Project 1 - TMA4220 - 2023

Deadline : 06 October 2023, at 11:59 pm
Weight on the final grade : 10%

The programming project will be split into two parts. The first part introduces the finite element method and is designed to build up a solid code base and design finite element solvers from scratch. For the second part, you will have to choose one of several tasks to implement, and some of them will be based on a higher-level finite element Python Library.

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

- the Python/Matlab code,
- a PDF file with the explanations of the obtained results.

An alternative would be a Jupyter notebook with the theoretical derivations and code. Specify the group and the group members when you submit.

This project focuses on the Poisson equation on one and two-dimensional domains. We build a finite element solver for the 1d problem and then extend it to 2d problems.

1 One dimensional problem

This first half of project 1 gives up to 60 points out of 100

Consider the boundary value problem (BVP)

$$\begin{cases} -u''(x) = f(x), & x \in (0,1) \\ u(0) = u(1) = 0, \end{cases} \tag{1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function. We introduce a partition of the domain $[0,1] \subset \mathbb{R}$ given by $n+1$ nodes

$$0 = x_0 < x_1 < ... < x_n = 1.$$

We denote with $\mathcal{T}_h = \{x_0, x_1, ..., x_n\}$ the given mesh, and call $h_i = x_i - x_{i-1}$ the sizes of the defined subintervals, i.e. the elements of our mesh.

As already seen with the first exercise sheet, one can obtain the weak formulation of this Poisson equation by integrating both sides against a test function $v \in H_0^1(0,1) := \{v : \mathbb{R} \to \mathbb{R} : \ v(0) = v(1) = 0\}$, and get the following weak formulation: Find $u \in H_0^1(0,1)$ such that

$$a(u,v) := \int_0^1 u'(x)v'(x)\mathrm{d}x = \int_0^1 f(x)v(x)\mathrm{d}x =: F(v), \quad \forall v \in V.$$

By introducing a finite-dimensional subspace $V_h = \mathrm{span}\{\varphi_1, ..., \varphi_m : \mathbb{R} \to \mathbb{R}\} \subset H_0^1(0,1)$, one can obtain the Ritz-Galerkin formulation of the same problem, which writes

$$a\left(\sum_{i=1}^{m} u_i \varphi_i, \varphi_j\right) = F(\varphi_j), \quad j = 1, ..., m.$$

Finding the solution of such Ritz-Galerkin formulation, i.e. the finite element approximation of u in V_h, is equivalent to solving the linear system

$$Au = F \in \mathbb{R}^m, \tag{2}$$

where

$$A_{ij} = a(\varphi_i, \varphi_j), \quad F_j = F(\varphi_j).$$

We now want to build such a solution for two specific finite-dimensional spaces: the space of piecewise linear and quadratic functions on $\mathcal{T}_h$. We first define the piecewise linear one:

$$V_h^1(\mathcal{T}_h) := \mathrm{span}\{\varphi_i : \mathbb{R} \to \mathbb{R} : \ i = 0, ..., n\}, \tag{3}$$

$$\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h_i}, & x_{i-1} \le x \le x_i \\ \frac{x_{i+1} - x}{h_{i+1}}, & x_i \le x \le x_{i+1} \\ 0, & \text{otherwise,} \end{cases} \quad i = 1, ..., n-1,$$

$$\varphi_0(x) = \begin{cases} \frac{x_1 - x}{h_1}, & 0 \le x \le x_1 \\ 0, & \text{otherwise,} \end{cases}$$

$$\varphi_n(x) = \begin{cases} \frac{x - x_{n-1}}{h_n}, & x_{n-1} \le x \le 1 \\ 0, & \text{otherwise.} \end{cases}$$

We then move to the definition of the piecewise quadratic one. For this space, we remark that to specify quadratic functions on the elements, we need to introduce additional nodes in the mesh. For symmetry, it is convenient to add the midpoints of each interval. We call them $x_{i+1/2}$, where $i = 0, ..., n-1$ and they are defined as

$$x_{i+1/2} = \frac{x_i + x_{i+1}}{2}.$$

We denote with $\varphi_{i+1/2}$ the basis function, which values 1 on the node $x_{i+1/2}$ and 0 on all the other ones.

