Numerical Approximation of PDEs

Spring Semester 2025

Lecturer: Prof. Annalisa Buffa Assistant: Mohamed Ben Abdelouahab

Session 4: March 20, 2025

Exercise 1. Suppose that $\mathcal{T}$ is a triangulation of a domain $\Omega \subseteq \mathbb{R}^2$ and that $V_h \subset H^1(\Omega)$ is the first-order finite element space with respect to that triangulation. Show that there exists a constant $C > 0$ such that for every vertex x of the triangulation and every $v \in V_h$:

$$|v(x)| \leq Ch^{-1} \|v\|_{L^2(\Omega)}.$$

Here, h is the maximum diameter of the cells.

Solution:
Let K be any triangle in $\mathcal{T}$ that includes the vertex x. We let $\hat{K}$ be the reference triangle, so that $K = F_K(\hat{K})$, and suppose that $\hat{x} = F_K^{-1}(x)$ is the corresponding vertex of the reference triangle. Then we see

$$v(x) = \hat{v}(\hat{x}), \tag{1}$$

where $\hat{v} = v \circ F_K$ is the transformation of v onto the reference triangle. Furthermore, we have by a change of variable

$$\|\hat{v}\|_{L^2(\hat{\Omega})} \leq \det(B_K^{-1})^{\frac{1}{2}} \|v\|_{L^2(\Omega)} \leq Ch_K^{-1} \|v\|_{L^2(\Omega)}. \tag{2}$$

That completes the proof.
In (2), we have made use of the fact that $|\hat{v}(\hat{x})| \leq c\|\hat{v}\|_{L^2(\hat{\Omega})}$ for some $c < 1$ that can be found by taking $v(x) = 0$ on all other vertices.

Exercise 2. The so-called **barycentric coordinates** are a popular tool in finite element methods. Consider the reference triangle $\hat{K}$ with vertices $\hat{v}_0 = (0,0)^T$, $\hat{v}_1 = (1,0)^T$ and $\hat{v}_3 = (0,1)^T$, as sketched in the lecture / lecture notes. The barycentric coordinates are linear functions $\hat{\lambda}_0, \hat{\lambda}_1, \hat{\lambda}_2$ that satisfy

$$\hat{\lambda}_i(v_j) = \delta_{ij}.$$

(In other words, they coincide with the hat functions.)

1. Check that $\hat{\lambda}_0 = 1 - x - y$, $\hat{\lambda}_1 = x$, and $\hat{\lambda}_2 = y$ on the reference triangle. Show that for any point $\hat{v} \in \hat{K}$ we have

   $$\hat{v} = \hat{\lambda}_0(v)\hat{v}_0 + \hat{\lambda}_1(v)\hat{v}_1 + \hat{\lambda}_2(v)\hat{v}_2.$$

   (This is why they are called "coordinates".)

2. Show that a basis of $P_2(\hat{K})$ is given by

   $$\hat{\lambda}_0\hat{\lambda}_0,\ \hat{\lambda}_0\hat{\lambda}_1,\ \hat{\lambda}_0\hat{\lambda}_2,\ \hat{\lambda}_1\hat{\lambda}_1,\ \hat{\lambda}_1\hat{\lambda}_2,\ \hat{\lambda}_2\hat{\lambda}_2.$$

Solution:

1. *We can easily the values at the vertices of the reference triangle. Moreover, if* $\hat{v} = (\hat{x}, \hat{y})$ *in Euclidean coordinates, then*

$$\begin{aligned}&\hat{\lambda}_0(v)\hat{v}_0 + \hat{\lambda}_1(v)\hat{v}_1 + \hat{\lambda}_2(v)\hat{v}_2 \\ &= (1 - \hat{x} - \hat{y})(0,0) + \hat{x}(1,0) + \hat{y}(0,1) = \hat{v}.\end{aligned}$$

2. *Since we have six proposed basis shape functions and the space is six-dimensional, we only need to show linear independence. In what follows, we write*

$$p = c_{00}\hat{\lambda}_0\hat{\lambda}_0 + c_{01}\hat{\lambda}_0\hat{\lambda}_1 + c_{02}\hat{\lambda}_0\hat{\lambda}_2 + c_{11}\hat{\lambda}_1\hat{\lambda}_1 + c_{12}\hat{\lambda}_1\hat{\lambda}_2 + c_{22}\hat{\lambda}_2\hat{\lambda}_2. \tag{3}$$

We outline three different approaches to solving this problem.

