TMA4220: Programming project - part 1

TMA4220 - Numerical solution of partial differential equations using the finite element method

The programming project will be split into two parts. This is the first part and is mandatory for all students. It is an introduction into the finite element method and is designed to build up a solid code base for part 2, where you will actually solve larger, real life problems. For the second part you will have to choose one of several tasks to implement.

1 Gaussian quadrature

At the heart of all finite element codes lies the evaluation of integrals. These may be of varying complexity depending on the problem at hand, and many do not even possess a known analytical solution. Some are *possible* to solve analytically, but of such computational complexity that doing so becomes impractical. Numerical integration schemes are therefore employed to do the core integration. One popular scheme is *Gaussian quadrature*.

In one dimension Gaussian quadrature takes the form

$$\int_{-1}^{1} g(x)dx \approx \sum_{q=1}^{N_q} \rho_q g(z_q),$$

where N_q is the number of integration points, z_q are the Gaussian quadrature points and ρ_q are the associated Gaussian weights, given in Table 1. For integrals over a general interval $[a, b]$, the integral may be transformed by a change of variables to run over the reference interval$[-1, 1]$, or equivalently the nodes z_q must be transformed to lie in $[a, b]$ by the mapping $x \mapsto \frac{(a+b)+x(b-a)}{2}$. The weights ρ_q in Table 1 sum to 2, the length of the interval $[-1, 1]$, these must therefore be scaled by a factor $\frac{b-a}{2}$; equivalently the whole sum may be scaled by this amount instead.

This extends to higher dimensions by

$$\int_K g(\boldsymbol{x})d\boldsymbol{x} \approx |K| \sum_{q=1}^{N_q} \rho_q g(\boldsymbol{z}_q),$$

where $|K|$ is the volume of the element K, and the vector quadrature points $\boldsymbol{z}_q$ are typically given in terms of barycentric coordinates. The volume of the element $|K|$ is computed by $|J|/d!$, where $|J|$ is the Jacobian determinant of the mapping $\hat{x} \mapsto x$ from the reference element K to $\hat{K}$, and $1/d!$ is the volume of the d-dimensional reference element (simplex). Note that for some integrals such as those appearing in the stiffness matrix, it is usually best to map the integral to the reference element $|\hat{K}|$ first, see the lecture notes.

a) 1D quadrature

Write a matlab function `I = quadrature1D(a,b,Nq,g)`. With the following arguments:

`I`	$\in \mathbb{R}$	value of the integral
`a`	$\in \mathbb{R}$	integration start
`b`	$\in \mathbb{R}$	integration end
`Nq`	$\in [1, 4]$	number of integration points
`g`	$: \mathbb{R} \to \mathbb{R}$	function pointer*

N_q	z_q	ρ_q
1-point-rule	0	2
2-point-rule	$-\sqrt{1/3}$	1
	$\sqrt{1/3}$	1
3-point-rule	$-\sqrt{3/5}$	5/9
	0	8/9
	$\sqrt{3/5}$	5/9
4-point-rule	$-\sqrt{\frac{3+2\sqrt{6/5}}{7}}$	$\frac{18-\sqrt{30}}{36}$
	$-\sqrt{\frac{3-2\sqrt{6/5}}{7}}$	$\frac{18+\sqrt{30}}{36}$
	$\sqrt{\frac{3-2\sqrt{6/5}}{7}}$	$\frac{18+\sqrt{30}}{36}$
	$\sqrt{\frac{3+2\sqrt{6/5}}{7}}$	$\frac{18-\sqrt{30}}{36}$

Table 1: 1D gauss quadrature

verify that the function evaluates correctly by comparing with the analytical solution of the integral

$$\int_1^2 e^x \, dx$$

b) 2D quadrature

In higher dimensions, the Gaussian quadrature nodes are often given in barycentric coordinates (or area coordinates as they are known in 2D). Recall that area coordinates are defined by

$$\begin{aligned} \lambda_1 &= \frac{A_1}{A} \\ \lambda_2 &= \frac{A_2}{A} \\ \lambda_3 &= \frac{A_3}{A} \end{aligned}$$

where A_1, A_2 and A_3 are the area of the triangles depicted in figure 1 and A is the total area of the triangle. Note that these do not form a linear independent basis as $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

N_q	$(\lambda_1, \lambda_2, \lambda_3)$	ρ
1-point rule	$(1/3, 1/3, 1/3)$	1
3-point rule	$(1/2, 1/2, 0)$	1/3
	$(1/2, 0, 1/2)$	1/3
	$(0, 1/2, 1/2)$	1/3
4-point rule	$(1/3, 1/3, 1/3)$	-9/16
	$(3/5, 1/5, 1/5)$	25/48
	$(1/5, 3/5, 1/5)$	25/48
	$(1/5, 1/5, 3/5)$	25/48