$$
\begin{aligned}
V_h^2(\mathcal{T}_h) &:= \operatorname{span}\{\varphi_i : \mathbb{R} \to \mathbb{R} : i = 0, \dots, n\} \\
&\quad \cup \operatorname{span}\{\varphi_{i+1/2} : \mathbb{R} \to \mathbb{R} : i = 0, \dots, n-1\}, \qquad (4)
\end{aligned}
$$

$$
\varphi_0(x) := \begin{cases} \frac{(x-x_{1/2})(x-x_1)}{(x_0-x_{1/2})(x_0-x_1)} & x_0 \le x \le x_1 \\ 0 & \text{otherwise} \end{cases}
$$

$$
\varphi_n(x) := \begin{cases} \frac{(x-x_{n-1/2})(x-x_{n-1})}{(x_n-x_{n-1/2})(x_n-x_{n-11})} & x_{n-1} \le x \le x_n \\ 0 & \text{otherwise} \end{cases}
$$

$$
\varphi_i(x) := \begin{cases} \frac{(x-x_{i-1/2})(x-x_{i-1})}{(x_i-x_{i-1/2})(x_i-x_{i-1})} & x_{i-1} \le x \le x_i \\ \frac{(x-x_{i+1/2})(x-x_{i+1})}{(x_i-x_{i+1/2})(x_i-x_{i+1})} & x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \end{cases} \qquad i = 1, \dots, n-1,
$$

$$
\varphi_{i+1/2}(x) := \begin{cases} \frac{(x-x_i)(x-x_{i+1})}{(x_{i+1/2}-x_i)(x_{i+1/2}-x_{i+1})} & x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \end{cases} \qquad i = 0, \dots, n-1.
$$

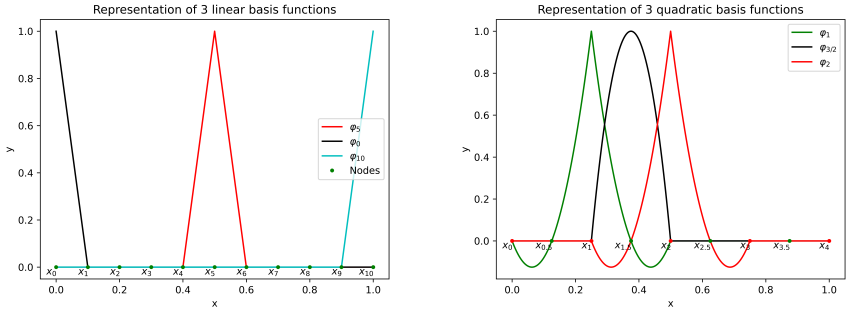

Figure 1: Representation of 3 basis functions for V_h^1, left, and V_h^2, right.

1.1 Integrals over an element

This question corresponds to up to 10 points

In this first point, we want to approximate the integral

$$
I := \int_{x_{i-1}}^{x_i} g(x)dx \qquad (5)
$$

for a given function $g : \mathbb{R} \to \mathbb{R}$. We want to do it using Gaussian quadrature rules, and hence as

$$
I \approx h_i \sum_{j=1}^{s} \rho_j g(x_{i-1} + \delta_j h_i), \qquad (6)
$$

for some quadrature weights $\rho_1, ..., \rho_s$ and nodes $\delta_1, ..., \delta_s$ rescaled over $[0, 1]$. For more details on Gaussian quadrature, see [1, 2].

To solve this task, you can use pre-existing libraries, like Scipy for Python; hence, you are not required to implement it from scratch (although this is still an accepted option).

Task 1 (7 points):

Write a method named *computeIntegral* that takes the inputs

- x_{i-1}, x_i : the two values defining the integration interval,
- g: a function you want to integrate over the interval,
- s: number of nodes of Gaussian quadrature rule used to approximate the integral,

and computes the approximation of (5) as in (6).

Task 2 (3 points):

The quadrature rule with s nodes should be exact when g is a polynomial up to degree $2s-1$. Verify that this happens on three test cases and report the results in a separate .pdf.