- *We can rewrite this in terms of the standard monomial basis. We have*

$$\begin{aligned}p &= c_{00}\hat{\lambda}_0\hat{\lambda}_0 + c_{01}\hat{\lambda}_0\hat{\lambda}_1 + c_{02}\hat{\lambda}_0\hat{\lambda}_2 + c_{11}\hat{\lambda}_1\hat{\lambda}_1 + c_{12}\hat{\lambda}_1\hat{\lambda}_2 + c_{22}\hat{\lambda}_2\hat{\lambda}_2 \\ &= c_{00}(1-x-y)^2 + c_{01}(1-x-y)x + c_{02}(1-x-y)y + c_{11}x^2 + c_{12}xy + c_{22}y^2 \\ &= c_{00}(1 + x^2 + y^2 - 2 - 2x - 2y) + c_{01}(x - x^2 - xy) + c_{02}(y - xy - y^2) + c_{11}x^2 + c_{12}xy + c_{22}y^2 \\ &= (-c_{00}) + (c_{01} - 2c_{00})x + (c_{02} - 2c_{00})y \\ &\qquad + (c_{11} + c_{00} - c_{01})x^2 + (c_{12} - c_{02} - c_{01})xy + (c_{22} + c_{00} - c_{02})y^2.\end{aligned}$$

  If we write p *in terms of the standard basis,*

$$p = a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2, \tag{4}$$

  then the transformation between the coefficients can be expressed by the system

$$\begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_{00} \\ c_{01} \\ c_{02} \\ c_{11} \\ c_{12} \\ c_{22} \end{pmatrix} = \begin{pmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{20} \\ a_{11} \\ a_{02} \end{pmatrix}$$

  The matrix is triangular with invertible diagonal entries. Hence it is invertible. If p *equals zero, then all the coefficients (4) in the monomial basis are zero, and then all coefficients (3) in the proposed barycentric basis are zero. Hence the the linear independence follows.*

- *A different approach: suppose that* $p = 0$*. We take the trace on the edge with* $y = 0$ *and find*

$$\begin{aligned}p(x,0) &= c_{00}\hat{\lambda}_0\hat{\lambda}_0 + c_{01}\hat{\lambda}_0\hat{\lambda}_1 + c_{11}\hat{\lambda}_1\hat{\lambda}_1 &(5)\\ &= c_{00}(1-x)^2 + c_{01}(1-x)x + c_{11}x^2. &(6)\end{aligned}$$

  We know that $p = 0$*. We can easily check that the polynomials* $(1-x)^2$, $(1-x)x$, x^2 *are linearly independent, so that* $c_{00} = c_{01} = c_{11} = 0$ *follows. Similarly, we take the trace on the edge with* $x = 0$ *and derive* $c_{00} = c_{02} = c_{22} = 0$*. Lastly, we must have* $p = c_{12}\hat{\lambda}_1\hat{\lambda}_2 = c_{12}xy$*.*

- *Alternatively, you compute the values at the six nodal points. For each barycentric monomial $\hat{\lambda}_i\hat{\lambda}_j$, you get a vector of six point values. You can then show that the six vectors are linearly independent and argue with the unisolvency of the point evaluations (no details here).*

Exercise 3. A general definition of the p-th order polynomial space $P_p(\hat{K})$ over $\hat{K}$ is the space of p-th order polynomials with canonical basis $\{\hat{x}^i\hat{y}^j \mid i + j \le p\}$, where we define $N_p := \dim P_p(\hat{K})$. Let $\hat{X} = (0, \frac{1}{p}, \frac{2}{p}, \dots, \frac{p-1}{p}, 1)$. A particularly useful FEM basis for this space is the basis $\{\hat{\phi}_1, \dots, \hat{\phi}_{N_p}\}$ that satisfies $\hat{\phi}_i(\hat{p}_j) = \delta_{ij}$, where $\hat{p}_j \in \{(\hat{x}, \hat{y}) \in \hat{X} \times \hat{X} \mid \hat{x} + \hat{y} \le 1\}$.

1. Show that $N_p := \dim P_p(\hat{K}) = \frac{p^2+3p+2}{2}$.
2. Derive a linear system of equations to find the weights $\{a_{jk}\}$ of $\hat{\phi}_i = \sum_{j,k} a_{jk}\,\hat{x}^j\hat{y}^k$ given that $\hat{\phi}_i(\hat{p}_j) = \delta_{ij}$.
3. On moodle you will fine the python template `code_04_03_template.py` which you can use to implement an algorithm capable of finding the a_{jk} of each $\hat{\phi}_i$.

