

Chapter 1

FEM for Poisson problem in 1D

Consider the one dimensional Poisson problem

$$-u''(x) = f(x) \quad \text{in } \Omega = (x_\ell, x_r) \quad (1.1a)$$

$$u(x_\ell) = u(x_r) = 0. \quad (1.1b)$$

In order to solve the problem (1.1), we multiply both sides of (1.1a) by the test function $v(x)$ and integrate over the domain,

$$-\int_{\Omega} u''(x)v(x) \, dx = \int_{\Omega} f(x)v(x) \, dx. \quad (1.2)$$

Then the weak formulation of the problem (1.1) is as follows: find $u(x) \in H_0^1(\Omega)$ such that

$$\int_{\Omega} u'(x)v'(x) \, dx = \int_{\Omega} f(x)v(x) \, dx, \quad (1.3)$$

for all $v(x) \in H_0^1(\Omega)$. The weak formulation (1.3) is obtained from (1.2) by using the boundary condition (1.1b) and integration by parts.

For the finite element approach, we need a conforming finite element space

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

where $P_k(I)$ is the polynomial function space of degrees $\leq k$. Then the variational formulation of the problem (1.1) is as follows: find $u_h \in V_h^k$ such that

$$\int_{\Omega} u_h' v_h' \, dx = \int_{\Omega} f v_h \, dx, \quad (1.4)$$

for all $v_h \in V_h^k$.

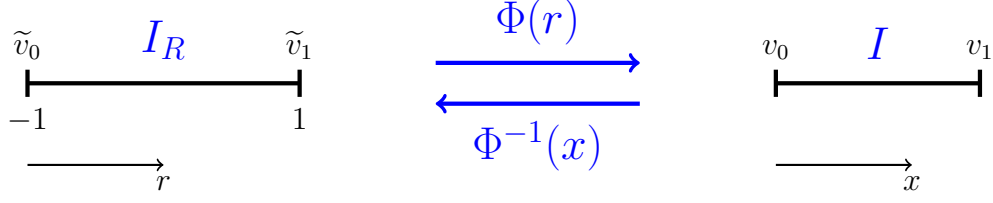


Figure 1.1: An affine mapping from the reference interval I_R to an interval I .

1.1 Affine mapping

Let us define the reference interval I_R as

$$I_R = \{r \mid -1 \leq r \leq 1\}.$$

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

$$\begin{cases} 0 \leq \tilde{\lambda}_i(r) \leq 1, & i = 0, 1, \\ \tilde{\lambda}_0(r) + \tilde{\lambda}_1(r) = 1. \end{cases} \quad (1.5)$$

Let us consider an interval I ,

$$I = \{x \mid v_0 \leq x \leq v_1\}.$$

Then we have an affine mapping Φ such that

$$\Phi(r) = v_0 \tilde{\lambda}_0(r) + v_1 \tilde{\lambda}_1(r) = x, \quad (1.6)$$

where $x \in I$. Then (1.5), $v_0 = -1$ and $v_1 = 1$ imply

$$\begin{aligned} r &= -\tilde{\lambda}_0(r) + \tilde{\lambda}_1(r) = -\tilde{\lambda}_0(r) + (1 - \tilde{\lambda}_0(r)) \\ &= 1 - 2\tilde{\lambda}_0(r). \end{aligned}$$

Thus, barycentric coordinates can be written as

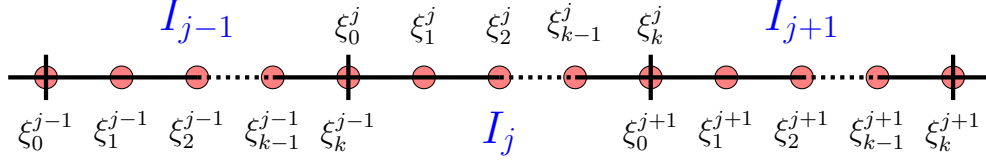
$$\tilde{\lambda}_0(r) = \frac{1-r}{2}, \quad \tilde{\lambda}_1(r) = \frac{1+r}{2}. \quad (1.7)$$

From (1.6) and (1.7),

$$\frac{dx}{dr} = \frac{v_1 - v_0}{2}.$$

A simple calculation yields that

$$I = \frac{dx}{dr} \frac{dr}{dx} = x_r r_x.$$

Figure 1.2: Nodes in \mathcal{T}_h with k .

Therefore, we have

$$r_x = \frac{1}{J},$$

where J is the Jacobian, $J = (v_1 - v_0)/2$.