1.2 Building the load vector F

This question corresponds to up to 15 points

The finite element method is based on building quantities locally on each element, and then properly combining them to get the global linear system to solve, i.e. (2). We now want to exploit the method built-in section 1.1 to build the load vector F in (2). This task asks to do it for both the finite-dimensional spaces V_h^1 and V_h^2, see (3) and (4).

Task 1 (15 points):

Build a method named *loadVector* that takes the inputs

- *mesh* : a list/array containing the nodes of the mesh $\mathcal{T}_h$,
- $p \in \{1, 2\}$: this number determines if you use the finite-dimensional space V_h^1 or V_h^2, i.e. we have F for V_h^p,
- f : the function defining the right hand side of the problem (1),
- s : the number of Gaussian quadrature nodes to approximate the needed integrals.

The output of this method should be the load vector F in (2). **To get complete score** this method has to be built first by assembling the elemental load vectors and then combining them to get the full F vector. To understand what this means, let us think about the integral

$$F_i := \int_0^1 f(x)\varphi_i(x)\mathrm{d}x.$$

We can split this integral as

$$F_i = \sum_{j=1}^{n} \int_{x_{j-1}}^{x_j} f(x)\varphi_i(x)\mathrm{d}x.$$

However, recalling the definition of $\varphi_i(x)$ for V_h^1, we notice that it is non-zero only on at most two elements. For example, if $i \in \{1,...,n-1\}$ we have

$$F_i = \int_{x_{i-1}}^{x_i} f(x)\varphi_i(x)\mathrm{d}x + \int_{x_i}^{x_{i+1}} f(x)\varphi_i(x)\mathrm{d}x.$$

This translates into the algorithm

$F \leftarrow \vec{0} \in \mathbb{R}^{n+1}$
for $i \in \{1,...,n\}$ **do**
  $I1 \approx \int_{x_{i-1}}^{x_i} f(x)\varphi_{i-1}(x)\mathrm{d}x$
  $I2 \approx \int_{x_{i-1}}^{x_i} f(x)\varphi_i(x)\mathrm{d}x$
  $F[i-1] = F[i-1] + I1$
  $F[i] = F[i] + I2$
end for

This algorithm loops over the n elements of the mesh and, on each element, approximates the only two non-zero integrals on that integration domain. These are then added to the correct entry of the output vector. We remark that we are working with the Python indexing convention, where the first index is 0.

1.3 Building the stiffness matrix A

This question corresponds to up to 15 points

In this task, we want to write a method that assembles the stiffness matrix A with entries

$$A_{ij} = a(\varphi_i, \varphi_j) = \int_0^1 \varphi_i'(x)\varphi_j'(x)\mathrm{d}x.$$

To do this, we notice that since the basis functions are generally locally supported, the matrix A will be sparse. Thus, it is not efficient to directly compute these integrals on the complete interval $(0,1)$. We hence follow a local-to-global approach as in the assembly of the load vector. This means that the procedure to implement for V_h^1 has to follow this structure:

$A \leftarrow 0 \in \mathbb{R}^{n+1\times n+1}$
for $i \in \{1,...,n\}$ **do**
$I1 \approx \int_{x_{i-1}}^{x_i} \varphi'_{i-1}(x)\varphi'_{i-1}(x)\mathrm{d}x$
$I2 \approx \int_{x_{i-1}}^{x_i} \varphi'_{i-1}(x)\varphi'_i(x)\mathrm{d}x$
$I3 \approx \int_{x_{i-1}}^{x_i} \varphi'_i(x)\varphi'_i(x)\mathrm{d}x$
$A[i-1,i-1] = A[i-1,i-1] + I1$
$A[i-1,i] = [i-1,i] + I2$
$A[i,i-1] = A[i,i-1] + I2$
$A[i,i] = A[i,i] + I3$
end for

Task 1 (5 points):

Implement the routine to assemble the stiffness matrix A for the case V_h^1. Build a method called *assembly1dLinear* that takes the following inputs

- *mesh* : list/array with the nodes in $\mathcal{T}_h$
- s: number of Gaussian quadrature nodes to use in the approximation of the integrals,

and outputs the matrix $A \in \mathbb{R}^{n+1\times n+1}$.

