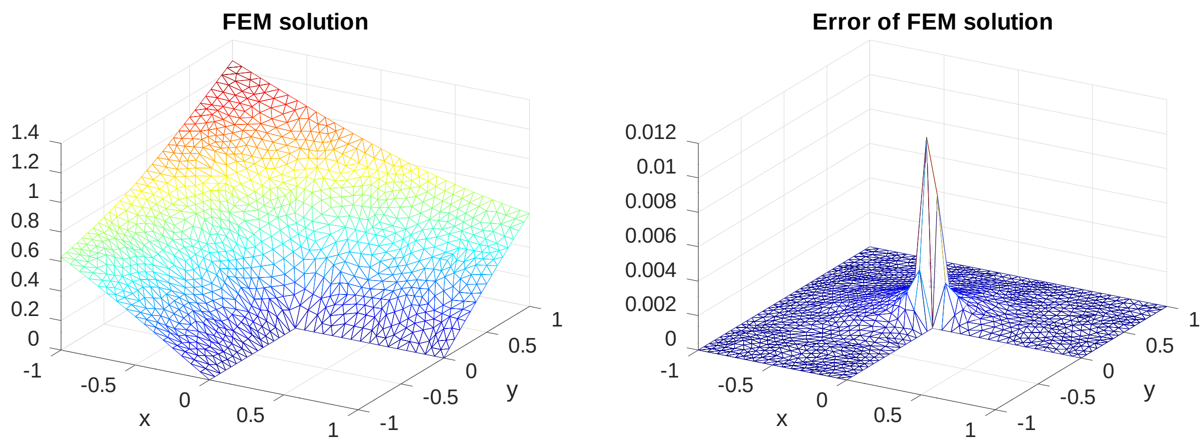


Figure 42: A solution with singular partial derivatives at the origin

The gradient in Cartesian coordinates can be determined by

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

and then visualized, leading to Figure 43. 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])
```

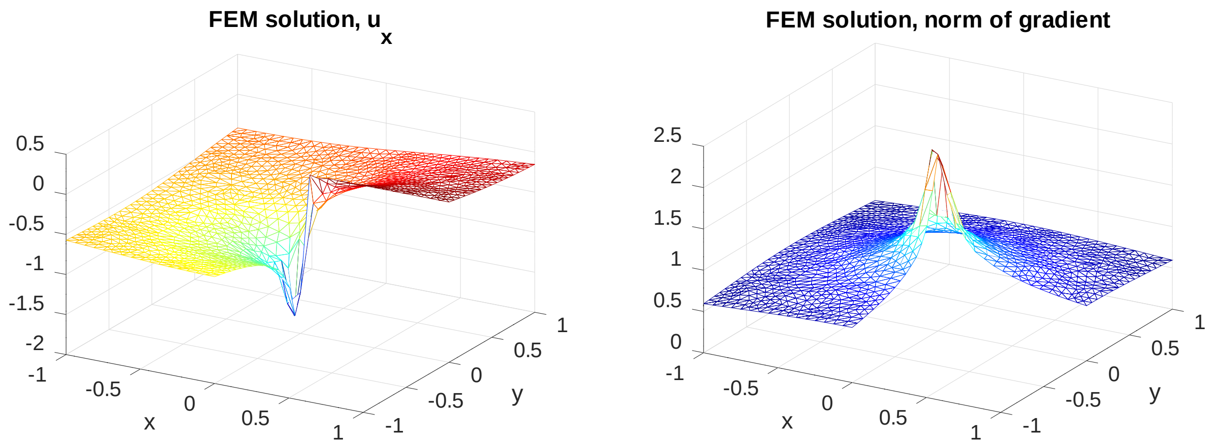


Figure 43: A solution with singular partial derivatives, graphs of $\frac{\partial u}{\partial x}$ and $\|\nabla u\|$

7.5 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 44. The goal is to find the velocity field $\vec{v}$ of the fluid.

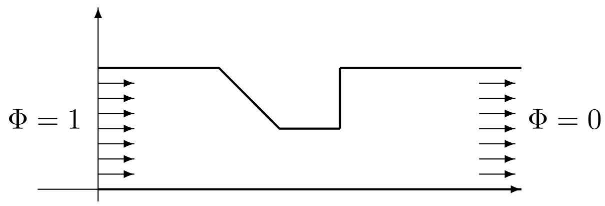


Figure 44: Fluid flow between two plates, the setup

