Chapter 2

FEM for Poisson problem in 2D with triangular elements

Consider the two dimensional Poisson problem

$$-\Delta u = f \qquad \text{in } \Omega \tag{2.1a}$$

$$u = 0 \qquad \text{on } \partial\Omega \tag{2.1b}$$

In order to solve the problem (2.1), we multiply both sides of (2.1a) by the test function $v(x, y)$ and integrate over the domain,

$$-\int_\Omega \Delta u v \; d\boldsymbol{x} = \int_\Omega f v \; d\boldsymbol{x}. \tag{2.2}$$

Then, the weak formulation of the problem (2.1) is as follows: find $u(x, y) \in H_0^1(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v \; d\boldsymbol{x} = \int_\Omega f v \; d\boldsymbol{x}, \tag{2.3}$$

for all $v \in H_0^1(\Omega)$. The weak formulation (2.3) is obtained from (2.2) by using the boundary condition (2.1b) and integration by parts.

For the finite element approach, we need a conforming finite element space for the triangular elements

$$V_h^k = \{v_h \in H_0^1(\Omega) \mid v_h|_T \in P_k(T), \; \forall T \in \mathcal{T}_h\},$$

where $P_k(T)$ is the polynomial function space of degrees $\leq k$. Then, the variational formulation of the problem (2.1) is as follows: find $u_h \in V_h^k$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v_h \; d\boldsymbol{x} = \int_\Omega f v_h \; d\boldsymbol{x}, \tag{2.4}$$

for all $v_h \in V_h^k$.

2.1 Affine mapping

Let us define the reference triangle T_R as

$$T_R = \{\boldsymbol{r} = (r, s) \mid -1 \le r,\ s \le 1,\ \ r + s \le 0\}.$$

Barycentric coordinates $(\widetilde{\lambda}_0,\ \widetilde{\lambda}_1,\ \widetilde{\lambda}_2)$ have the properties

$$\begin{cases} 0 \le \widetilde{\lambda}_i(\boldsymbol{r}) \le 1, \qquad i = 0,\ 1,\ 2 \\ \widetilde{\lambda}_0(\boldsymbol{r}) + \widetilde{\lambda}_1(\boldsymbol{r}) + \widetilde{\lambda}_2(\boldsymbol{r}) = 1. \end{cases} \tag{2.5}$$

Let us define a triangle T as

$$T = \mathrm{span}\{\boldsymbol{v}_0,\ \boldsymbol{v}_1,\ \boldsymbol{v}_2\}, \quad \boldsymbol{v}_i = (v_i^{(1)}, v_i^{(2)}),$$

where $\boldsymbol{v}_0,\ \boldsymbol{v}_1,\ \boldsymbol{v}_2$ are vertices of T. Then, a point $\boldsymbol{x} = (x, y)$ in T can be represented as

$$\boldsymbol{x} = \lambda_0 \boldsymbol{v}_0 + \lambda_1 \boldsymbol{v}_1 + \lambda_1 \boldsymbol{v}_2.$$

Similarly, the point $\boldsymbol{r}$ in T_R can be written as

$$\begin{aligned} \begin{pmatrix} r \\ s \end{pmatrix} &= \lambda_0 \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \lambda_1 \begin{pmatrix} -1 \\ 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} -1 \\ -1 \end{pmatrix} \\ &= \begin{pmatrix} \lambda_0 - \lambda_1 - \lambda_2 \\ -\lambda_0 + \lambda_1 - \lambda_2 \end{pmatrix} = \begin{pmatrix} 2\lambda_0 - 1 \\ 2\lambda_1 - 1 \end{pmatrix} \end{aligned}$$

From above result and the property of barycentric coordinates, we have

$$\lambda_0 = \frac{r+1}{2}, \quad \lambda_1 = \frac{s+1}{2}, \quad \lambda_2 = -\frac{r+s}{2}.$$

Then, we have an affine mapping Φ such that

$$\Phi(\boldsymbol{r}) = \frac{r+1}{2}\boldsymbol{v}_0 + \frac{s+1}{2}\boldsymbol{v}_1 - \frac{r+s}{2}\boldsymbol{v}_2 = \boldsymbol{x}.$$

From the above equation,

$$\frac{\partial \boldsymbol{x}}{\partial r} = \frac{1}{2}\boldsymbol{v}_0 - \frac{1}{2}\boldsymbol{v}_2, \qquad \frac{\partial \boldsymbol{x}}{\partial s} = \frac{1}{2}\boldsymbol{v}_1 - \frac{1}{2}\boldsymbol{v}_2.$$

Hence,

$$\begin{aligned} x_r &= (v_0^{(1)} - v_2^{(1)})/2, \qquad y_r = (v_0^{(2)} - v_2^{(2)})/2, \\ x_s &= (v_1^{(1)} - v_2^{(1)})/2, \qquad y_s = (v_1^{(2)} - v_2^{(2)})/2. \end{aligned}$$

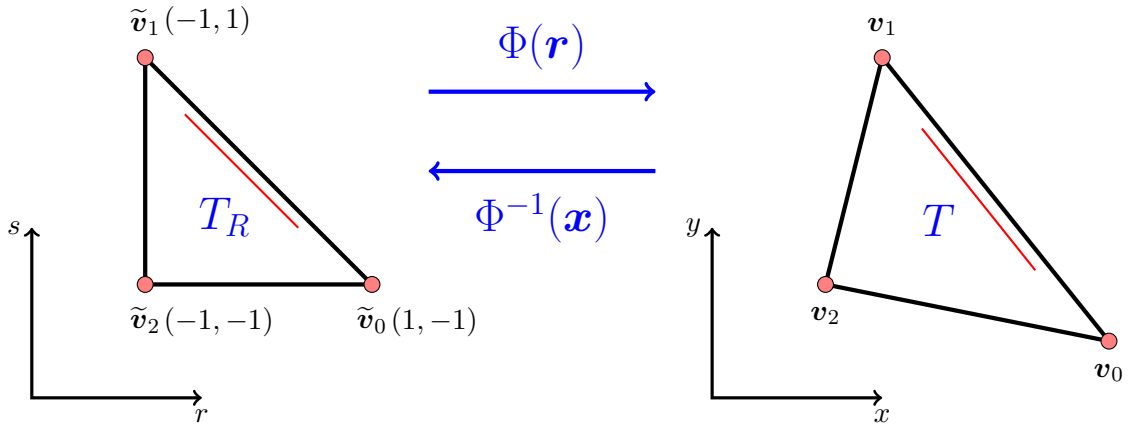


Figure 2.1: An affine mapping from the reference triangle T_R to a triangle T.

Using the property $I = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{r}} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}$, we have

$$I = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{r}} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} = \begin{bmatrix} x_r & x_s \\ y_r & y_s \end{bmatrix} \begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} = \frac{1}{x_r y_x - s_x y_r} \begin{bmatrix} y_s & -x_s \\ -y_r & x_r \end{bmatrix}.$$

Therefore,

$$r_x = \frac{y_s}{J}, \quad r_y = -\frac{x_s}{J}, \quad s_x = -\frac{y_r}{J}, \quad s_y = \frac{x_r}{J},$$

where $J = x_r y_s - x_s y_r$.