Solution:

1. *From the definition of $P_p(\hat{K})$ we see that, clearly, N_p satisfies*

$$N_p = \dim\{(i,j) \in (\mathbb{Z}^{\ge 0} \times \mathbb{Z}^{\ge 0}) \mid i + j \le p\}.$$

Let us consider the regular lattice of points (i, j) with $i, j \in \{0, \dots, p\}$. The lattice has $(p+1)^2$ points in it and the dimension of $P_p(\hat{K})$ is the number of lattice points in the bottom left block including the diagonal running from the point $(p, 0)$ to $(0, p)$. The diagonal itself has $p+1$ points. Therefore, the strictly bottom left part has $\frac{(p+1)^2-(p+1)}{2}$ points in it. We add to that $p+1$ points of the diagonal and we acquire $N_p = \frac{(p+1)^2+p+1}{2} = \frac{p^2+3p+2}{2}$.

2. *Let $\hat{p}_j = (\hat{x}_j, \hat{y}_j)$. We define the $N_p \times N_p$ matrix M with*

$$M = \begin{bmatrix} 1 & \hat{x}_1 & \hat{y}_1 & \hat{x}_1\hat{y}_1 & \dots & \hat{y}_1^p \\ 1 & \hat{x}_2 & \hat{y}_2 & \hat{x}_2\hat{y}_2 & \dots & \hat{y}_2^p \\ \vdots & & & & & \vdots \\ 1 & \hat{x}_{N_p} & \hat{y}_{N_p} & \hat{x}_{N_p}\hat{y}_{N_p} & \dots & \hat{y}_{N_p}^p \end{bmatrix}.$$

Note that the choice of the ordering of the columns is not unique (i.e., we are assigning a particular global index to the multi-index (i, j)).
Then, the vector a_i with entries $a_i = (a_{00}, a_{10}, a_{01}, a_{11}, \dots, a_{0p})^T$ solves the equation $Ma = e_i$, where e_i is the i-th unit vector of length N_p.

3. *The python code can be found in Listing 1.*

Listing 1: Python code

```
import numpy as np
from itertools import product

def compute_polynomial_weights_local_Lagrange(p: int) -> np.ndarray:
    """
```

```
  Given the polynomial order p >= 1, compute the weights with respect
  to the canonical basis {1, y, y^2, ..., y^p, x, xy, ...., x^p} of
      P_p(Khat)
  for each basis function phi_i with phi_i(p_j) = \delta_ij.
  Here, the p_j are the points (x, y) \in X \times X, with X = {0,
      1/p, 2/p, ..., 1}
  and x + y <= 1.

  Paramters
  ---------
  p : `int`
    The polynomial order.

  Returns
  -------

  weights : np.ndarray
    A matrix of shape N_p x N_p whose i-th column contains the
        polynomial weights of \phi_i
    in the canonical python ordering (a_00, a_01, a_02, ... a_0p,
        a_10, a_11, ... a_p0 )
  """

assert (p := int(p)) >= 1
Np = (p**2 + 3 * p + 2) // 2

# create an array of multi-indices whose L-th row contains the multi
    index (i, j)
# representing the polynomial powers x^i y^j in the canonical python
    ordering.

# product(range(p+1), range(p+1)) creates the pairs:
# for i in range(p+1):
#   for j in range(p+1):
#     pair = (i, j)
multi_indices = np.stack([multi_index for multi_index in
    product(range(p+1), range(p+1)) if sum(multi_index) <=
    p]).astype(int)

# create an array containing as rows the p_j = (x_j, y_j) of the
    triangle.
P = multi_indices / p

# create a matrix M containing as L-th column the L-th canonical
    polynomial of P_p(Khat) evaluated in the P_j

M = np.empty((Np, Np), dtype=float)

# iterate simultaneously over the L-th column index and the
    corresponding multi index
for L, (i, j) in enumerate(multi_indices):
  M[:, L] = P[:, 0] ** i * P[:, 1] ** j

# create the right hand side matrix whose L-th column corresponds to
    the right hand side of the L-th nodal basis function.
Rhs = np.eye(Np)
```

```python
    # np.linalg.solve accepts several right hand sides as a matrix
    return np.linalg.solve(M, Rhs)

if __name__ == '__main__':
    for p in (1, 2, 3, 4):

        # round to 7 figures for better formatting.
        myweights = np.round(compute_polynomial_weights_local_Lagrange(p), 7)

        # create multi indices corresponding to order p in canonical python
            ordering
        multi_indices = tuple(multi_index for multi_index in
            product(range(p+1), range(p+1)) if sum(multi_index) <= p)

        # print to stdout
        print('With respect to the canonical polynomial basis with
            powers\n\n {},\n\n'
              'the weights of the nodal basis functions of order {} are
                given by: \n\n{}.\n\n'.format(str(multi_indices)[1:-1], p,
                '\n\n'.join(map(str, myweights.T))))
```

Exercise 4. [Building local stiffness and mass matrices] We use reference transformations to compute the local matrices over triangles.