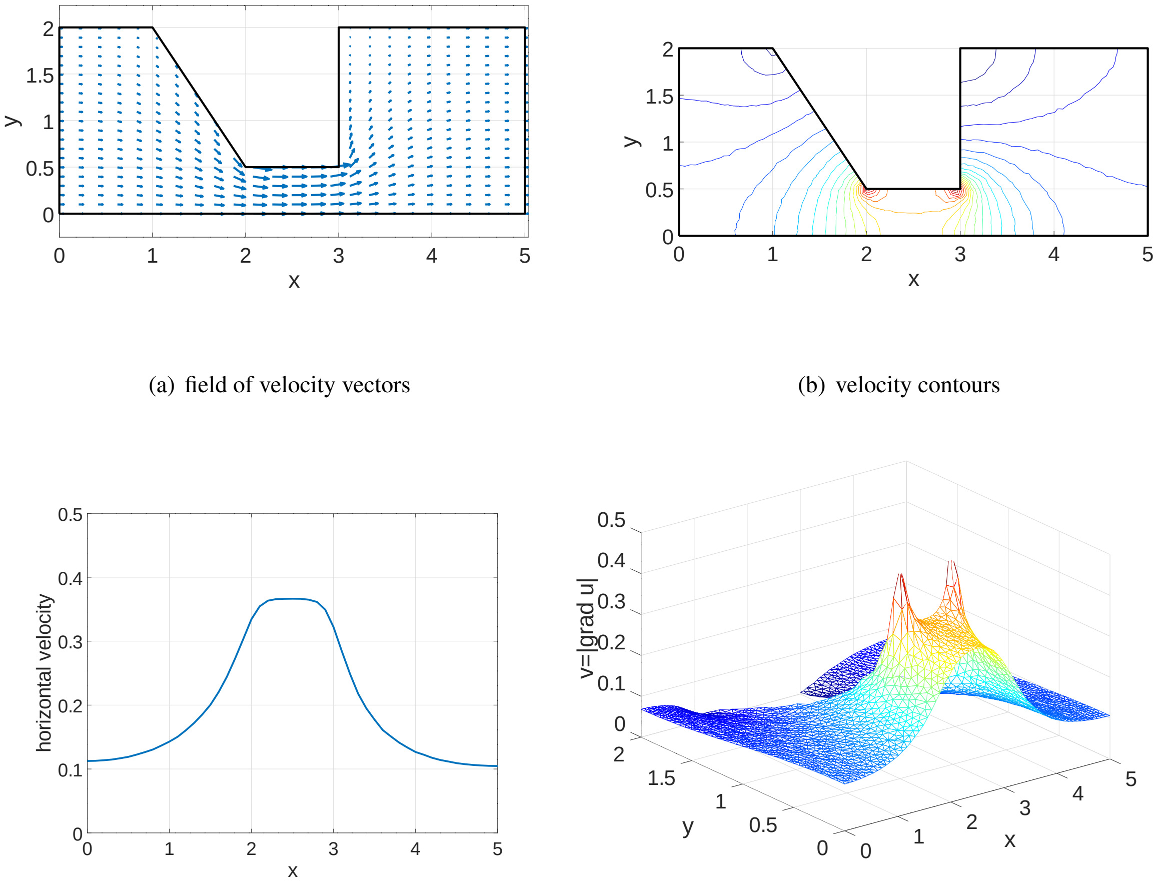
This problem is solved by introducing a velocity potential $\Phi(x, y)$. The velocity vector $\vec{v}$ is then given by

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = -\begin{pmatrix} \frac{\partial \Phi}{\partial x} \\ \frac{\partial \Phi}{\partial y} \end{pmatrix} .$$

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 45 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 7.4.



(a) field of velocity vectors

(b) velocity contours

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

(d) the velocity

Figure 45: Velocity field of a ideal fluid

The results are generated by the code below.

PotentialFlow.m