Table 2: 2D gauss quadrature

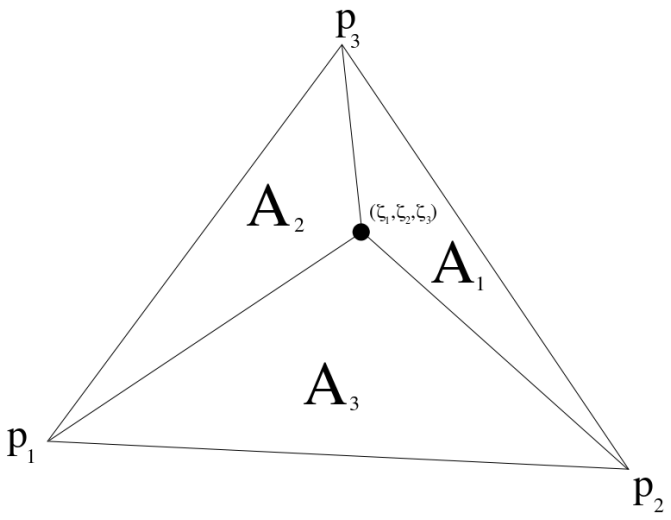

Figure 1: Barycentric coordinates in two dimensions

Write a matlab function `I = quadrature2D(p1,p2,p3,Nq,g)`. With the following arguments:

Hint: An easy way of mapping barycentric coordinates $\boldsymbol{\lambda}$ to physical coordinates $\boldsymbol{x}$ is by $\boldsymbol{x} = \lambda_1 \boldsymbol{p}_1 + \lambda_2 \boldsymbol{p}_2 + \lambda_3 \boldsymbol{p}_3$, where $\boldsymbol{p}_i, i = 1, 2, 3$ are the corner points of the triangle.

`I`	$\in \mathbb{R}$	value of the integral
`p1`	$\in \mathbb{R}^2$	first corner point of the triangle
`p2`	$\in \mathbb{R}^2$	second corner point of the triangle
`p3`	$\in \mathbb{R}^2$	third corner point of the triangle
`Nq`	$\in \{1, 3, 4\}$	number of integration points
`g`	$: \mathbb{R}^2 \to \mathbb{R}$	function pointer*

Verify that the function evaluates correctly by comparing with the analytical solution of the integral

$$\iint_{\Omega} \log(x + y)\, dx\, dy,$$

where Ω is the triangle defined by the corner points $(1, 0)$, $(3, 1)$ and $(3, 2)$.

c) 3D quadrature

The extension of barycentric coordinates to 3 dimensions and tetrahedral elements should be straightforward. The integration schemes can be found in the following table

Write a matlab function `I = quadrature3D(p1,p2,p3,p4,Nq,g)`. With the following arguments:

N_q	$(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$	ρ
1-point rule	$(1/4, 1/4, 1/4, 1/4)$	1
4-point rule	(0.5854102, 0.1381966, 0.1381966, 0.1381966)	0.25
	(0.1381966, 0.5854102, 0.1381966, 0.1381966)	0.25
	(0.1381966, 0.1381966, 0.5854102, 0.1381966)	0.25
	(0.1381966, 0.1381966, 0.1381966, 0.5854102)	0.25
5-point rule	$(1/4, 1/4, 1/4, 1/4)$	-4/5
	$(1/2, 1/6, 1/6, 1/6)$	9/20
	$(1/6, 1/2, 1/6, 1/6)$	9/20
	$(1/6, 1/6, 1/2, 1/6)$	9/20
	$(1/6, 1/6, 1/6, 1/2)$	9/20

Table 3: 3D gauss quadrature

`I`	$\in \mathbb{R}$	value of the integral
`p1`	$\in \mathbb{R}^3$	first corner point of the tetrahedron
`p2`	$\in \mathbb{R}^3$	second corner point of the tetrahedron
`p3`	$\in \mathbb{R}^3$	third corner point of the tetrahedron
`p4`	$\in \mathbb{R}^3$	fourth corner point of the tetrahedron
`Nq`	$\in \{1, 4, 5\}$	number of integration points
`g`	$: \mathbb{R}^3 \to \mathbb{R}$	function pointer*

Verify that the function evaluates correctly by comparing with the analytical solution of the integral

$$\iiint_{\Omega} e^x \, dx \, dy \, dz,$$

where Ω is the tetrahedron defined by the corner points $(0, 0, 0)$, $(0, 2, 0)$, $(0, 0, 2)$ and $(2, 0, 0)$.