Similar to the rectangular elements, we have

$$\lambda_i(\boldsymbol{x}) = \widetilde{\lambda}_i(\Phi^{-1}(\boldsymbol{x})), \tag{2.6a}$$

$$\widetilde{\lambda}_i(\boldsymbol{r}) = \lambda_i(\Phi(\boldsymbol{r})). \tag{2.6b}$$

Then, we can obtain the following relations by simple calculation.

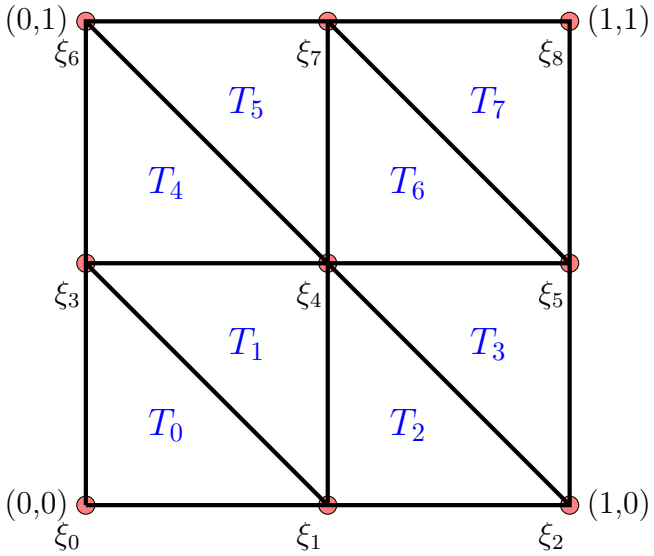
$$\frac{d}{dx}\lambda_i(\boldsymbol{x}) = \frac{dr}{dx}\frac{d}{dr}\widetilde{\lambda}_i(\Phi^{-1}(\boldsymbol{x})) + \frac{ds}{dx}\frac{d}{ds}\widetilde{\lambda}_i(\Phi^{-1}(\boldsymbol{x})) = r_x \frac{d}{dr}\widetilde{\lambda}_i(\boldsymbol{r}) + s_x \frac{d}{ds}\widetilde{\lambda}_i(\boldsymbol{r}) \tag{2.7a}$$

$$\frac{d}{dy}\lambda_i(\boldsymbol{x}) = \frac{dr}{dy}\frac{d}{dr}\widetilde{\lambda}_i(\Phi^{-1}(\boldsymbol{x})) + \frac{ds}{dy}\frac{d}{ds}\widetilde{\lambda}_i(\Phi^{-1}(\boldsymbol{x})) = r_y \frac{d}{dr}\widetilde{\lambda}_i(\boldsymbol{r}) + s_y \frac{d}{ds}\widetilde{\lambda}_i(\boldsymbol{r}) \tag{2.7b}$$

2.2 Triangulation

Consider the domain $\Omega = (0,1)^2$. The number of nodes in the triangulation $\mathcal{T}_h$ is determined by the number of elements and the polynomial order. Let N be the number of nodes, $h = 1/M$

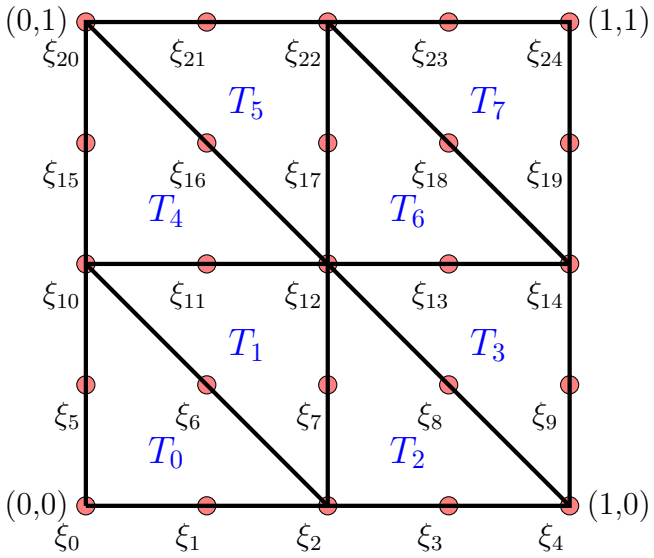
be the length of an edge in $\mathcal{T}_h$. Then, the number of elements is $2M^2$ and the number of nodes is $N = (kM+1)^2$. The data for a given triangulation $\mathcal{T}_h$ are described in $N \times 2$ matrix **c4n**, $2M^2 \times 3$ matrix **n4e**, N_D dimensional vector **n4db** and $2M^2 \times (k+1)(k+2)/2$ matrix **ind4e**. Here N_D is the number of nodes on Dirichlet boundary. For example if $\Omega = (0,\ 1)^2$, $M = 2$ and $k = 1$, data are stored as follows.



$$
\begin{aligned}
\text{c4n} &= [[0,0],\ [1/2,0],\ [1,0],\ [0,1/2],\ [1/2,1/2],\ [1,1/2],\ [0,1],\ [1/2,1],\ [1,1]],\\
\text{n4e} &= [[1,3,0],\ [3,1,4],\ [2,4,1],\ [4,2,5],\ [4,6,3],\ [6,4,7],\ [5,7,4],\ [7,5,8]],\\
\text{ind4e} &= [[0,1,3],\ [4,3,1],\ [1,2,4],\ [5,4,2],\ [3,4,6],\ [7,6,4],\ [4,5,7],\ [8,7,5]],\\
\text{n4db} &= [0,1,2,3,5,6,7,8]
\end{aligned}
$$

Here the size of n4e is the same as the size of ind4e. However their elements are different each other. The matrix n4e has 3 columns and the components in each row are corresponding to the vertex nodes in the corresponding element. These nodes have usually in a counterclockwise orientation and the first two vertices are the end-points of the longest edge. On the other hand, ind4e has $(k+1)(k+2)/2$ columns and the components in each column are corresponding to all nodes in the corresponding element. These nodes are ordered from left to right and from bottom to top, and the starting node is in the third row of n4e.

If $\Omega = (0,\ 1)^2$, $M = 2$ and $k = 2$, data are stored as follows.



$$\begin{aligned}
\text{c4n} &= [[0,0],\ [1/4,0],\ [1/2,0],\ [3/4,0],\ [1,0],\ [0,1/4],\ [1/4,1/4],\ [1/2,1/4],\ [3/4,1/4],\\
&\quad [1,1/4],\ [0,1/2],\ [1/4,1/2],\ [1/2,1/2],\ [3/4,1/2],\ [1,1/2],\ [0,3/4],\ [1/4,3/4],\\
&\quad [1/2,3/4],\ [3/4,3/4],\ [1,3/4],\ [0,1],\ [1/4,1],\ [1/2,1],\ [3/4,1],\ [1,1]],\\
\text{n4e} &= [[2,10,0],\ [10,2,12],\ [4,12,2],\ [12,4,14],\\
&\quad [12,20,10],\ [20,12,22],\ [14,22,12],\ [22,14,24]]\\
\text{ind4e} &= [[0,1,2,5,6,10],\ [12,11,10,7,6,2],\ [2,3,4,7,8,12],\ [14,13,12,9,8,4],\\
&\quad [10,11,12,15,16,20],\ [22,21,20,17,16,12],\ [12,13,14,17,18,22],\\
&\quad [24,23,22,19,18,14]]\\
\text{n4db} &= [0,1,2,3,5,9,10,14,15,19,20,21,22,23,24].
\end{aligned}$$

