# Project 1 - TMA4220 - 2022

**Deadline** : 02 October 2022 at midnight
**Weight on the final grade** : 10%

The programming project will be split into two parts. The first part 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.

To submit, send an email to `davide.murari@ntnu.no` attaching a folder with

- the Python code,
- a PDF file with the explanations and theoretical questions.

An alternative would be a Jupyter notebook with both the theoretical derivations and the code together. Make clear what are the answers to the theoretical questions. Specify the group and the group members when you submit.

## 1 Gaussian quadrature

At the heart of every finite element code, lies the evaluation of an integral. This integral may be of varying complexity depending on the problem at hand, and many of these integrals do not even have a known analytical solution. Some integrals are possible to solve analytically, but of such computational complexity that it is impractical to do so. As such, one often refers to numerical quadrature schemes to do the core integration. One popular integration scheme is the **Gaussian quadrature**.

In one dimension the Gauss quadrature takes the form

$$\int_{-1}^{1} g(z)dz \approx \sum_{q=1}^{N_q} \rho_q g\left(z_q\right),$$

where $N_q$ is the number of integration points, $z_q$ are the Gaussian quadrature points and $\rho_q$ are the associated Gaussian weights.

This extends to higher dimensions by

$$\int_{\hat{\Omega}} g(\boldsymbol{z}) d\boldsymbol{z} \approx \sum_{q=1}^{N_q} \rho_q g\left(\boldsymbol{z}_q\right)$$

specifying the vector quadrature points $\boldsymbol{z}_q$ as well as integrating over a suitable reference domain $\hat{\Omega}$ (i.e. squares or triangles in 2D, tetrahedra or cubes in 3D).

## 1.1 1D quadrature

Write a Python function `quadrature1D`$(a, b, N_q, g)$ that returns a value $I$ where the variables are defined as:

- $I \in \mathbb{R}$, value of the integral
- $a \in \mathbb{R}$, integration start
- $b \in \mathbb{R}$, integration end
- $N_q \in \{1, 2, 3, 4\}$, number of integration points
- $g : \mathbb{R} \to \mathbb{R}$, function pointer[1].

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

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

## 1.2 2D quadrature

Using all numerical quadratures, it is important to first map the function to the reference domain. In one dimension, this is the interval $\zeta \in [-1, 1]$. In higher dimensions, we often map to barycentric coordinates (or area coordinates as they are known in 2D). The gauss points are then given as triplets in this coordinate system. The area coordinates are defined by

$$\zeta_1 = \frac{A_1}{A}$$

$$\zeta_2 = \frac{A_2}{A}$$

[1] A **function pointer** in Python is a variable which represents a function instead of the usual numerical values. In its simplest form it is declared as `f= lambda 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 by $y = f(4)$, which should yield the result $y = 17$. A function may take several arguments, i.e. $f(x, y) = x^2 + y^2$ can be declared as `f = lambda x,y: x**2 + y**2`. This is even compatible with vector or matrix operations.

| $N_q$ | $z_q$ | $\rho_q$ |
|---|---|---|
| 1-point-rule | 0 | 2 |
| 2-point-rule | $-\sqrt{1/3}$ | 1 |
| | $\sqrt{1/3}$ | 1 |
| | $-\sqrt{3/5}$ | 5/9 |
| 3-point-rule | 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 nodes and weights for the reference interval $[-1, 1]$

$$\zeta_3 = \frac{A_3}{A}$$

where $A_1$, $A_2$ and $A_3$ are the area of the traingles depicted in figure 1 and $A$ is the total area of the triangle. Note that these do not form a linear independent basis as $\zeta_1 + \zeta_2 + \zeta_3 = 1$. An intuitive way to interpret the barycentric coordi-