Let $\lambda_0(x)$ and $\lambda_1(x)$ be the barycentric coordinates on I such that

$$\lambda_0(x) = \frac{v_1 - x}{v_1 - v_0}, \quad \lambda_1(x) = \frac{x - v_0}{v_1 - v_0}. \quad (1.8)$$

Since $\lambda_i(x)$ and $\tilde{\lambda}_i(r)$ ($i = 0, 1$) are linear functions and Φ is a linear mapping, we have

$$\lambda_i(x) = \tilde{\lambda}_i(\Phi^{-1}(x)), \quad (1.9a)$$

$$\tilde{\lambda}_i(r) = \lambda_i(\Phi(r)). \quad (1.9b)$$

Also we can obtain the following relations by simple calculation.

$$\frac{d}{dx} \lambda_i(x) = \frac{dr}{dx} \frac{d}{dr} \tilde{\lambda}_i(\Phi^{-1}(x)) = r_x \frac{d}{dr} \tilde{\lambda}_i(r). \quad (1.10)$$

1.2 Triangulation

Consider the domain $\Omega = [0, 1]$. The number of nodes in the triangulation \mathcal{T}_h is determined by the number of elements and the polynomial order (see, Figure 1.2). Let N be the number of nodes, M be the number of elements, I_j be the j -th element in \mathcal{T}_h , ξ_i be the i -th node in \mathcal{T}_h , and ξ_i^j be the i -th node in I_j . Then the following relations hold

$$N = kM + 1,$$

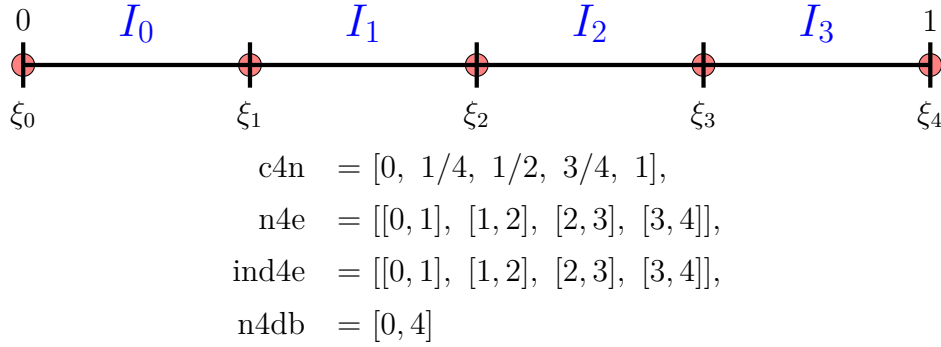
$$\xi_i^j = \xi_{jk+i},$$

$$\xi_k^j = \xi_0^{j+1}.$$

Here the indices i and j satisfy $0 \leq j \leq M - 1$ and $0 \leq i \leq k$, respectively.

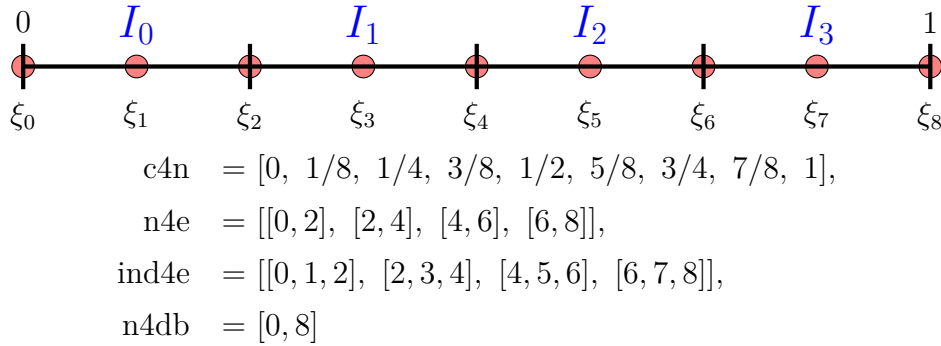
The data for a given triangulation \mathcal{T}_h are described in N dimensional vector $\mathbf{c4n}$, $M \times 2$ matrix $\mathbf{n4e}$, 2 dimensional vector $\mathbf{n4db}$ and $M \times (k+1)$ matrix $\mathbf{ind4e}$. The vector $\mathbf{c4n}$ means coordinates for nodes and its i -th row represents the coordinate for i -th node ξ_i . The matrix $\mathbf{n4e}$ means nodes for elements and j -th row of $\mathbf{n4e}$ shows the left-end and right-end nodes in I_j . The matrix $\mathbf{ind4e}$ means index for elements and its j -th row represents all indices of nodes in I_j . The vector $\mathbf{n4db}$ means nodes for Dirichlet boundary and $\mathbf{n4db} = [\xi_1, \xi_N]^t$.