In this case, each element has an extra point except vertex nodes. Thus the size of ind4e is larger than that of n4e. Here, the components in n4e and ind4e are ordered to be the same as n4e and ind4e for $k = 1$, respectively.

mesh_fem_2d The following python code generates an uniform triangular mesh on the domain $(x_\ell, x_r) \times (y_\ell, y_r)$ in 2D with $2M_x$ elements along x-direction and $2M_y$ elements along y-direction. Also this code returns an index matrix for continuous k-th order polynomial approximations.

```
def mesh_fem_2d(xl, xr, yl, yr, Mx, My, k):
    tmp = np.array([[np.arange(0,k*Mx,k) + (k*Mx+1)*i*k] for i in range(My)]).flatten()
    tmp2 = np.array([[np.arange(k,k*Mx+1,k) + (k*Mx+1)*(i+1)*k] for i in range(My)]).flatten()
    tmp3 = list(np.concatenate([list(j*(Mx*k+1)+np.arange(0,k+1-j)) for j in range(k+1)]))
    ind4e = np.array([tmp[np.int32(i/2)]+tmp3 if i % 2 == 0
```

```
    else tmp2[np.int32((i-1)/2)] - tmp3 for i in range(2*Mx*My)])
n4e = np.array([[ind4e[i,k], ind4e[i,np.int32((k+1)*(k+2)/2-1)], ind4e[i,0]]
    for i in range(ind4e.shape[0])])
n4db = np.concatenate([[i for i in range(0,k*Mx+1)],
    [i for i in range(2*k*Mx+1,(k*Mx+1)*(k*My+1),(k*Mx+1))],
    [i for i in range((k*Mx+1)*(k*My+1)-2,k*My*(k*Mx+1)-1,-1)],
    [i for i in range((k*My-1)*(k*Mx+1),k*Mx,-(k*Mx+1))]])
x = np.linspace(xl,xr,k*Mx+1)
y = np.linspace(yl,yr,k*My+1)
x = np.tile(x, (1,k*My+1)).flatten()
y = np.tile(y,(k*Mx+1,1)).T.flatten()
c4n = np.array([[x[i], y[i]] for i in range(len(x))])
return (c4n,n4e,n4db,ind4e)
```

2.3 Basis functions of V_h^k and Numerical solution

This section is almost the same as that for rectangular elements (see, Section 2.3). Thus, a few differences are stated here.

Let $\psi_i^n(x)$ be the i-th basis function in the n-th element. Then, the following relations hold

$$\psi_i^n(\boldsymbol{x}) = 0 \quad \text{if } x \in \Omega \setminus T_n$$
$$\sum_{i=0}^{N_n-1} \psi_i^n(\boldsymbol{x}) = 1 \quad \forall x \in T_n,$$

where $N_n = (k+1)(k+2)/2$ is the number of nodes in T_n. Figure 2.2 shows node numbering in T_n.

The numerical solution can be written locally because the solution is piecewise polynomial function:

$$u_h\Big|_{T_n} = \sum_{i=0}^{N_n-1} u_i^n \psi_i^n \tag{2.8}$$

where $u_i^j = u_h(\xi_i^j)$. Then, the gradient of the local solution is easily obtained from (2.8)

$$\nabla u_h\Big|_{T_n} = \sum_{i=0}^{N_n-1} u_i^n \nabla\psi_i^n. \tag{2.9}$$

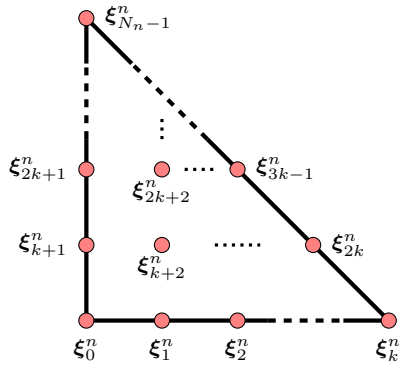


Figure 2.2: Node numbering in the n-th triangular element

2.4 Mass matrix and Stiffness matrix

Similar to Chapter 1, for a test function $\psi_i(\boldsymbol{x}) \in V_h^k$, the variational formulation (2.4) can be rewritten as

$$\sum_{j=0}^{N-1} u_j \int_\Omega \nabla\psi_i \cdot \nabla\psi_j \, d\boldsymbol{x} = \int_\Omega f\psi_i \, d\boldsymbol{x}. \tag{2.10}$$

For all basis functions in V_h^k, we have the following finite element system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b} \tag{2.11}$$

where

$$(\boldsymbol{A})_{ij} = \int_\Omega \nabla\psi_{i-1} \cdot \nabla\psi_{j-1} \, d\boldsymbol{x} \tag{2.12a}$$

$$(\boldsymbol{b})_i = \int_\Omega f\psi_{i-1} \, d\boldsymbol{x} \tag{2.12b}$$

$$(\boldsymbol{u})_j = u_{j-1}. \tag{2.12c}$$

Here, the matrix $\boldsymbol{A}$ is called the global stiffness matrix and the right-hand side b is called the load vector. In order to compute the load vector, f is replaced with the interpolate function $\mathcal{I}f(\boldsymbol{x})$ in general. Thus, FE system (2.11) can be rewritten as

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{M}\boldsymbol{f} \tag{2.13}$$

where

$$(\boldsymbol{M})_{ij} = \int_\Omega \psi_{i-1}\psi_{j-1} \, d\boldsymbol{x} \tag{2.14a}$$

$$(\boldsymbol{f})_j = f_{j-1}. \tag{2.14b}$$

Here, the matrix $\boldsymbol{M}$ is called the global mass matrix. For the accurate computation, f can be replaced by the L^2-orthogonal projection $\pi f(\boldsymbol{x})$.

The global stiffness matrix and the global mass matrix can be assembled by using local basis functions.

$$\boldsymbol{A} = \sum_{n=1}^{M} \boldsymbol{A}_{T_n}, \qquad \boldsymbol{M} = \sum_{n=1}^{M} \boldsymbol{M}_{T_n} \tag{2.15}$$

where N_n by N_n matrices $\boldsymbol{A}_{T_n}$ and $\boldsymbol{M}_{T_n}$ are defined as

$$(\boldsymbol{A}_{T_n})_{ij} = \int_{T_n} \nabla\psi_{i-1}^n(x) \cdot \nabla\psi_{j-1}^n \, d\boldsymbol{x} \tag{2.16}$$

$$(\boldsymbol{M}_{T_n})_{ij} = \int_{T_n} \psi_{i-1}^n \psi_{j-1}^n \, d\boldsymbol{x} \tag{2.17}$$

where $1 \le i,\ j \le N_n$. Here $\boldsymbol{A}_{T_n}$ and $\boldsymbol{M}_{T_n}$ are called the local stiffness matrix and the local mass matrix, respectively.