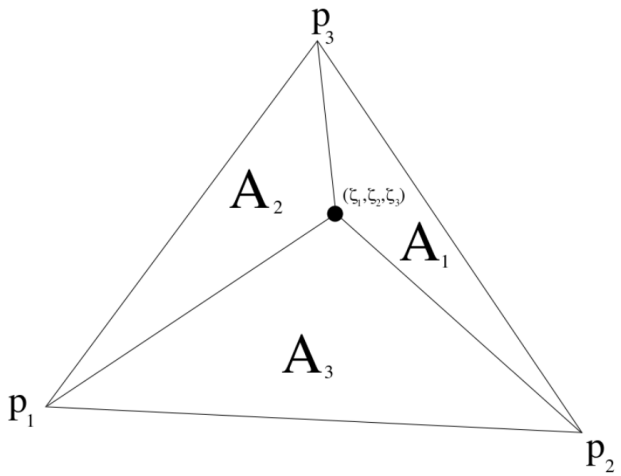


Figure 1: Barycentric coordinates in two dimensions

nates is as the coefficients of the weighted sum of the triangle vertices that give out the point of interest. More precisely, it is the triple $(\zeta_1, \zeta_2, \zeta_3)$ such that

$$P = \zeta_1 p_1 + \zeta_2 p_2 + \zeta_3 p_3 = \zeta_1 p_1 + \zeta_2 p_2 + (1 - \zeta_1 - \zeta_2) p_3.$$

Manipulating a bit this relation, one can get also another interpretation of the barycentric coordinates $\zeta_1, \zeta_2$:

$$P = p_3 + \zeta_1(p_1 - p_3) + \zeta_2(p_2 - p_3)$$

and hence $(\zeta_1, \zeta_2)$ are the coordinates of the point $P$ in the reference frame $(p_3, \overrightarrow{p_3p_1}, \overrightarrow{p_3p_2})$.

| $N_q$ | $(\zeta_1, \zeta_2, \zeta_3)$ | $\rho$ |
|---|---|---|
| 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 nodes and weights expressed in barycentric coordinates.

We remark that these quadrature rules should all be exact for at least the constant function $f(x, y) = 1$. Thus, we see that on the reference triangle $T_e$ the quadrature rule has to be of the form

$$\int_{T_e} f(\xi) d\xi \approx \frac{1}{2} \sum_{q=1}^{N_q} \rho_q f(z_q)$$

when considering the weights and nodes in Table 2. Indeed if $f(\xi) = 1$ and we use the 1-point rule we get

$$\frac{1}{2} = \int_{T_e} 1 d\xi, \text{ and } \frac{1}{2}\rho_1 z_1 = \frac{1}{2} f(z_1) = \frac{1}{2}$$

as desired.

Write a Python function `quadrature2D` $(p_1, p_2, p_3, N_q, g)$ that returns a value $I$ where the variables are defined as:

- $I \in \mathbb{R}$, value of the integral,
- $p_1 \in \mathbb{R}^2$, first corner point of the triangle,
- $p_2 \in \mathbb{R}^2$, second corner point of the triangle,
- $p_3 \in \mathbb{R}^2$, third corner point of the triangle,
- $N_q \in \{1, 3, 4\}$, number of integration points,

- $g : \mathbb{R}^2 \to \mathbb{R}$, function pointer.

**Hint**: An easy way of mapping barycentric coordinates $\boldsymbol{\zeta}$ to physical coordinates $\boldsymbol{x}$ is by $\boldsymbol{x} = \zeta_1 \boldsymbol{p}_1 + \zeta_2 \boldsymbol{p}_2 + \zeta_3 \boldsymbol{p}_3$, where $\boldsymbol{p}_i, i = 1, 2, 3$ are the corner points of the triangle.

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

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

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

# 2 Poisson in 2 dimensions

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

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

with $f$ given as

$$f(x, y) = -8\pi \cos\left(2\pi(x^2 + y^2)\right) + 16\pi^2(x^2 + y^2)\sin\left(2\pi(x^2 + y^2)\right)$$

on the domain $\Omega$ given by the unit disk, i.e. $\Omega = \left\{(x, y) : x^2 + y^2 \leq 1\right\}$. Recall $\partial\Omega = \{(x, y) : x^2 + y^2 = 1\}$.

**Advice:** For the visual representation, you might find useful the following Python libraries and methods:

- `https://matplotlib.org/stable/api/tri_api.html`
- `https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.tricontourf.html`

## 2.1 Analytical solution

Verify that the following expression is in fact a solution to the problem (1):

$$u(x, y) = \sin\left(2\pi\left(x^2 + y^2\right)\right).$$

## 2.2 Weak formulation

Show that the problem can be rewritten as

$$a(u, v) = l(v), \quad \forall v \in X$$

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

$$a(u,v) = \iint_\Omega \nabla u \cdot \nabla v dx dy$$
$$l(v) = \iint_\Omega f v dx dy.$$

What is the definition of the space $X$?