```
%% define the domain
xy = [0 0 -2; 5 0 -1;5 2 -2; 3 2 -2; 3 0.5 -2; 2 0.5 -2; 1 2 -2; 0 2 -1];
if 1      %% linear elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,0.003);
elseif 1  %% quadratic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,4*0.003);
  FEMmesh = MeshUpgrade(FEMmesh,'quadratic');
else      %% cubic elements
  FEMmesh = CreateMeshTriangle('PotentialFlow',xy,9*0.003);
  FEMmesh = MeshUpgrade(FEMmesh,'cubic');
endif
```

```
x = FEMmesh.nodes(:,1);  y = FEMmesh.nodes(:,2);
function res = gD(xy)   res = 1-xy(:,1)/5;  endfunction
u = BVP2Dsym(FEMmesh,1,0,0,'gD',0,0);
figure(1); FEMtrimesh(FEMmesh,u)
           xlabel('x'); ylabel('y'); zlabel('potential')

[xx,yy] = meshgrid(linspace(0,5-0.01,25),linspace(0,2-0.01,21));
[u_int,ux_int,uy_int] = FEMgriddata(FEMmesh,-u, xx, yy);

figure(2); quiver(xx,yy,ux_int,uy_int)
           xlabel('x'); ylabel('y');
           hold on; plot([xy(:,1);0],[xy(:,2);0],'k'); hold off; axis equal

xx = linspace(0,5,101); yy = 0.25*ones(101,1);
[u_int,ux_int,uy_int] = FEMgriddata(FEMmesh,-u,xx,yy);
figure(3); plot(xx,ux_int)
           xlabel('x'); ylabel('horizontal velocity'); ylim([0 0.5])

[ux,uy] = FEMEvaluateGradient(FEMmesh,u);
figure(4); FEMtrimesh(FEMmesh,sqrt(ux.^2+ uy.^2))
           xlabel('x'); ylabel('y'); zlabel('v=|grad u|'); zlim([0 0.5])


figure(5); FEMtricontour(FEMmesh,sqrt(ux.^2+ uy.^2),21)
           xlabel('x'); ylabel('y'); zlabel('| grad u|')
           hold on; plot([xy(:,1);0],[xy(:,2);0],'k'); hold off
           xlim([0 5]); ylim([0 2]); axis equal
```

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)
```