Task 2 (10 points):

Implement the routine to assemble the stiffness matrix A for the case V_h^2. Build a method called *assembly1dQuadratic* that takes the following inputs

- *mesh* : list/array with the nodes in $\mathcal{T}_h$,
- s : number of Gaussian quadrature nodes to use in the approximation of the integrals,

and outputs the matrix $A \in \mathbb{R}^{2n+1\times 2n+1}$.

1.4 Imposition of the boundary conditions

This question corresponds to up to 10 points

Our boundary value problem (1) has homogeneous Dirichlet boundary conditions, i.e. $u(0) = u(1) = 0$.

Task 1 (5 points):

Write a function called *applyHomogeneousDirichlet* that takes as input the stiffness matrix A and the load vector F and returns them transformed so that the boundary conditions are applied. There is no unique way to do this. Choose one approach and explain it.

Task 2 (5 points):

What do you have to do if, instead of having both Dirichlet boundary conditions, you had $u(0) = 0$ and $u'(1) = 0$? You do not have to write the code for this. Just describe the procedure in the .pdf document you submit with the code.

1.5 Solution of the linear system and comparison with analytical solution

This question corresponds to up to 10 points

We now focus on the case $f(x) = 16\pi^2 \sin(4\pi x)$. The analytical solution here is $\overline{u}(x) = \sin(4\pi x)$.

Task 1 (6 points):

We now want to compute the finite element approximation of the solution for meshes $\mathcal{T}_h$ having $n+1$ equispaced nodes on the interval $[0,1]$, i.e. $h_i = 1/n$ for every $i = 1, ..., n$.

For each of the ns in the set $\{2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ assemble the stiffness matrix A, and the load vector F for both V_h^1 and V_h^2, setting $f(x) = 16\pi^2 \sin(4\pi x)$. Apply the homogeneous Dirichlet boundary conditions and solve the linear system $Au = F$ with your favourite solver for linear systems. Be careful that the system can get quite big as n increases, so an informed choice on the solver needs to be made here.

Let $\overline{u}(x)$ be the analytical solution to the problem. Define the vectors with entries given by the evaluation of $\overline{u}(x)$ on all the nodes in the meshes (including the hidden ones for quadratic elements). Compute then the ℓ^2 norm of the difference between this vector and the approximations provided by the solutions of the linear systems. Plot them all in a log-log plot using the x axis given by n and y axis given by the computed ℓ^2 norm.

Task 2 (4 points):

We now want to compute the finite element approximation of the solution for meshes $\mathcal{T}_h$ having $n+1$ points randomly picked from the uniform distribution on $[0,1]$. Order these points increasingly and use them to define a mesh with non-uniform interval lengths. Two of the points in the mesh must be $x_0 = 0$ and $x_n = 1$.

For each of the ns in the set $\{2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ assemble the stiffness matrix A, and the load vector F for both V_h^1 and V_h^2, setting $f(x) = 16\pi^2 \sin(4\pi x)$. Apply the homogeneous Dirichlet boundary conditions and solve the linear system $Au = F$ with your favourite solver for linear systems. Repeat the plot in the experiment with uniform mesh and comment on the results. How are the errors behaving? Do you see the same decay in the error as in the uniform mesh?

2 Two dimensional problem

This second half of project 1 gives up to 40 points out of 100

Consider the boundary value problem (BVP)

$$\begin{cases} -\Delta u(x,y) = f(x,y), \quad (x,y) \in (0,1)^2 \\ u(x,0) = u(x,1) = u(0,y) = u(1,y) = 0, \quad x,y \in [0,1], \end{cases} \tag{7}$$

where $f : \mathbb{R}^2 \to \mathbb{R}$ is a continuous function, and $\Delta u(x,y) = \partial_{xx} u(x,y) + \partial_{yy} u(x,y)$. We introduce a partition of the domain $[0,1]^2 \subset \mathbb{R}^2$ given by a quadrilateral mesh defined with $n+1$ nodes on all the 4 edges:

$$0 = x_0 < x_1 < \ldots < x_n = 1, \quad 0 = y_0 < y_1 < \ldots < y_n = 1,$$

see Figure 2. We denote with $\mathcal{T}_h = \{(x_i, y_j) : \ i,j \in \{0,\ldots,n\}\}\}$ the given mesh. We remark that if we call $\mathcal{S}_{h,x} = \{0 = x_0 < x_1 < \ldots < x_n = 1\}$ and $\mathcal{S}_{h,y} = \{0 = y_0 < y_1 < \ldots < y_n = 1\}$, it follows $\mathcal{T}_h = \mathcal{S}_{h,x} \times \mathcal{S}_{h,y}$. This will be helpful afterwards to build a finite element approximation to the solution u of (7).

2.1 Weak formulation

This question corresponds to up to 10 points

The integration by parts rule extends to higher dimensions. Let $\Omega \subset \mathbb{R}^2$. We recall that given a scalar-valued function $u : \Omega \to \mathbb{R}$ and a vector-valued one $V : \Omega \to \mathbb{R}^2$, it holds

$$\nabla \cdot (uV) = \langle \nabla u, V \rangle + u \nabla \cdot V,$$

where $V(x,y) = [V_1(x,y), V_2(x,y)]$, and $\nabla \cdot V = \partial_x V_1(x,y) + \partial_y V_2(x,y)$. This implies, by the Divergence theorem, that

$$\begin{aligned} &\int_{\partial\Omega} \langle u(x,y)V(x,y), n(x,y) \rangle \mathrm{d}x\mathrm{d}y = \int_\Omega \nabla \cdot u(x,y)V(x,y) \mathrm{d}x\mathrm{d}y \\ &= \int_\Omega \langle \nabla u(x,y), V(x,y) \rangle \mathrm{d}x\mathrm{d}y + \int_\Omega u(x,y) \nabla \cdot V(x,y) \mathrm{d}x\mathrm{d}y, \end{aligned}$$

which is exactly the generalisation of the integration by parts formula. This is an important result to derive the weak formulation of (7).

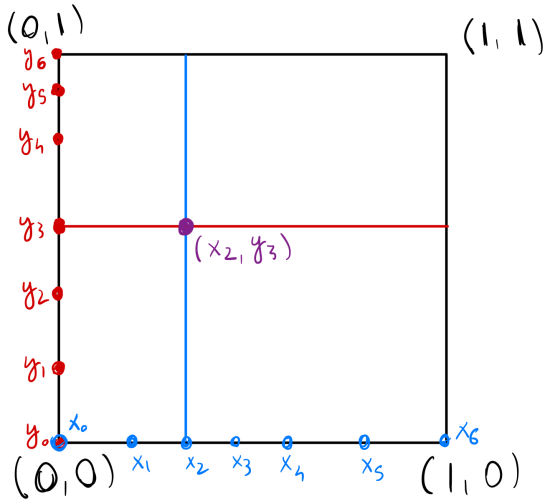

Figure 2: Nodes of the mesh for the case $n = 6$

Task 1 (5 points):

Find the weak formulation of the problem in (7), defining the bilinear form $a(\cdot,\cdot)$, the linear functional $F(\cdot)$ and the functional space V where it is defined.

Task 2 (5 points):

Write now the Ritz-Galerkin formulation of such a problem for the finite-dimensional space

$$V_h(\mathcal{T}_h) = \text{span}\{\Psi_i : [0,1]^2 \to \mathbb{R} : \ i = 0, ..., N\},$$

where $N+1$ is the dimension of the finite dimensional space $V_h(\mathcal{T}_h)$.

We now focus only on piecewise linear finite elements. Thus, when we refer to φ_i or ϕ_i, it is an element of the basis functions in V_h^1 defined by (3). However, everything we do can be extended to general basis functions.