For example if $\Omega = (0, 1)$, $M = 4$ and $k = 1$, data are stored as follows.



In this case, each element has two nodes which are left-end and right-end nodes. Note that $\mathbf{n4e}$ and $\mathbf{ind4e}$ are exactly the same for $k = 1$.

If $\Omega = (0, 1)$, $M = 4$ and $k = 2$, data are stored as follows.



In this case, each element has an extra point except left-end and right-end nodes. Thus $\mathbf{ind4e}$ is different from $\mathbf{n4e}$. Here the first column of $\mathbf{ind4e}$ is the same as the first column of $\mathbf{n4e}$ and the third column of $\mathbf{ind4e}$ is the same as the second column of $\mathbf{n4e}$. In general, for a fixed k , the first column of $\mathbf{ind4e}$ is the same as the first column of $\mathbf{n4e}$ and the last column of $\mathbf{ind4e}$ is the same as the second column of $\mathbf{n4e}$.

mesh_fem_1d The following python code generates an uniform mesh on the domain $\Omega = (a, b)$ in \mathbb{R} with mesh size $h = 1/M$. Also this code returns an index matrix for continuous k -th order polynomial approximations.

```

1 def mesh_fem_1d(a,b,M,k):
2     nrNodes = k*M + 1
3     c4n = np.linspace(a, b, nrNodes)
4     n4e = np.array([[i*k, (i+1)*k] for i in range(M)])
5     n4db = np.array([0, nrNodes-1])
6     ind4e = np.array([list(range(i*k, (i+1)*k+1)) for i in range(M)])
7     return (c4n,n4e,n4db,ind4e)

```

1.3 Basis functions of V_h^k and Numerical solution

Basis functions of V_h^k are piecewise polynomial and they have Kronecker delta property,

$$\phi_i(\xi_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases} \quad (1.11)$$

$$\sum_{i=1}^N \phi_i(x) = 1 \quad \forall x \in \Omega, \quad (1.12)$$

where ϕ_i is the i -th basis function of V_h^k and ξ_j is the j -th node point in \mathcal{T}_h . These basis functions can be locally expressed similar to the node points. The local basis function $\phi_i^j(x)$ is the i -th basis function in the j -th element. Then the following relations hold

$$\begin{aligned} \phi_i^j(x) &= \phi_{jk+i}(x) \Big|_{I_j} \\ \phi_i^j(x) &= 0 \quad \text{if } x \in \Omega \setminus I_j \\ \sum_{i=0}^k \phi_i^j(x) &= 1 \quad \forall x \in I_j. \end{aligned}$$

Figure 1.3 and 1.4 show global and local basis functions, respectively.

Using these basis functions, the interpolate function of $f(x)$ can be written as follows

$$\mathcal{I}f(x) = \sum_{i=0}^{N-1} f_i \phi_i(x) \quad (1.13)$$

where $f_i = f(\xi_i)$. Clearly, $f(\xi_i) = \mathcal{I}f(\xi_i)$ for all $0 \leq i \leq N-1$ by (1.11). If $f(x) \in V_h^k$, the interpolate function is the same as $f(x)$, i.e., $f(x) = \mathcal{I}f(x)$. Thus, the numerical solution can be written as follows

$$u_h(x) = \sum_{i=0}^{N-1} u_i \phi_i(x) \quad (1.14)$$

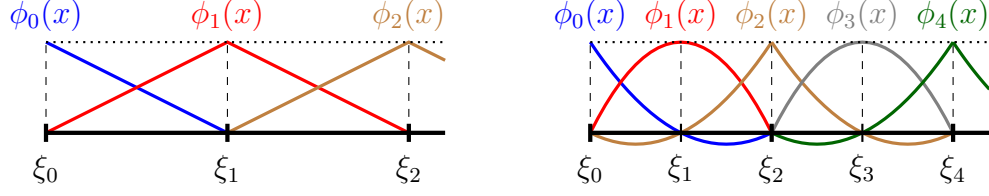


Figure 1.3: Global basis functions of V_h^1 (left) and V_h^2 (right) on an interval.