7.6 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

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

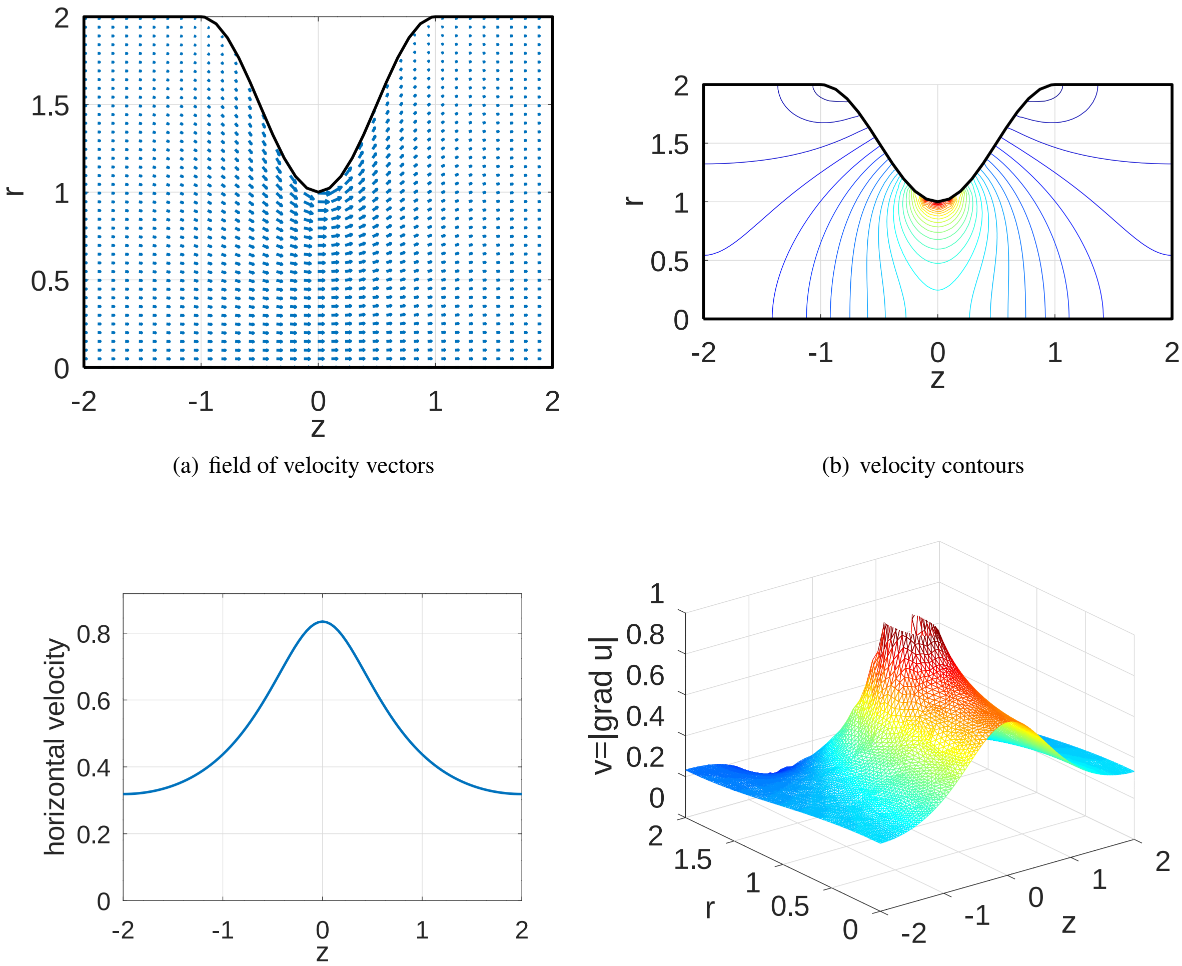
The upper half of a section is visible in Figure 46. Assuming that the solution is independent on the angle θ the equation $\Delta\Phi = 0$ has to be reformulated in cylindrical coordinates and simplified.

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

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 7.5, 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 46: 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.

7.7 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.

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

This software is not directly capable of solving non linear problems, but a simple iteration will lead to an

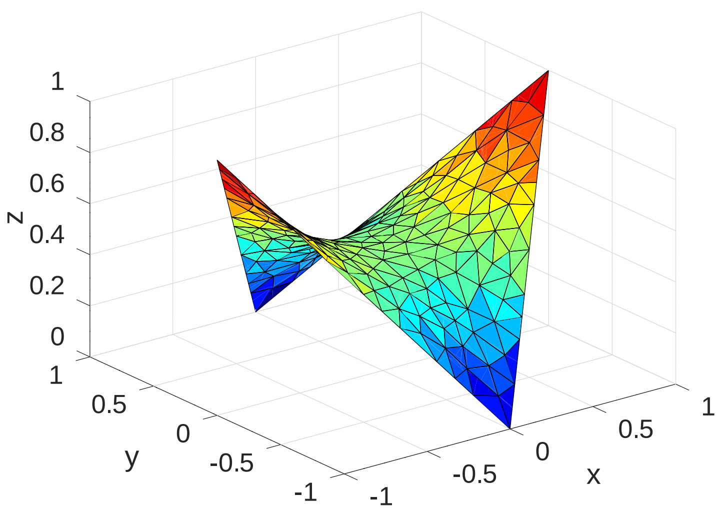


Figure 47: A minimal surface

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

The code below implements this algorithm for a square Ω and leads to the result in Figure 47. 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 the the computed minimal area will be smaller. This should not come as a surprise, the better the resolution, the smaller the minimal area.

7.8 Computing a capacitance

7.8.1 State the problem

Examine a circular plate capacitance as shown in Figure 48. Based on the radial symmetry one should be able to consider a two dimensional section only for the computations.