In order to compute gradient, we now introduce differentiation matrices $\boldsymbol{Dx}$ and $\boldsymbol{Dy}$ such that

$$(\boldsymbol{Dx})_{ij} = \frac{\partial\psi_{j-1}}{\partial x}(\boldsymbol{\xi}_{i-1}), \qquad (\boldsymbol{Dy})_{ij} = \frac{\partial\psi_{j-1}}{\partial y}(\boldsymbol{\xi}_{i-1}). \tag{2.18}$$

Clearly, $\frac{\partial\psi_j}{\partial x}(\boldsymbol{\xi}_i),\ \frac{\partial\psi_j}{\partial y}(\boldsymbol{\xi}_i) \in V_h^{k-1} \subset V_h^k$ because $\psi_i(\boldsymbol{x}) \in V_h^k$. Thus,

$$\frac{\partial}{\partial x}\psi_{i-1}(\boldsymbol{x}) = \sum_{j=0}^{N-1} \frac{\partial\psi_{i-1}}{\partial x}(\boldsymbol{\xi}_j)\psi_j(\boldsymbol{x}) = (\boldsymbol{Dx}^t)_i\boldsymbol{\psi}$$

$$\frac{\partial}{\partial y}\psi_{i-1}(\boldsymbol{x}) = \sum_{j=0}^{N-1} \frac{\partial\psi_{i-1}}{\partial y}(\boldsymbol{\xi}_j)\psi_j(\boldsymbol{x}) = (\boldsymbol{Dy}^t)_i\boldsymbol{\psi}$$

where $(\boldsymbol{Dx}^t)_i$ and $(\boldsymbol{Dy}^t)_i$ are the i-th row of the matrices $\boldsymbol{Dx}^t$ and $\boldsymbol{Dy}^t$, respectively, i.e. $(\boldsymbol{Dx}^t)_i = [\frac{\partial\psi_{i-1}}{\partial x}(\xi_0)\ \cdots\ \frac{\partial\psi_{i-1}}{\partial x}(\xi_{N-1})]$, $(Dy^t)_i = [\frac{\partial\psi_{i-1}}{\partial y}(\xi_0)\ \cdots\ \frac{\partial\psi_{i-1}}{\partial y}(\xi_{N-1})]$, and $\boldsymbol{\psi} = [\psi_0(x)\ \cdots\ \psi_{N-1}(x)]^t$. Similar to the stiffness and mass matrices, $\boldsymbol{Dx}$ and $\boldsymbol{Dy}$ can be assembled by the local differentiation matrix

$$\boldsymbol{Dx} = \sum_{n=0}^{2M^2-1} \boldsymbol{Dx}_{T_n}, \qquad (\boldsymbol{Dx}_{T_n})_{ij} = \frac{\partial\psi_{j-1}^n}{\partial x}(\boldsymbol{\xi}_{i-1}^n)$$

$$\boldsymbol{Dy} = \sum_{n=0}^{2M^2-1} \boldsymbol{Dy}_{T_n}, \qquad (\boldsymbol{Dy}_{T_n})_{ij} = \frac{\partial\psi_{j-1}^n}{\partial y}(\boldsymbol{\xi}_{i-1}^n).$$

Thus, the gradient of the local basis function $\psi_i^n(\boldsymbol{x})$ is written as

$$\begin{aligned}\nabla\psi_{i-1}^n(\boldsymbol{x}) &= \Big(\sum_{j=0}^{N_n-1}\frac{\partial\psi_{i-1}^n}{dx}(\boldsymbol{\xi}_j^n)\psi_j^n(\boldsymbol{x}),\ \sum_{j=0}^{N_n-1}\frac{\partial\psi_{i-1}^n}{dy}(\boldsymbol{\xi}_j^n)\psi_j^n(\boldsymbol{x})\Big)\\ &= \Big((\boldsymbol{D}\boldsymbol{x}_{T_n}^t)_i\boldsymbol{\psi}^n,\ (\boldsymbol{D}\boldsymbol{y}_{T_n}^t)_i\boldsymbol{\psi}^n\Big)\end{aligned} \tag{2.19}$$

where $(\boldsymbol{D}\boldsymbol{x}_{T_n}^t)_i$ and $(Dy_{T_n}^t)_i$ are the i-th row of the matrices $(\boldsymbol{D}\boldsymbol{x}_{T_n}^t)_i$ and $(Dy_{T_n}^t)_i$, respectively, and $\boldsymbol{\psi}^n = [\psi_0^n(\boldsymbol{x})\ \cdots\ \psi_{N_n-1}^n(\boldsymbol{x})]^t$. Then, we have

$$\begin{aligned}\nabla u_h(\boldsymbol{\xi}_m) &= \sum_{i=0}^{N-1}u_i\nabla\psi_i(\boldsymbol{\xi}_m) = \sum_{i=0}^{N-1}u_i\sum_{j=0}^{N-1}\nabla\psi_i(\boldsymbol{\xi}_j)\psi_j(\boldsymbol{\xi}_m) = \sum_{i=0}^{N-1}u_i\nabla\psi_i(\boldsymbol{\xi}_m)\\ &= \Big((\boldsymbol{D}\boldsymbol{x})_m\boldsymbol{u},\ (\boldsymbol{D}\boldsymbol{y})_m\boldsymbol{u}\Big)\\ \nabla u_h(\boldsymbol{\xi}_m^n) &= \sum_{i=0}^{N_n-1}u_i^n\nabla\psi_i^n(\boldsymbol{\xi}_m) = \sum_{i=0}^{N_n-1}u_i^n\sum_{j=0}^{N_n-1}\nabla\psi_i^n(\boldsymbol{\xi}_j)\psi_j(\boldsymbol{\xi}_m^n) = \sum_{i=0}^{N_n-1}u_i^n\nabla\psi_i^n(\boldsymbol{\xi}_m^\ell)\\ &= \Big((\boldsymbol{D}\boldsymbol{x}_{T_n})_m\boldsymbol{u},\ (\boldsymbol{D}\boldsymbol{y}_{T_n})_m\boldsymbol{u}\Big).\end{aligned}$$

We can compute the local stiffness and mass matrices by using the affine mapping. For convenience, we will use the notations M and S instead of $\boldsymbol{M}_{T_n}$ and $\boldsymbol{A}_{T_n}$ for a fixed element T_n, respectively. Also, ψ and $\widetilde{\psi}$ are basis functions on the element T_n and the reference interval T_R, respectively. Then,

$$(\boldsymbol{M})_{ij} = \int_{R_n}\psi_{i-1}(\boldsymbol{x})\psi_{j-1}(\boldsymbol{x})\ d\boldsymbol{x} = J\int_{R_R}\widetilde{\psi}_{i-1}(\boldsymbol{r})\widetilde{\psi}_{j-1}(\boldsymbol{r})\ dr = J(\boldsymbol{M}_R)_{ij} \tag{2.20}$$