## 2.3 Galerkin projection

Instead of searching for a solution $u$ in the entire space $X$ we are going to be looking for a solution in a much smaller space $X_h \subset X$. Let $\Omega$ 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 $\boldsymbol{x}_i$. For each of these nodes there corresponds one basis function. The space $X_h$ is then defined by

$$X_h = \left\{ v \in X : v|_{K_k} \in \mathbb{P}_1(K_k), 1 \leq k \leq M \right\}$$

for which the basis functions $\{\varphi_i\}_{i=1}^n$ satisfy

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

and $\delta_{ij}$ is the Kronecker delta. By searching for a solution $u_h \in X_h$, it is then possible to write this as a weighted sum of the basis functions, i.e.

$$u_h = \sum_{i=1}^{n} u_h^i \varphi_i(x,y).$$

Show that the problem "Find $u_h \in X_h$ such that $a(u_h, v) = l(v) \quad \forall v \in X_h$ " is equivalent to the following problem:

Find $\boldsymbol{u}$ such that

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

with

$$\boldsymbol{A} = \left[A_{ij}\right] = \left[a\left(\varphi_i, \varphi_j\right)\right]$$
$$\boldsymbol{u} = \left[u_h^i\right]$$
$$\boldsymbol{f} = [f_i] = [l(\varphi_i)]$$

## 2.4 Implementation

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 `https://wiki.math.ntnu.no/tma4220/2022h/project` you may download the mesh generator.

The function `GetDisc`, in the script `getdisc.py`, is generating a mesh on the unit disk $\Omega$. Plot at least three meshes of different sizes using this function.

### 2.5 Stiffnes matrix

Build the stiffness matrix $\boldsymbol{A}$. Use the Gaussian quadrature from exercise 1 to do this. The matrix $\boldsymbol{A}$ should now be singular. Verify this in your code and explain why this is the case.

### 2.6 Right hand side

Build the right hand side vector $\boldsymbol{f}$ in the same manner as $\boldsymbol{A}$.

### 2.7 Boundary conditions

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

### 2.8 Verification

Solve the system (2) and verify that you are (approximately) getting the same result as the analytical solution seen in task 2.1. A good way to do so is to represent both the solutions and also their difference. Moreover, you should check if/how the quality of the approximation changes as the number of triangles in the mesh changes.

## 3 Neumann boundary conditions

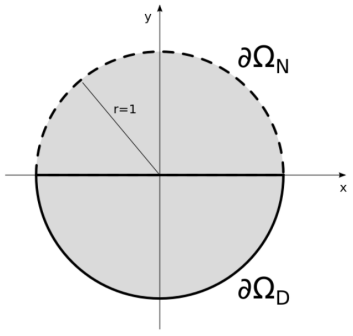


Figure 2: Dirichlet and Neumann boundary conditions

We are going to change the boundary conditions of our problem and study

the BVP

$$\begin{cases} \nabla^2 u(x,y) = -f(x,y) \\ u(x,y)\big|_{\partial\Omega_D} = 0 \\ \dfrac{\partial u(x,y)}{\partial n}\Big|_{\partial\Omega_N} = g(x,y) \end{cases}$$

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

$$g(x,y) = 4\pi\sqrt{x^2+y^2}\cos\left(2\pi(x^2+y^2)\right). \tag{3}$$

The Dirichlet boundary condition is defined on $\partial\Omega_D = \left\{x^2+y^2 = 1, y < 0\right\}$, and the Neumann boundary condition on $\partial\Omega_N = \left\{x^2+y^2 = 1, y > 0\right\}$, as shown in figure 2.

## 3.1 Boundary condition

Verify that the analytical solution presented in task 2.1 satisfies (3) at the boundary.

## 3.2 Variational formulation

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

## 3.3 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. `quadrature1D`$(a, b, N_q, g)$ should get as inputs $a \in \mathbb{R}^2$ and $b \in \mathbb{R}^2$.

As a test case, check again that it computes correctly the line integral of $e^x$ on the line segment connecting the points $a = (1,0)$ and $b = (2,0)$.

## 3.4 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?