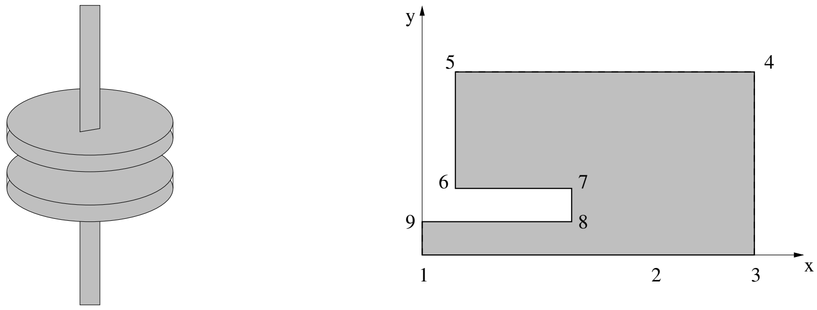


Figure 48: The capacitance and the section used for the modeling

Consider the voltage u as unknown. On the upper conductor assume $u = 1$ and on the lower conductor $u = -1$. Based on the symmetry consider a section only and use $u = 0$ in the plane centered between the conductors. Use the Laplace operator in cylindrical coordinates. Thus the following boundary value problem has to be solved.

$$\begin{array}{rcll} \operatorname{div}(x \ \operatorname{grad} u(x,y)) & = & 0 & \text{in domain} \\ u(x,0) & = & 0 & \text{along edge } y = 0 \\ u(x,y) & = & 1 & \text{along edges of upper conductor} \\ \frac{\partial}{\partial n} u(x,y) & = & 0 & \text{on remaining boundary} \end{array} \tag{30}$$

Assume that the domain is embedded in the rectangle $0 \leq x \leq R$ and $0 \leq y \leq H$. The lower edge of the conductor is at $y = h$ and $0 \leq x \leq r$. If $h \ll r$ expect the gradient of u to be $1/h$ between the plates and zero away from the plates. Thus

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

Because the electric field will not be homogeneous around the boundaries of the disk expect deviations from the result of an idealized circular disk. With the divergence theorem and a physical argument one can verify that the flux trough the midplane is proportional to the capacitance. By applying the following steps compute the capacitance by analyzing the solution of a boundary value problem.

1. Create a mesh for the domain in question.
2. Define parameters and boundary conditions.
3. Solve the partial differential equation and visualize the solution.
4. Compute the flux through the midplane as an integral to determine the capacitance.

7.8.2 Create the mesh and solve the BVP

According to Figure 48 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 48
- 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 48. With the above use the program `triangle` to generate a mesh.

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

The mesh consists of 2189 nodes, forming 4036 triangles.