where $\boldsymbol{M}_R$ is the mass matrix on T_R. By (2.7) and (2.9),

$$
\begin{aligned}
(\boldsymbol{S})_{ij} &= \int_{T_n} \nabla\psi_{i-1}(\boldsymbol{x}) \cdot \nabla\psi_{j-1}(\boldsymbol{x})\ d\boldsymbol{x} \\
&= \int_{T_n} \frac{\partial}{\partial x}\psi_{i-1}(\boldsymbol{x}) \frac{\partial}{\partial x}\nabla\psi_{j-1}(\boldsymbol{x})\ d\boldsymbol{x} + \int_{T_n} \frac{\partial}{\partial y}\psi_{i-1}(\boldsymbol{x}) \frac{\partial}{\partial y}\nabla\psi_{j-1}(\boldsymbol{x})\ d\boldsymbol{x} \\
&= \int_{T_n} \Big(\sum_{\ell=0}^{N_n-1} \frac{\partial\psi_{i-1}}{\partial x}(\boldsymbol{\xi}_\ell)\psi_\ell(\boldsymbol{x}) \Big) \Big(\sum_{m=0}^{N_n-1} \frac{\partial\psi_{j-1}}{\partial x}(\boldsymbol{\xi}_m)\psi_m(\boldsymbol{x}) \Big)\ d\boldsymbol{x} \\
&\qquad + \int_{T_n} \Big(\sum_{\ell=0}^{N_n-1} \frac{\partial\psi_{i-1}}{\partial y}(\boldsymbol{\xi}_\ell)\psi_\ell(\boldsymbol{x}) \Big) \Big(\sum_{m=0}^{N_n-1} \frac{\partial\psi_{j-1}}{\partial y}(\boldsymbol{\xi}_m)\psi_m(\boldsymbol{x}) \Big)\ d\boldsymbol{x} \qquad (2.21) \\
&= J\int_{T_R} \Big[\sum_{\ell=0}^{N_n-1} \Big(r_x \frac{\partial\widetilde{\psi}_{i-1}}{\partial r}(\widetilde{\boldsymbol{\xi}}_\ell) + s_x \frac{\partial\widetilde{\psi}_{i-1}}{\partial s}(\widetilde{\boldsymbol{\xi}}_\ell) \Big) \widetilde{\psi}_\ell(\boldsymbol{r}) \Big] \Big[\sum_{m=0}^{N_n-1} \Big(r_x \frac{\partial\widetilde{\psi}_{j-1}}{\partial r}(\widetilde{\boldsymbol{\xi}}_m) + s_x \frac{\partial\widetilde{\psi}_{j-1}}{\partial s}(\widetilde{\boldsymbol{\xi}}_m) \Big) \widetilde{\psi}_m(\boldsymbol{r}) \Big] d\boldsymbol{r} \\
&\qquad + J\int_{T_R} \Big[\sum_{\ell=0}^{N_n-1} \Big(r_y \frac{\partial\widetilde{\psi}_{i-1}}{\partial r}(\widetilde{\boldsymbol{\xi}}_\ell) + s_y \frac{\partial\widetilde{\psi}_{i-1}}{\partial s}(\widetilde{\boldsymbol{\xi}}_\ell) \Big) \widetilde{\psi}_\ell(\boldsymbol{r}) \Big] \Big[\sum_{m=0}^{N_n-1} \Big(r_y \frac{\partial\widetilde{\psi}_{j-1}}{\partial r}(\widetilde{\boldsymbol{\xi}}_m) + s_y \frac{\partial\widetilde{\psi}_{j-1}}{\partial s}(\widetilde{\boldsymbol{\xi}}_m) \Big) \widetilde{\psi}_m(\boldsymbol{r}) \Big] d\boldsymbol{r} \\
&= J\Big[(r_x^2 + r_y^2)(\boldsymbol{S}_R^{rr})_{ij} + (r_x s_x + r_y s_y) \big] \big((\boldsymbol{S}_R^{rs})_{ij} + (\boldsymbol{S}_R^{sr})_{ij} \big) + (s_x^2 + s_y^2)(\boldsymbol{S}_R^{ss})_{ij} \Big]
\end{aligned}
$$

and

$$
\begin{aligned}
(\boldsymbol{S}_R^{rr})_{ij} &= \int_{T_R} \Big((\boldsymbol{D}\boldsymbol{r}_R^t)_i \widetilde{\boldsymbol{\psi}} \Big) \Big((\boldsymbol{D}\boldsymbol{r}_R^t)_j \widetilde{\boldsymbol{\psi}} \Big)\ d\boldsymbol{r} \\
&= (\boldsymbol{D}\boldsymbol{r}_R^t)_i \boldsymbol{M}_R (\boldsymbol{D}\boldsymbol{r}_R)_j \qquad (2.22a) \\
&= (\boldsymbol{D}\boldsymbol{r}_R^t \boldsymbol{M}_R \boldsymbol{D}\boldsymbol{r}_R)_{ij} \\
(\boldsymbol{S}_R^{rs})_{ij} &= \int_{T_R} \Big((\boldsymbol{D}\boldsymbol{r}_R^t)_i \widetilde{\boldsymbol{\psi}} \Big) \Big((\boldsymbol{D}\boldsymbol{s}_R^t)_j \widetilde{\boldsymbol{\psi}} \Big)\ d\boldsymbol{r} \\
&= (\boldsymbol{D}\boldsymbol{r}_R^t)_i \boldsymbol{M}_R (\boldsymbol{D}\boldsymbol{s}_R)_j \qquad (2.22b) \\
&= (\boldsymbol{D}\boldsymbol{r}_R^t \boldsymbol{M}_R \boldsymbol{D}\boldsymbol{s}_R)_{ij} \\
(\boldsymbol{S}_R^{sr})_{ij} &= \int_{T_R} \Big((\boldsymbol{D}\boldsymbol{s}_R^t)_i \widetilde{\boldsymbol{\psi}} \Big) \Big((\boldsymbol{D}\boldsymbol{r}_R^t)_j \widetilde{\boldsymbol{\psi}} \Big)\ d\boldsymbol{r} \\
&= (\boldsymbol{D}\boldsymbol{s}_R^t)_i \boldsymbol{M}_R (\boldsymbol{D}\boldsymbol{r}_R)_j \qquad (2.22c) \\
&= (\boldsymbol{D}\boldsymbol{s}_R^t \boldsymbol{M}_R \boldsymbol{D}\boldsymbol{r}_R)_{ij} \\
(\boldsymbol{S}_R^{ss})_{ij} &= \int_{T_R} \Big((\boldsymbol{D}\boldsymbol{s}_R^t)_i \widetilde{\boldsymbol{\psi}} \Big) \Big((\boldsymbol{D}\boldsymbol{s}_R^t)_j \widetilde{\boldsymbol{\psi}} \Big)\ d\boldsymbol{r} \\
&= (\boldsymbol{D}\boldsymbol{s}_R^t)_i \boldsymbol{M}_R (\boldsymbol{D}\boldsymbol{s}_R)_j \qquad (2.22d) \\
&= (\boldsymbol{D}\boldsymbol{s}_R^t \boldsymbol{M}_R \boldsymbol{D}\boldsymbol{s}_R)_{ij}
\end{aligned}
$$

where $\boldsymbol{S}_R^{rr}$, $\boldsymbol{S}_R^{rs}$, $\boldsymbol{S}_R^{sr}$ and $\boldsymbol{S}_R^{ss}$ are the local stiffness matrices on T_R, $\boldsymbol{D}\boldsymbol{r}_R$ and $\boldsymbol{D}\boldsymbol{s}_R$ is the differentiation matrices on T_R, and $\widetilde{\boldsymbol{\psi}} = [\widetilde{\psi}_0(\boldsymbol{r}) \ \cdots \widetilde{\psi}_{N_n-1}(\boldsymbol{r})]^t$. Thus, the local mass and

stiffness matrices are obtained from the reference mass and stiffness matrices by using affine mapping.

$$\boldsymbol{M} = J\boldsymbol{M}_R, \qquad \boldsymbol{S} = J\Big((r_x^2 + r_y^2)\boldsymbol{S}_R^{rr} + (r_x s_x + r_y s_y)(\boldsymbol{S}_R^{rs} + \boldsymbol{S}_R^{sr}) + (s_x^2 + s_y^2)\boldsymbol{S}_R^{ss}\Big). \tag{2.23}$$

The following lemma is helpful to compute the local matrices.