2.2 Building the stiffness matrix

This question corresponds to up to 10 points

We now want to extend the methods written for the 1d problem and apply them to the 2d case in equation (7). To do so, we define two finite-dimensional spaces made of so-called "tensor-product basis functions". The general idea is to start with a pair of finite-dimensional spaces $V_h(\mathcal{S}_{h,x}) = \{\varphi_i : [0,1] \to \mathbb{R} : \ i =$

$0,...,n\}$ and $V_h(\mathcal{S}_{h,y}) = \{\phi_i : [0,1] \to \mathbb{R} :\ i = 0,...,n\}$ with

$$\varphi_i(x_j) = \phi_i(y_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

for every i and every $x_j \in \mathcal{S}_{h,x}$, $y_j \in \mathcal{S}_{h,y}$. Then, for the mesh $\mathcal{T}_h = \mathcal{S}_{h,x} \times \mathcal{S}_{h,y}$ one can build a finite dimensional space

$$V_h(\mathcal{T}_h) = \text{span}\{\Psi_{ij} : [0,1]^2 \to \mathbb{R} :\ \Psi_{ij}(x,y) = \varphi_i(x)\phi_j(y),\ \varphi_i \in V_h(\mathcal{S}_{h,x}),\ \phi_i \in V_h(\mathcal{S}_{h,y})\}. \tag{8}$$

Again, each basis function of $V_h(\mathcal{T}_h)$ is valued 1 on one and only one node of $\mathcal{T}_h$, and is valued 0 on the others.

We now recall a helpful tool for our calculations, which is the Kronecker product of two matrices $A, B \in \mathbb{R}^{d\times d}$, i.e. $C = A \otimes B \in \mathbb{R}^{d^2\times d^2}$ defined as

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ a_{21} & a_{22} & \dots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \dots & a_{dd} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1d} \\ b_{21} & b_{22} & \dots & b_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & \dots & b_{dd} \end{bmatrix},$$

$$C = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1d}B \\ a_{21}B & a_{22}B & \dots & a_{2d}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1}B & a_{d2}B & \dots & a_{dd}B \end{bmatrix}.$$

From now onwards, both φ_i and ϕ_i are piecewise linear basis functions as defined by (3). Furthermore, since (7) has homogeneous Dirichlet boundary conditions, we can solve a reduced problem. Indeed, let

$$u_h(x,y) = \sum_{i,j=0}^{n} \Psi_{ij}(x,y)u_{i,j}$$

be the finite-dimensional representation of the solution on $V_h(\mathcal{T}_h)$. We can then say that

$$u_{0,j} = u_{n,j} = u_{i,0} = u_{i,n} = 0, \quad \forall i,j \in \{0,...,n\}.$$

This means that we only need to find the remaining coefficients. In other words, we can suppose $u_h(x,y)$ to be defined as

$$u_h(x,y) = \sum_{i,j=1}^{n-1} \Psi_{ij}(x,y)u_{i,j}.$$

In practice, we work only with a restricted set of basis functions. Thus, the stiffness matrix and load vector we are going to build have the following sizes:

$$A^r \in \mathbb{R}^{n-1\times n-1},\ F^r \in \mathbb{R}^{n-1},$$

where we use the superscript r to denote they are *reduced*. In other words, when we use this superscript, we refer only to the internal nodes.

We also notice that the set of indices

$$\{1,...,n-1\}\times\{1,...,n-1\}$$

made of pairs (i,j) can be ordered. In other words, we can introduce a bijective function

$$\ell : \{1,...,n-1\}\times\{1,...,n-1\} \to \{1,...,(n-1)^2\},$$

where $\ell(i,j)$ is the label associated to the pair (i,j). We use this to access entries in the stiffness matrix and the load vector. Here, to make the indexing more natural, we set the first index to be 1 and not 0. This makes the Kronecker structure easier to understand.