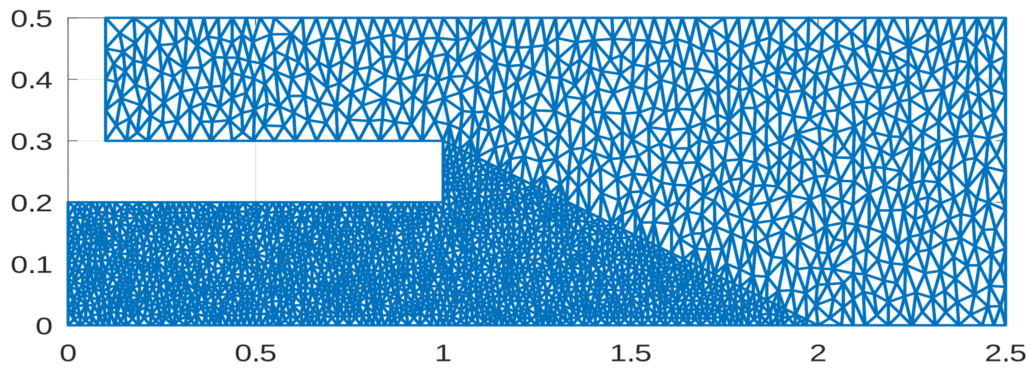


Figure 49: A mesh on the domain

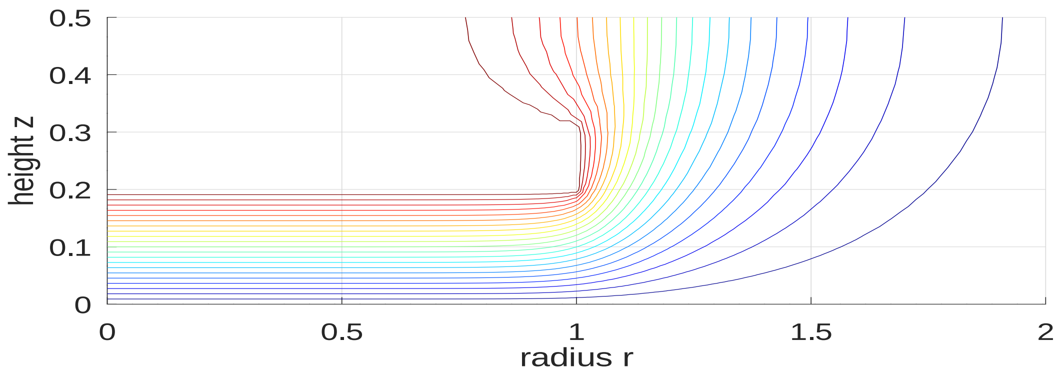
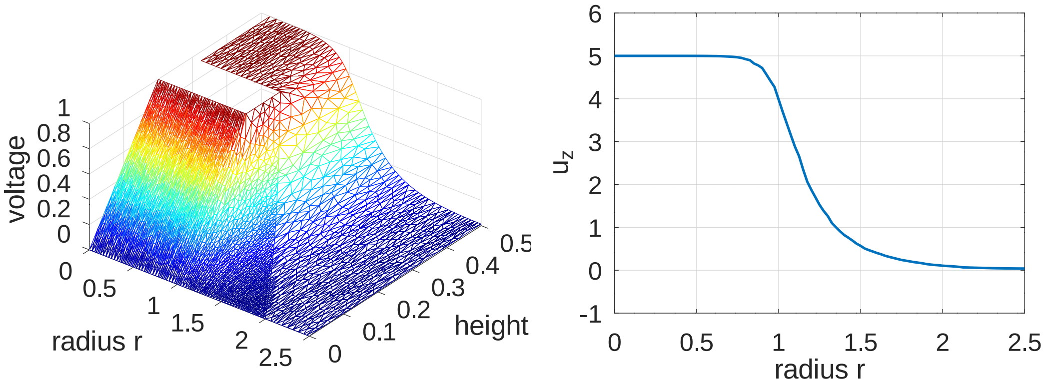


Figure 50: The contour lines of the resulting voltage

To solve the BVP (30) 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 51.



(a) the voltage (b) vertical field along the horizontal axis

Figure 51: Voltage plot and electric field between the plates of the capacitance

7.8.3 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 51. The graph confirms that between the plates the gradient is approximately $1/h = 1/0.2 = 5$ and vanishes away from the plate. Then a trapezoidal rule is used to determine the flux accross the midplane with the integral.

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

For the selected values of h, H, r and R obtain a factor of 1.5 between result of the boundary value problem and the idealized approximation $\pi\, r^2/h$. Thus the simple formula is not a good approximation, the distance h is too large compared to the radius r.

Capacitance.m

```
FEMmesh = ReadMeshTriangle('capacitance.1');
%% FEMmesh = MeshUpgrade(FEMmesh,'quadratic');  %% uncomment fora quadratic mesh
figure(1); FEMtrimesh(FEMmesh)                  %% display the generated mesh

function res = a(xy)      res = xy(:,1);     endfunction
function res = Volt(xy)   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
```

7.9 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].