Lemma 2.1. *For a given interval $T = conv\{\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2\}$ with barycentric coordinates $\lambda_0,\ \lambda_1$ and λ_2, it holds for $a,\ b,\ c \in \mathbb{N}_0$ that*

$$\int_T \lambda_0^a \lambda_1^b \lambda_2^c \, d\boldsymbol{x} = 2|T| \frac{a!\ b!\ c!}{(a+b+c+2)!} \tag{2.24}$$

where $|T|$ is area of T.

Now we compute the reference matrices for the linear ($k = 1$) and quadratic ($k = 2$) approximations. For the k-th order approximation, the matrices are obtained similarly.

P_1 matrices The basis functions of $P_1(T_R)$ are the same as the barycentric coordinates with a simple permutation. Thus the basis functions and their gradients are

$$\begin{aligned}
\widetilde{\psi}_0(\boldsymbol{r}) &= \widetilde{\lambda}_2(\boldsymbol{r}), \qquad & \nabla\widetilde{\psi}_0(\boldsymbol{r}) &= \Big(-\frac{1}{2},\ -\frac{1}{2}\Big),\\
\widetilde{\psi}_1(\boldsymbol{r}) &= \widetilde{\lambda}_0(\boldsymbol{r}), \qquad & \nabla\widetilde{\psi}_1(\boldsymbol{r}) &= \Big(\frac{1}{2},\ 0\Big),\\
\widetilde{\psi}_2(\boldsymbol{r}) &= \widetilde{\lambda}_1(\boldsymbol{r}), \qquad & \nabla\widetilde{\psi}_2(\boldsymbol{r}) &= \Big(0,\ \frac{1}{2}\Big).
\end{aligned}$$

Then (2.17) and (2.24) yield

$$\boldsymbol{M}_R = \frac{1}{6}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \tag{2.25}$$

The differentiation matrices $\boldsymbol{Dr}_R$ and $\boldsymbol{Ds}_R$ are obtained from (2.18)

$$\boldsymbol{Dr}_R = \frac{1}{2}\begin{pmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{pmatrix} \tag{2.26a}$$

$$\boldsymbol{Ds}_R = \frac{1}{2}\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}. \tag{2.26b}$$

Therefore, the local stiffness matrices are obtained from (2.22)

$$\boldsymbol{S}_R^{rr} = \boldsymbol{Dr}_R^t \boldsymbol{M}_R \boldsymbol{Dr}_R = \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.27a}$$

$$\boldsymbol{S}_R^{rs} = \boldsymbol{Dr}_R^t \boldsymbol{M}_R \boldsymbol{Ds}_R = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.27b}$$

$$\boldsymbol{S}_R^{sr} = \boldsymbol{Ds}_R^t \boldsymbol{M}_R \boldsymbol{Dr}_R = \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix} \tag{2.27c}$$

$$\boldsymbol{S}_R^{ss} = \boldsymbol{Ds}_R^t \boldsymbol{M}_R \boldsymbol{Ds}_R = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}. \tag{2.27d}$$

P_2 matrices Similar to the P_1 matrices, the basis functions in $P_2(R_R)$ are obtained from the 1D quadratic basis functions and barycentric coordinates such that

$$\begin{aligned}
&\widetilde{\psi}_0(\boldsymbol{r}) = \widetilde{\lambda}_2(\boldsymbol{r})(2\widetilde{\lambda}_2(\boldsymbol{r}) - 1), && \nabla\widetilde{\psi}_0(\boldsymbol{r}) = \Big(-2\widetilde{\lambda}_2(\boldsymbol{r}) + \frac{1}{2},\ -2\widetilde{\lambda}_2(\boldsymbol{r}) + \frac{1}{2}\Big), \\
&\widetilde{\psi}_1(\boldsymbol{r}) = 4\widetilde{\lambda}_0(\boldsymbol{r})\widetilde{\lambda}_2(\boldsymbol{r}), && \nabla\widetilde{\psi}_1(\boldsymbol{r}) = \Big(2\widetilde{\lambda}_2(\boldsymbol{r}) - 2\widetilde{\lambda}_0(\boldsymbol{r}),\ -2\widetilde{\lambda}_0(\boldsymbol{r})\Big), \\
&\widetilde{\psi}_2(\boldsymbol{r}) = \widetilde{\lambda}_0(\boldsymbol{r})(2\widetilde{\lambda}_0(\boldsymbol{r}) - 1), && \nabla\widetilde{\psi}_2(\boldsymbol{r}) = \Big(2\widetilde{\lambda}_0(\boldsymbol{r}) - \frac{1}{2},\ 0\Big), \\
&\widetilde{\psi}_3(\boldsymbol{r}) = 4\widetilde{\lambda}_1(\boldsymbol{r})\widetilde{\lambda}_2(\boldsymbol{r}), && \nabla\widetilde{\psi}_3(\boldsymbol{r}) = \Big(-2\widetilde{\lambda}_1(\boldsymbol{r}),\ 2\widetilde{\lambda}_1(\boldsymbol{r}) - 2\widetilde{\lambda}_2(\boldsymbol{r})\Big), \\
&\widetilde{\psi}_4(\boldsymbol{r}) = 4\widetilde{\lambda}_0(\boldsymbol{r})\widetilde{\lambda}_1(\boldsymbol{r}), && \nabla\widetilde{\psi}_4(\boldsymbol{r}) = \Big(2\widetilde{\lambda}_1(\boldsymbol{r}),\ 2\widetilde{\lambda}_0(\boldsymbol{r})\Big), \\
&\widetilde{\psi}_5(\boldsymbol{r}) = \widetilde{\lambda}_1(\boldsymbol{r})(2\widetilde{\lambda}_1(\boldsymbol{r}) - 1), && \nabla\widetilde{\psi}_5(\boldsymbol{r}) = \Big(0,\ 2\widetilde{\lambda}_1(\boldsymbol{r}) - \frac{1}{2}\Big),
\end{aligned}$$

Then, we have the mass matrix

$$\boldsymbol{M}_R = \frac{1}{90}\begin{pmatrix} 6 & 0 & -1 & 0 & -4 & -1 \\ 0 & 32 & 0 & 16 & 16 & -4 \\ -1 & 0 & 6 & -4 & 0 & -1 \\ 0 & 16 & -4 & 32 & 16 & 0 \\ -4 & 16 & 0 & 16 & 32 & 0 \\ -1 & -4 & -1 & 0 & 0 & 6 \end{pmatrix}, \tag{2.28}$$

the differentiation matrices

$$\boldsymbol{Dr}_R = \frac{1}{2}\begin{pmatrix} -3 & 4 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 3 & 0 & 0 & 0 \\ -1 & 2 & -1 & -2 & 2 & 0 \\ 1 & -2 & 1 & -2 & 2 & 0 \\ 1 & 0 & -1 & -4 & 4 & 0 \end{pmatrix}, \tag{2.29a}$$