Task 1 (5 points):

We have now fixed the labelling function to

$$\ell(i,j) = (n-1)(j-1)+i, \tag{9}$$

which for the case $n = 4$ corresponds to the labelling in Figure 3. The reduced stiffness matrix $A^r \in \mathbb{R}^{(n-1)^2\times(n-1)^2}$ has thus entries given by

$$A^r_{\ell(i,j),\ell(l,m)} = A^r_{(n-1)(j-1)+i,(n-1)(m-1)+l} = \int_{[0,1]^2} \langle \nabla\Psi_{ij}(x,y), \nabla\Psi_{lm}(x,y)\rangle \mathrm{d}x\mathrm{d}y.$$

Let $A^r_x, A^r_y \in \mathbb{R}^{n-1\times n-1}$ be the 1−d reduced stiffness matrices with entries

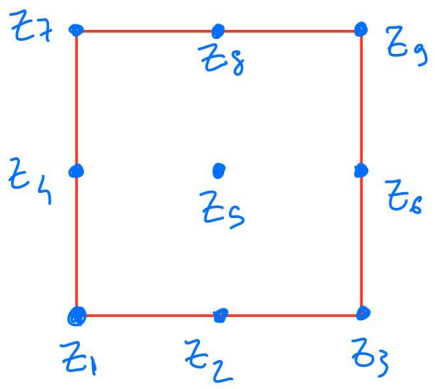

Figure 3: Labelling defined in (9), for $n = 4$

$$(A^r_x)_{ij} = \int_0^1 \varphi_i'(x)\varphi_j'(x)\mathrm{d}x, \quad (A^r_y)_{lm} = \int_0^1 \phi_l'(y)\phi_m'(y)\mathrm{d}y,$$

for $i,j,l,m \in \{1,...,n-1\}$, and let $M_x^r, M_y^r \in \mathbb{R}^{n-1\times n-1}$ be the reduced 1−d mass matrices with entries

$$(M_x^r)_{ij} = \int_0^1 \varphi_i(x)\varphi_j(x)\mathrm{d}x, \quad (M_y^r)_{lm} = \int_0^1 \phi_l(y)\phi_m(y)\mathrm{d}y,$$

again for $i,j,l,m \in \{1,...,n-1\}$.

Show that the reduced stiffness matrix A^r associated to the tensor-product basis functions defining $V_h(\mathcal{T}_h)$ in (8) can be written as

$$A^r = A_y^r \otimes M_x^r + M_y^r \otimes A_x^r.$$

Task 2 (3 points):

Write a method that computes the reduced mass matrix M_x^r for the set of piecewise linear basis functions. More precisely, given $V_h(\mathcal{S}_{h,x})$ as in (3), define a matrix $M_x^r \in \mathbb{R}^{n-1\times n-1}$ with entries given by

$$(M_x^r)_{ij} = \int_0^1 \varphi_i(x)\varphi_j(x)\mathrm{d}x,\ i,j \in \{1,...,n-1\}.$$

Follow a similar algorithm to the one you wrote to assemble the stiffness matrix, i.e. as in section 1.3. Call the method *massMatrix* and write it so it receives as input

- *mesh*: a list/array of nodes in $[0,1]$,
- s : the number of quadrature nodes to approximate the integrals.

The method should give the matrix $M_x^r \in \mathbb{R}^{n+1\times n+1}$ as an output.

Task 3 (2 points):

We first notice that the method written for the 1-d stiffness matrix can be used to obtain the reduced ones A_x^r, A_y^r. Indeed, A_x^r, A_y^r are given precisely by the full matrices where you remove the first and last rows and columns, i.e. for the case A_x^r you have

$$\mathbb{R}^{n+1\times n+1} \ni A_x^{\text{full}} = \begin{bmatrix} a_{00} & \dots & a_{0n} \\ \vdots & A_x^r & \vdots \\ a_{n0} & \dots & a_{nn} \end{bmatrix}.$$

Write a method that computes the reduced stiffness matrix $A^r \in \mathbb{R}^{(n-1)^2\times(n-1)^2}$ using the Kronecker product structure just discovered.

Call such function *stiffness2d* and make it receive as inputs

- *mesh*_x : a list/array of $n+1$ nodes in $[0,1]$, $0 = x_0 < x_1 < ... < x_{n-1} < x_n = 1$,
- *mesh*_y : a list/array of $n+1$ nodes in $[0,1]$, $0 = y_0 < y_1 < ... < y_{n-1} < y_n = 1$,
- s : the number of quadrature nodes to approximate the integrals.

As an output of the method, there should be the reduced stiffness matrix A^r.