(*) A function pointer in `matlab` is a variable which represents a function instead of the usual numerical values. In its simplest form it is declared as

```
f = @(x) x^2 + 1
```

which would cause the *variable* `f` to contain a *pointer* to the function $f(x) = x^2 + 1$. The function can then be evaluated using one of two methods

```
y = f(4);
y = feval(f,4);
```

both of which should yield the same result `y = 17`. A function may take in several arguments, i.e. $f(x, y) = x^2 + y^2$ may be declared as

```
f = @(x,y) x^2 + y^2
```

again the evaluation of the function is straightforward

```
y = f(2,2);
y = feval(f,2,2);
```

Provided that the actual function body is capable of vector or matrix operations, then the input arguments may be of vector or matrix form. The syntax remains unchanged by this. When vectorizing functions, remember that placing a dot before basic arithmetical operations causes them to be performed arraywise, eg

```
f = @(x) sum(x.^2,1)
```

will square every component of x and then sum them along the columns. This extends the previous function in the sense that $f([x;y])$ will evaluate $f(x,y) = x^2 + y^2$, but there is no longer a restriction on the dimension of the vector. Moreover, if a matrix $A = [v_1, \ldots, v_n]$ is input, f will compute a row vector $[f(v_1), \ldots, f(v_n)]$. Vectorizing functions in this manner not only allows for greater flexibility, but also speeds up computations. In fact, the `matlab` built-in numerical integration routines only accept vectorized functions.

You may also use variables in the function declaration, i.e.

```
a = 2;
f = @(x) x*a
```

will result in a function `f` which is doubling its input argument (even if `a` is changed at a later point).

2 Poisson in 2 dimensions

We are going to solve the two-dimensional Poisson problem, given by

$$\begin{aligned} \nabla^2 u(x,y) &= -f(x,y) \\ u(x,y)|_{\partial\Omega} &= 0, \end{aligned} \tag{1}$$

with f given as

$$f(x,y) = 16\pi^2 xy(x^2+y^2)\sin\left(2\pi(x^2+y^2)\right) - 24xy\pi\cos\left(2\pi(x^2+y^2)\right)$$

on the domain Ω given by a 3 quarter slice of the unit disc, i.e. in polar coordinates:
$\Omega = \{(r,\theta) \,:\, r \leq 1 \text{ and } \theta \in [0, 3\pi/2]\}$, see figure below

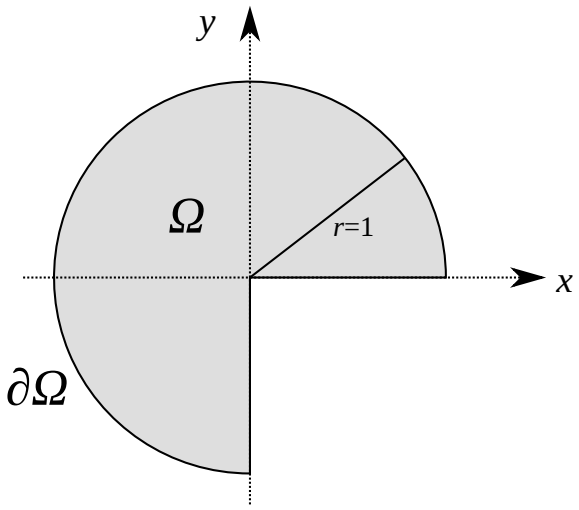

Figure 2: Domain Ω

The problem can be rewritten as: find $u \in V$

$$a(u,v) = F(v), \quad \forall v \in V.$$

with the bilinear functional a and the linear functional l given by

$$\begin{aligned} a(u,v) &= \iint_\Omega \nabla u \cdot \nabla v \, dx\, dy, \\ F(v) &= \iint_\Omega f v \, dx\, dy. \end{aligned}$$

where $V = H_0^1(\Omega)$

Let Ω be discretized into M triangles such that our computational domain is the union of all of these $\Omega = \cup_{k=1}^{M} K_k$. Each triangle K_k is then defined by its three corner *nodes* $\boldsymbol{x}_i$. We will then solve the Galerkin problem with $V = V_h$, where V_h is the space of elementwise linear polynomials, i.e.

$$V_h = \{v \in V \,:\, v|_{K_k} \in \mathbb{P}_1(K_k), 1 \leq k \leq M\}$$

The linear basis functions are then the usual nodal basis $\{\varphi_i\}_{i=1}^n$ satisfying

$$V_h = \text{span}\{\varphi_i\}_{i=1}^n \qquad \varphi_j(\boldsymbol{x}_i) = \delta_{ij}$$

where δ_{ij} is the Kronecker delta. By searching for a solution $u_h \in V_h$, it is then possible to write this as a weighted sum of the basis functions, i.e. $u_h = \sum_{i=1}^n u_h^i \varphi_i(x, y)$. The Galerkin problem is then equivalent to:

Find $\boldsymbol{u}$ such that

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{2}$$

with

$$\begin{aligned} \boldsymbol{A} &= [A_{ij}] = [a(\varphi_i, \varphi_j)] \\ \boldsymbol{u} &= [u_h^i] \\ \boldsymbol{f} &= [f_i] = [F(\varphi_i)]. \end{aligned}$$

a) Geometry

We are now going to actually solve the system (2). First we are going to take a look at the triangulation $\{K_k\}$. From the webpage `http://wiki.math.ntnu.no/tma4220/2017h/start` you may download the mesh generators. For organization purposes you might want to keep them in a separate directory and see the matlab `addpath` command.

The function `getSlice` outputs the following:

1. An $N \times 2$ matrix `p` for which the ith row is a row vector of the coordinates of the vertex $\mathbf{x}_i$
2. An $M \times 3$ 'connectivity' matrix `tri` for which the ith row is a row vector containing the vertex numbers of the 3 corners of the element K_i, i.e. if K_i is the element with corners $\mathbf{x}_5, \mathbf{x}_{12}, \mathbf{x}_{100}$, the ith row will be $[5, 12, 100]$.
3. A $P \times 2$ matrix `edge` for which each row is the vertex numbers i, j of an edge with endpoints $\mathbf{x}_i, \mathbf{x}_j$.

You will need to lookup the arrays `p,tri` frequently during the assembly process; whilst `edge` may be useful during the implementation of boundary conditions. You may find it useful to construct a triangulation class with this data, see the help files for the matlab function `triangulation`.

Plot at least three meshes of different sizes using the mesh generated from this function. You may want to check the matlab function `trimesh` or `triplot`.

b) Assembly

Now write a loop over the elements that constructs the stiffness matrix $\boldsymbol{A}$ and the load vector $\boldsymbol{f}$. For the load vector $\boldsymbol{f}$, you may require Gaussian quadrature.

The matrix $\boldsymbol{A}$ should now be singular. Verify this in your code and explain why this is the case.

c) Boundary conditions

Implement the homogeneous Dirichlet boundary conditions. Describe what method you used for this and how you did it.

d) Verification

Solve the system (2) and verify that you are getting (approximately) the same result as the analytical solution

$$u(x, y) = xy \sin\left(2\pi(x^2 + y^2)\right). \tag{3}$$

3 Neumann boundary conditions

We are going to change to boundary conditions of our problem to

$$\begin{aligned} \nabla^2 u(x,y) &= -f(x,y) \\ u(x,y)|_{\partial\Omega_D} &= 0, \\ \left.\frac{\partial u(x,y)}{\partial n}\right|_{\partial\Omega_N} &= g(x,y), \end{aligned} \tag{4}$$

with the source term f and exact solution u given as above, and g as

$$g(x,y) = \begin{cases} -x\sin(2\pi x^2), & y = 0 \text{ and } (x,y) \in \partial\Omega_N \\ +y\sin(2\pi y^2), & x = 0 \text{ and } (x,y) \in \partial\Omega_N \end{cases} \tag{5}$$

The dirichlet boundary condition is defined on $\partial\Omega_D = \{r = 1, \theta \in [0, 3pi/2]\}$, and the neumann boundary condition as $\partial\Omega_N = \{\theta = \{0, 3\pi/2\}\}$ shown in figure 3.

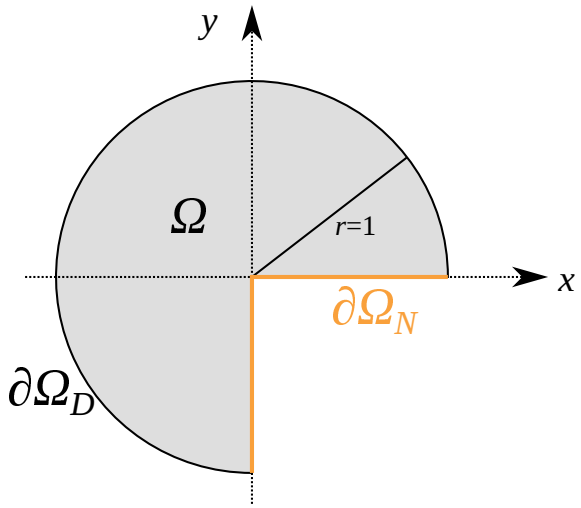

Figure 3: Dirichlet and Neumann boundary conditions

a) Boundary condition

Verify that (3) satisfies (5) on the boundary

b) Variational formulation

How does $a(\cdot,\cdot)$ and $F(\cdot)$ change with the introduction of Neumann boundary conditions?