$$\boldsymbol{Ds}_R = \frac{1}{2}\begin{pmatrix} -3 & 0 & 0 & 4 & 0 & -1 \\ -1 & -2 & 0 & 2 & 2 & -1 \\ 1 & -4 & 0 & 0 & 4 & -1 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & -2 & 0 & -2 & 2 & 1 \\ 1 & 0 & 0 & -4 & 0 & 3 \end{pmatrix}, \tag{2.29b}$$

and the stiffness matrices

$$\boldsymbol{S}_R^{rr} = \frac{1}{6}\begin{pmatrix} 3 & -4 & 1 & 0 & 0 & 0 \\ -4 & 8 & -4 & 0 & 0 & 0 \\ 1 & -4 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & -8 & 0 \\ 0 & 0 & 0 & -8 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.30a}$$

$$\boldsymbol{S}_R^{rs} = \frac{1}{6}\begin{pmatrix} 3 & 0 & 0 & -4 & 0 & 1 \\ -4 & 4 & 0 & 4 & -4 & 0 \\ 1 & -4 & 0 & 0 & 4 & -1 \\ 0 & 4 & 0 & 4 & -4 & -4 \\ 0 & -4 & 0 & -4 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.30b}$$

$$\boldsymbol{S}_R^{sr} = \frac{1}{6}\begin{pmatrix} 3 & -4 & 1 & 0 & 0 & 0 \\ 0 & 4 & -4 & 4 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 4 & 0 & 4 & -4 & 0 \\ 0 & -4 & 4 & -4 & 4 & 0 \\ 1 & 0 & -1 & -4 & 4 & 0 \end{pmatrix}, \tag{2.30c}$$

$$\boldsymbol{S}_R^{ss} = \frac{1}{6}\begin{pmatrix} 3 & 0 & 0 & -4 & 0 & 1 \\ 0 & 8 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 8 & 0 & -4 \\ 0 & -8 & 0 & 0 & 8 & 0 \\ 1 & 0 & 0 & -4 & 0 & 3 \end{pmatrix}. \tag{2.30d}$$

get_matrices_2d_triangle The following python code generates the mass matrix $\boldsymbol{M}_R$, the stiffness matrices $\boldsymbol{S}_R^{rr}$, $\boldsymbol{S}_R^{rs}$, $\boldsymbol{S}_R^{sr}$, $\boldsymbol{S}_R^{ss}$ and the differentiation matrices $\boldsymbol{Dr}_R$, $\boldsymbol{Ds}_R$ for continuous k-th order polynomial approximations on the reference triangle T_R.

```
def get_matrices_2d_triangle(k=1):
    if k == 1:
        M_R = np.array([[2, 1, 1], [1, 2, 1], [1, 1, 2]])/6.
        Srr_R = np.array([[1, -1, 0], [-1, 1, 0], [0, 0, 0]])/2.
        Srs_R = np.array([[1, 0, -1], [-1, 0, 1], [0, 0, 0]])/2.
        Ssr_R = np.array([[1, -1, 0], [0, 0, 0], [-1, 1, 0]])/2.
        Sss_R = np.array([[1, 0, -1], [0, 0, 0], [-1, 0, 1]])/2.
        Dr_R = np.array([[-1, 1, 0], [-1, 1, 0], [-1, 1, 0]])/2.
        Ds_R = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])/2.
    elif k == 2:
        M_R = np.array([[6, 0, -1, 0, -4, -1], [0, 32, 0, 16, 16, -4], [-1, 0, 6, -4, 0, -1],
            [0, 16, -4, 32, 16, 0], [-4, 16, 0, 16, 32, 0], [-1, -4, -1, 0, 0, 6]])/90.
        Srr_R = np.array([[3 ,-4, 1, 0, 0, 0], [-4, 8, -4, 0, 0, 0], [1, -4, 3, 0, 0, 0],
            [0, 0, 0, 8, -8, 0], [0, 0, 0, -8, 8, 0], [0, 0, 0, 0, 0, 0]])/6.
        Srs_R = np.array([[3, 0, 0, -4, 0, 1], [-4, 4, 0, 4, -4, 0], [1, -4, 0, 0, 4, -1],
            [0, 4, 0, 4, -4, -4], [0, -4, 0, -4, 4, 4], [0, 0, 0, 0, 0, 0]])/6.
        Ssr_R = np.array([[3, -4, 1, 0, 0, 0], [0, 4, -4, 4, -4, 0], [0, 0, 0, 0, 0, 0],
            [-4, 4, 0, 4, -4, 0], [0, -4, 4, -4, 4, 0], [1, 0, -1, -4, 4, 0]])/6.
        Sss_R = np.array([[3, 0, 0, -4, 0, 1], [0, 8, 0, 0, -8, 0], [0, 0, 0, 0, 0, 0],
            [-4, 0, 0, 8, 0, -4], [0, -8, 0, 0, 8, 0], [1, 0, 0, -4, 0, 3]])/6.
        Dr_R = np.array([[-3, 4, -1, 0, 0, 0], [-1, 0, 1, 0, 0, 0], [1, -4, 3, 0, 0, 0],
            [-1, 2, -1, -2, 2, 0], [1, -2, 1, -2, 2, 0], [1, 0, -1, -4, 4, 0]])/2.
        Ds_R = np.array([[-3, 0, 0, 4, 0, -1], [-1, -2, 0, 2, 2, -1], [1, -4, 0, 0, 4, -1],
            [-1, 0, 0, 0, 0, 1], [1, -2, 0, -2, 2, 1], [1, 0, 0, -4, 0, 3]])/2.
    elif: ...
    return (M_R, Srr_R, Srs_R, Ssr_R, Sss_R, Dr_R, Ds_R)
```

2.4.1 Matlab codes in 2D with triangular elements

Now we are ready to assemble the global stiffness matrix A and the global load vector b in (2.11). In the matlab code, A and b are assembled by using the local stiffness matrix (2.21) and the local mass matrix (2.20).

fem_for_poisson_2d The following Matlab code solves the Poisson problem. In order to use this code, mesh information (c4n, n4e, n4db, ind4e), matrices ($\boldsymbol{M}_R, \boldsymbol{S}_R^{rr}, \boldsymbol{S}_R^{rs}, \boldsymbol{S}_R^{sr}, \boldsymbol{S}_R^{ss}$), the source f, and the boundary condition u_D. Then the results of this code are the numerical solution u, the global stiffness matrix A, the global load vector b and the freenodes.

```
def fem_for_poisson_2d(c4n,n4e,n4db,ind4e,M_R,Srr_R,Srs_R,Ssr_R,Sss_R,f,u_D):
    number_of_nodes = c4n.shape[0]
```

```
    number_of_elements = n4e.shape[0]
    b = np.zeros(number_of_nodes)
    u = np.zeros(number_of_nodes)
    xr = np.array([(c4n[n4e[i,0],0] - c4n[n4e[i,2],0])/2. for i in range(number_of_elements)])
    yr = np.array([(c4n[n4e[i,0],1] - c4n[n4e[i,2],1])/2. for i in range(number_of_elements)])
    xs = np.array([(c4n[n4e[i,1],0] - c4n[n4e[i,2],0])/2. for i in range(number_of_elements)])
    ys = np.array([(c4n[n4e[i,1],1] - c4n[n4e[i,2],1])/2. for i in range(number_of_elements)])
    J = xr*ys - xs*yr
    rx=ys/J
    ry=-xs/J
    sx=-yr/J
    sy=xr/J
    Aloc = np.array([J[i]*((rx[i]**2+ry[i]**2)*Srr_R.flatten()
        + (rx[i]*sx[i]+ry[i]*sy[i])*(Srs_R.flatten()+Ssr_R.flatten())
        + (sx[i]**2+sy[i]**2)*Sss_R.flatten()) for i in range(number_of_elements)])
    for i in range(number_of_elements):
        b[ind4e[i]] += J[i]*np.matmul(M_R, f(c4n[ind4e[i]]))
    row_ind = np.tile(ind4e.flatten(),(ind4e.shape[1],1)).T.flatten()
    col_ind = np.tile(ind4e,(1,ind4e.shape[1])).flatten()
    A_COO = coo_matrix((Aloc.flatten(), (row_ind, col_ind)),
        shape=(number_of_nodes, number_of_nodes))
    A = A_COO.tocsr()
    dof = np.setdiff1d(range(0,number_of_nodes), n4db)
    u[dof] = spsolve(A[dof, :].tocsc()[:, dof].tocsr(), b[dof])
    return u
```