3 Building the load vector

This question corresponds to up to 10 points

We recall that for a sufficiently regular function $f : [0,1]^2 \to \mathbb{R}$, by Fubini's theorem, it follows

$$F^r_{\ell(i,j)} = \int_{[0,1]^2} f(x,y)\varphi_i(x)\phi_j(y)\mathrm{d}x\mathrm{d}y = \int_0^1 \left(\int_0^1 f(x,y)\varphi_i(x)\mathrm{d}x \right) \phi_j(y)\mathrm{d}y, \ i,j \in \{1,...,n-1\}.$$

Thus, we can compute $F^r_{\ell(i,j)}$ by computing 2 one-dimensional integrals:

$$G^r_i(y) := \int_0^1 f(x,y)\varphi_i(x)\mathrm{d}x, \quad F^r_{\ell(i,j)} = \int_0^1 G_i(y)\phi_j(y)\mathrm{d}y.$$

Task 1 (10 points):

Write a function *loadVector2d* that exploits the previously written method *loadVector* to compute the reduced load vector for this 2-dimensional problem, i.e. a vector $F^r \in \mathbb{R}^{(n-1)^2}$ with entries

$$F^r_{\ell(i,j)} = \int_0^1 f(x,y)\Psi_{ij}(x,y)\mathrm{d}x\mathrm{d}y.$$

The method should take as input

- *mesh_x* : a list/array of $n+1$ nodes in $[0,1]$, $0 = x_0 < x_1 < ... < x_{n-1} < x_n = 1$,
- *mesh_y* : a list/array of $n+1$ nodes in $[0,1]$, $0 = y_0 < y_1 < ... < y_{n-1} < y_n = 1$,
- s : the number of quadrature nodes to approximate the integrals.
- f : the function defining the right hand side of the problem (7),

and output $F^r \in \mathbb{R}^{(n-1)^2}$.

4 Test problem

This question corresponds to up to 10 points

Now, we have everything ready to solve a test problem. We consider the right-hand-side

$$f(x,y) = 5\pi^2 \sin(\pi x)\sin(2\pi y).$$

The problem has analytical solution $u(x,y) = \sin(\pi x)\sin(2\pi y)$.

Task 1 (5 points):

We now want to compute the finite element approximation of the solution for meshes $\mathcal{T}_h$ having $n+1$ equispaced nodes in both directions, i.e. x and y, on the domain $[0,1]^2$, i.e. $(x_i, y_j) = (hi, hj)$ where $h = 1/n$.

For each of the ns in the set $\{2^2, 2^3, 2^4, 2^5, 2^6, 2^7\}$ assemble the reduced stiffness matrix A^r, and the reduced load vector F^r taking piecewise linear basis functions. Apply the homogeneous Dirichlet boundary conditions, and solve the linear system $A^r u = F^r$ with your favourite solver for linear systems. Be careful that the system can get quite big as n increases, so an informed choice on the solver needs to be made here.

Let $\overline{u}(x, y)$ be the analytical solution to the problem. Define the vectors with entries given by the evaluation of $\overline{u}(x, y)$ on all the internal nodes, i.e.

$$\mathbb{R}^{(n-1)^2} \ni \overline{U},\ \overline{U}_{\ell(i,j)} = \overline{u}(x_i, y_j),\ i, j \in \{1, ..., n-1\}.$$

Compute then the ℓ^2 norm of the difference between this vector and the approximations provided by the solutions of the linear systems. Plot them all in a log-log plot using x axis given by n, and y axis given by the computed ℓ^2 norm.

Task 2 (5 points) [<u>More difficult</u>]:

We now push the code a bit more. You are tasked to solve as fast (efficiently) as possible the problem in Task 1 with $n = 2^9$. Various options are possible here. The first strategy can be to explore *sparse* storing and assembling of the matrices and vectors coupled with sparse linear solvers. Another is to exploit the Kronecker product structure in the stiffness matrix. In this task, it is important to motivate your choices, show the results you get regarding time improvements, and comment on them. More precisely, it would be best first to solve the problem naïvely, measure the required time, and then show the improvements you get.

References

[1] Gaussian quadrature - wiki. `https://en.wikipedia.org/wiki/Gaussian_quadrature`.

[2] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.