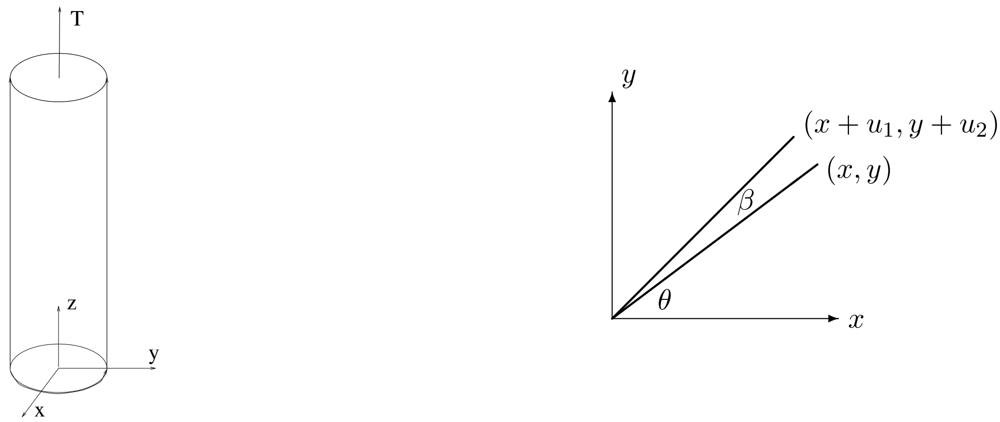


Figure 52: Torsion of a shaft

7.9.1 The setup with warp function and 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 52. 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 52 and a linear approximation

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

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

$$\begin{array}{rcll} \operatorname{div}(\nabla\phi) = \Delta\phi &=& 0 & \text{in the cross section} \quad \Omega \\ \vec{n}\cdot\nabla\phi &=& \begin{pmatrix} y \\ -x \end{pmatrix}\cdot\vec{n} & \text{on the boundary} \quad \partial\Omega \end{array}\,. \tag{31}$$

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[15], 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

$$\begin{array}{rclcl} -\Delta\chi &=& 2 & \text{in} & \Omega\\ \chi &=& 0 & \text{on} & \Gamma \end{array}\,. \tag{32}$$

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

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

7.9.2 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 (31) 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

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

[15] This restriction can be removed.

7.9.3 On a square

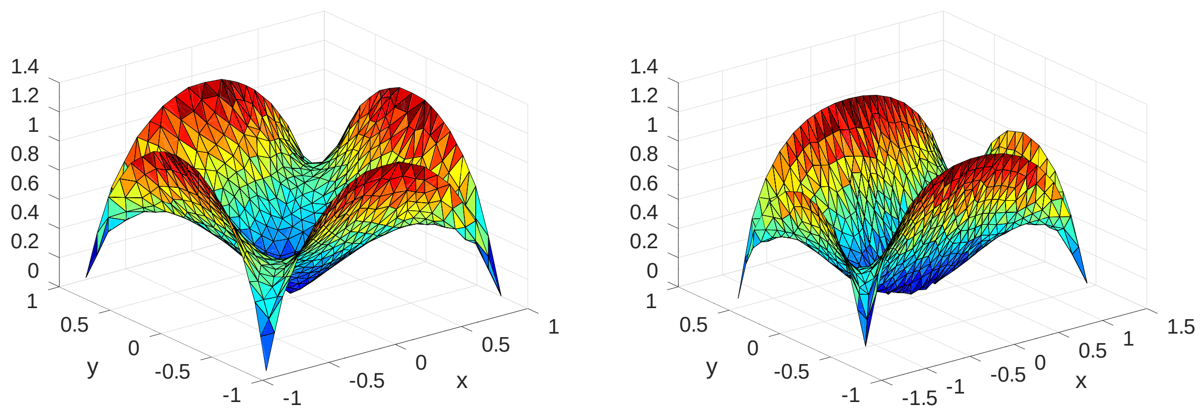
To examine the stiffness of a square cross section with a circular cross section examine a square with the same area as a circle with radius $R = 1$. Thus the length of a side is $\sqrt{\pi} \approx 1.77$. The code below solves the boundary value problem (32) and then computes the torsional rigidity 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 53(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 53: 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
```

7.9.4 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.

Bibliography

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

[Brae02] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, second edition, 2002.

[Ciar02] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. SIAM, 2002.

[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.

[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.

[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