1. Compute the local stiffness matrix for an arbitrary triangle K

$$\left(A^{loc,K}\right)_{i,j} = \int_K \nabla\varphi_i \cdot \nabla\varphi_j \, dx\, dy, \qquad i,j = 1,2,3$$

   where φ_i, φ_j are the $\mathbb{P}_1$ Lagrange basis functions in 2D.
   Hint: derive expressions for the linear map

$$F_K : \begin{array}{ccc} \hat{K} & \rightarrow & K \\ \hat{x} & \mapsto & B_K\hat{x} + b_K \end{array}$$

   where $\hat{K}$ is the reference element and $B_K \in \mathbb{R}^{2\times 2}$ and $b_K \in \mathbb{R}^{2\times 1}$. Then compute the local matrix $A^{loc,K}$ by recasting the integral over the reference element, as discussed in the lecture. Note that the ϕ_i and their local counterparts have a constant gradient !

2. Compute the local mass matrix for an arbitrary triangle K

$$\left(M^{loc,K}\right)_{i,j} = \int_K \varphi_i\varphi_j \, dx\, dy, \qquad i,j = 1,2,3$$

   where φ_i, φ_j are the Lagrange basis functions, by recasting the integral over the reference element.

3. Implement this as a Python code to assemble the full matrices. On Moodle, you find Python codes `mesh.py` and `code_04_04_template.py`. The file `mesh.py` provides you with a ready-to-go mesh class that you can use. [1] The file `code_04_04_template.py`

[1] This library relies on pygmsh, which should be easy to install with the command `pip install pygmsh`. To use the code **you do not need to understand the mesh.py, util.py and solve.py files in detail. Everything is explained in the template script.** Once you have finalised your implementation, you may run the script and it will plot a mesh and the solution of reaction-diffusion benchmark problem for you making use of the implemented matrices.

provides you two functions: `stiffness_matrix` and `mass_matrix`. You can complete these functions. These take a mesh as input and produce the respective matrices.

4. In the file `code_04_04_template.py` you also find the method `load_vector`. Implement this function to compute the load vector in the case of a piecewise constant right hand side. The function takes as input: the mesh, and the constant value F of the right-hand side.

Solution:

1. *Given an arbitrary triangle K with the vertices $\{a = (a_1, a_2),\ b = (b_1, b_2),\ c = (c_1, c_2)\}$, the mapping F_K of the reference triangle can be obtained by*

$$B_K = \begin{bmatrix} b_1 - a_1 & c_1 - a_1 \\ b_2 - a_2 & c_2 - a_2 \end{bmatrix}, \quad b_K = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}.$$

Recall that the reference shape functions have the constant gradients

$$\hat{\nabla}\hat{\varphi}_1 = (-1, -1),\ \hat{\nabla}\hat{\varphi}_2 = (1, 0),\ \hat{\nabla}\hat{\varphi}_3 = (0, 1).$$

We have $\varphi_i = \hat{\varphi}_i \circ F_K^{-1}$. By the chain rule, $\hat{\nabla}\varphi_i(x) = B_K^{-T}\hat{\nabla}\hat{\varphi}_i(F_K^{-1}(x))$. Substituting $x = F_K(\hat{x})$ in the integral leads to

$$\left(A^{loc,K}\right)_{i,j} = \int_K \nabla\varphi_i \cdot \nabla\varphi_j \, dx = \int_{\hat{K}} \left(B_k^{-T}\hat{\nabla}\hat{\phi}_i\right) \cdot \left(B_k^{-T}\hat{\nabla}\hat{\phi}_j\right) |\det B_K| \, d\hat{x}.$$

2. *Using the substitution to the reference element we derive*

$$\begin{aligned} \left(M^{loc,K}\right)_{i,j} &= \int_K \varphi_i\varphi_j \, dxdy = |\det B_K| \int_{\hat{K}} \hat{\varphi}_i\hat{\varphi}_i \, d\hat{x}d\hat{y} \\ &= |\det B_K| \int_0^1 \int_0^{1-\hat{y}} \hat{\varphi}_i\hat{\varphi}_i \, d\hat{x}d\hat{y} \end{aligned}$$

which results in

$$M^{loc,K} = \frac{|\det B_K|}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

3. *The implementation can be found on Moodle.*

4. *The local load vector is computed as*

$$L_i = \int_K F\varphi dx = F \int_{\hat{K}} \hat{\varphi} |\det B_K| d\hat{x} = F\frac{\det B_K}{6}$$

for F constant. An implementation of the local load vector can be found on Moodle.