compute_error_fem_2d The following Matlab code computes the semi H1 error between the exact solution and the numerical solution.

```
def compute_error_fem_2d(c4n,n4e,ind4e,M_R,Dr_R,Ds_R,u,ux,uy):
    error = 0
    number_of_elements = n4e.shape[0]
    xr = np.array([(c4n[n4e[i,0],0] - c4n[n4e[i,2],0])/2. for i in range(number_of_elements)])
    yr = np.array([(c4n[n4e[i,0],1] - c4n[n4e[i,2],1])/2. for i in range(number_of_elements)])
    xs = np.array([(c4n[n4e[i,1],0] - c4n[n4e[i,2],0])/2. for i in range(number_of_elements)])
    ys = np.array([(c4n[n4e[i,1],1] - c4n[n4e[i,2],1])/2. for i in range(number_of_elements)])
    J = xr*ys-xs*yr
    rx=ys/J
    ry=-xs/J
    sx=-yr/J
    sy=xr/J
    for i in range(number_of_elements):
        Du_x = np.matmul(rx[i]*Dr_R+sx[i]*Ds_R, u[ind4e[i]])
        Du_y = np.matmul(ry[i]*Dr_R+sy[i]*Ds_R, u[ind4e[i]])
        Dex = ux(c4n[ind4e[i]]) - Du_x
        Dey = uy(c4n[ind4e[i]]) - Du_y
        error += J[i] * (np.matmul(Dex,np.matmul(M_R,Dex)) + np.matmul(Dey,np.matmul(M_R,Dey)))
    return np.sqrt(error)
```

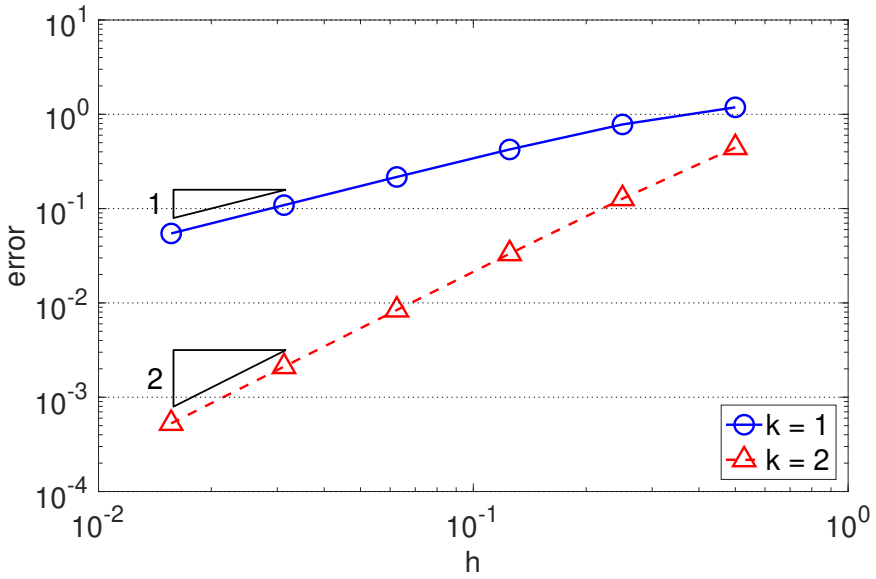


Figure 2.3: Convergence history for Example 2.1

The following numerical example is introduced to verify above matlab codes.

Example 2.1. *Consider the domain* $\Omega = [0,1]^2$. *The source term* f *is chosen such that*

$$u = \sin(\pi x)\sin(\pi y) \tag{2.31}$$

is the analytical solution to (2.1).

The results of this example are displayed in Figure 2.3. Here, the optimal rates of convergence are obtained. The matlab code for this example is as follows.

main_2d_triangle The following Matlab code solves the Poisson problem by using several matlab codes such as mesh_fem_2d, get_matrices_2d_triangle, fem_for_poisson_2d and compute_error_fem_2d.

```
iter = 6
xl, xr, yl, yr=0, 1, 0, 1
M = 2 ** np.arange(2,iter+2)
f = lambda x: 2 * np.pi**2 * np.sin(np.pi * x[:,0]) * np.sin(np.pi * x[:,1])
u_D = lambda x: 0 * x[:,0]
ux = lambda x: np.pi * np.cos(np.pi * x[:,0]) * np.sin(np.pi * x[:,1])
uy = lambda x: np.pi * np.sin(np.pi * x[:,0]) * np.cos(np.pi * x[:,1])

```

```
h = 1 / M
k = 2
error = np.zeros(iter)
M_R, Srr_R, Srs_R, Ssr_R, Sss_R, Dr_R, Ds_R = get_matrices_2d_triangle(k)
for i in range(iter):
    c4n, n4e, n4db, ind4e = mesh_fem_2d(xl, xr, yl, yr, M[i], M[i], k)
    u = fem_for_poisson_2d(c4n,n4e,n4db,ind4e,M_R,Srr_R,Srs_R,Ssr_R,Sss_R,f,u_D)
    error[i] = compute_error_fem_2d(c4n,n4e,ind4e,M_R,Dr_R,Ds_R,u,ux,uy)

rate = (np.log(error[1:])-np.log(error[:-1]))/(np.log(h[1:])-np.log(h[:-1]))
print(rate)
```

Exercises

1. Add the matrices for the cubic approximations ($k = 3$) in **get_matrices_2d_triangle** and check the convergence rate.

2. Modify **fem_for_poisson_2d_triangle_ex2** to solve the Poisson problem with non-homogeneous Dirichlet boundary condition,

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}) && \text{in } \Omega \\ u(\boldsymbol{x}) &= u_D(\boldsymbol{x}) && \text{on } \partial\Omega. \end{aligned}$$

3. Modify **fem_for_poisson_2d_ex3** to solve the Poisson problem with mixed boundary condition,

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}) && \text{in } \Omega \\ u(\boldsymbol{x}) &= u_D(\boldsymbol{x}) && \text{on } \Gamma_D \\ \nabla u(\boldsymbol{x}) \cdot \boldsymbol{n} &= u_N(\boldsymbol{x}) && \text{on } \Gamma_N, \end{aligned}$$

where Γ_D denotes the Dirichlet boundary, Γ_N denotes the Neumann boundary, and $\boldsymbol{n}$ is the outward unit normal vector.

4. Prove Lemma 2.1.