c) Gauss quadrature

The Neumann boundary condition is given as an integral and should be evaluated using Gaussian quadrature. Modify your quadrature methods from task 1 to solve line integrals in two dimensions, i.e. the method signature in `I = quadrature1D(a,b,Nq,g)` should change to `a` $\in \mathbb{R}^2$ and `b` $\in \mathbb{R}^2$.

d) Implementation

Change your code from task 2 to solve this new boundary value problem. How does your solution in the interior compare to the one you got in task 2? How does your solution at the boundary compare?

4 Moving into 3 dimensions

a) The Poisson in 3d

We are now going to solve the problem

$$\begin{aligned} \nabla^2 u &= -f \\ u|_{\partial\Omega} &= 0 \end{aligned}$$

in three dimensions, meaning that we are looking for a solution $u(x, y, z)$.

Generate a mesh, using the function `getBox` from the downloaded mesh generators. This will give you three variables which will describe the nodal points, the tetrahedral elements and the index of the boundary nodes. These should be familiar from task 2 as the only difference is that spatial coordinates have one more component, as well as the elements require one more index to describe.

Modify your code from task 2 to deal with tetrahedral elements in three dimensions. Use the following f

$$f(x, y, z) = -12\pi^2 \sin(2\pi x) \sin(2\pi y) \sin(2\pi z)$$

and homogeneous Dirichlet boundary conditions ($u^D = 0$). This corresponds to the exact solution

$$u(x, y, z) = \sin(2\pi x) \sin(2\pi y) \sin(2\pi z) \tag{6}$$

b) Inhomogeneous dirichlet boundary conditions

Modify your equation to solve the problem

$$\begin{aligned} \nabla^2 u &= -f \\ u|_{\partial\Omega} &= g(x, y, z) \end{aligned}$$

on the domain Ω given by the unit ball. Use the same f as above, but modify your code to use the inhomogeneous dirichlet conditions $g(x, y, z) = u(x, y, z)$ from (6). Use the function `getBall` as your grid generator.

c) Volume visualization

Plot the domain Ω (i.e. the unit sphere). Note that you will not be required to plot every element, as most will be hidden on the inside of the domain. See the matlab function `TriRep` or `tetramesh` for functionality relating to this.

Plot your solution using isosurfaces. Note that the matlab function `isosurface` requires your data to be structured, which it currently is not. You will have to post process the data to get it on the desired form. Read up on `TriScatteredInterp` for this.

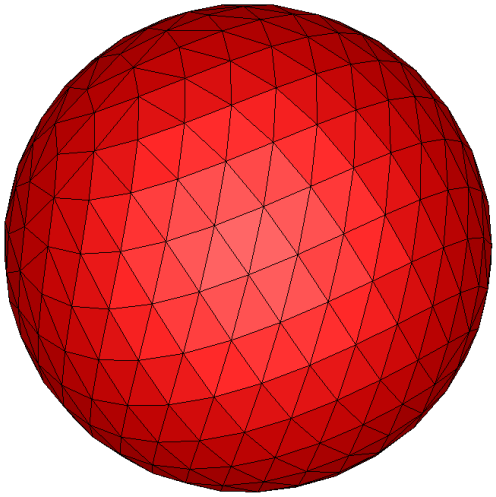

Figure 4: Sample 3d shape

c) GLview (optional)

GLview is a dedicated commercial visualising program to display the results from finite element analysis. It is available to all NTNU students through progdist. While it is possible to display 2D-results in GLview, the advantage of using a dedicated visualiser really is apparent when trying to draw 3D volumetric results.

Download and install GLview inova from progdist and write a `.vtf` output file to view your results. See the attached `writeVTF` function in the `grids.zip` file.

d) Neumann boundary condition (optional)

Change the problem definition to use Neumann boundary conditions for the top half of the sphere (i.e. $z > 0$) and Dirichlet for the bottom part. The neumann condition is given by

$$\begin{aligned} \frac{\partial u}{\partial n} \quad = \quad 2\pi(\quad & x \quad \cos(2\pi x) \quad \sin(2\pi y) \quad \sin(2\pi z) + \\ & y \quad \sin(2\pi x) \quad \cos(2\pi y) \quad \sin(2\pi z) + \\ & z \quad \sin(2\pi x) \quad \sin(2\pi y) \quad \cos(2\pi z)) \end{aligned}$$