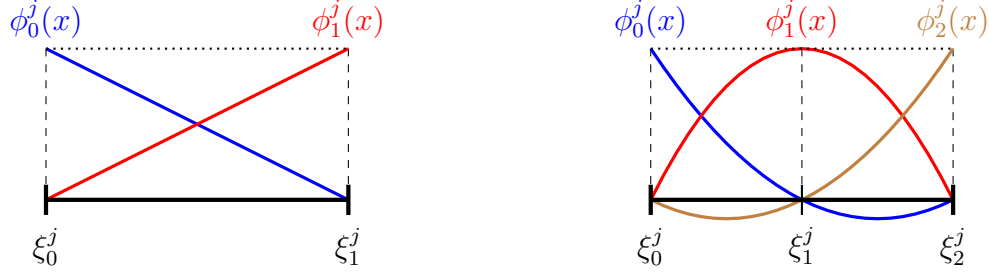


Figure 1.4: Local basis functions of V_h^1 (left) and V_h^2 (right) on the j -th element.

where $u_i = u_h(\xi_i)$. Especially, the numerical solution can be written locally because the solution is piecewise polynomial function:

$$u_h(x) \Big|_{I_j} = \sum_{i=0}^k u_i^j \phi_i^j(x) \quad (1.15)$$

where $u_i^j = u_h(\xi_i^j)$. Then the derivative of the solution is easily obtained from (1.14) and (1.15).

$$\frac{d}{dx} u_h(x) = \sum_{i=0}^{N-1} u_i \frac{d}{dx} \phi_i(x), \quad (1.16)$$

$$\frac{d}{dx} u_h(x) \Big|_{I_j} = \sum_{i=0}^k u_i^j \frac{d}{dx} \phi_i^j(x). \quad (1.17)$$

1.4 Mass matrix and Stiffness matrix

For a test function $\phi_i(x) \in V_h^k$, the variational formulation (1.4) can be rewritten as

$$\sum_{j=0}^{N-1} u_j \int_{\Omega} \phi_i'(x) \phi_j'(x) dx = \int_{\Omega} f(x) \phi_i(x) dx \quad (1.18)$$

by (1.14) and (1.15). Then, for all basis functions in V_h^k , we have the following finite element system

$$A\mathbf{u} = \mathbf{b} \quad (1.19)$$

where

$$(A)_{ij} = \int_{\Omega} \phi'_{i-1}(x) \phi'_{j-1}(x) \, dx \quad (1.20a)$$

$$(\mathbf{b})_i = \int_{\Omega} f(x) \phi_{i-1}(x) \, dx \quad (1.20b)$$

$$(\mathbf{u})_j = u_{j-1}. \quad (1.20c)$$

Here, the matrix A is called the global stiffness matrix and the right-hand side \mathbf{b} is called the load vector. In order to compute the load vector, f is replaced by the interpolate function $\mathcal{I}f(x)$ in general. Thus, FE system (1.19) can be rewritten as

$$A\mathbf{u} = M\mathbf{f} \quad (1.21)$$

where

$$(M)_{ij} = \int_{\Omega} \phi_{i-1}(x) \phi_{j-1}(x) \, dx \quad (1.22a)$$

$$(\mathbf{f})_j = f_{j-1}. \quad (1.22b)$$

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

As in the previous section, a basis function $\phi_i(x)$ is zero except at most two elements and $\phi'_i(x)$ is also zero in elements where $\phi_i(x) = 0$. Thus, the global stiffness matrix and the global mass matrix can be assembled by using local basis functions,

$$A = \sum_{\ell=1}^M A_{I_{\ell}}, \quad (1.23)$$

$$M = \sum_{\ell=1}^M M_{I_{\ell}}. \quad (1.24)$$

Here, $(k+1)$ -by- $(k+1)$ matrices $A_{I_{\ell}}$ and $M_{I_{\ell}}$ are defined as

$$(A_{I_{\ell}})_{ij} = \int_{I_{\ell}} \frac{d}{dx} \phi_{i-1}^{\ell}(x) \frac{d}{dx} \phi_{j-1}^{\ell}(x) \, dx \quad (1.25)$$

$$(M_{I_{\ell}})_{ij} = \int_{I_{\ell}} \phi_{i-1}^{\ell}(x) \phi_{j-1}^{\ell}(x) \, dx \quad (1.26)$$

for $1 \leq i, j \leq k+1$. The matrices A_{I_ℓ} and M_{I_ℓ} are called the local stiffness matrix and the local mass matrix, respectively.

The following lemma is helpful to compute the local matrices.

Lemma 1.1. *For a given interval $I = [v_0, v_1]$ with barycentric coordinates λ_0 and λ_1 , it holds for $a, b \in \mathbb{N}_0$ that*

$$\int_I \lambda_0^a \lambda_1^b dx = |I| \frac{a! b!}{(a+b+1)!} \quad (1.27)$$

where $|I| = v_1 - v_0$.

Proof. From the definition of barycentric coordinates (1.8),

$$\begin{aligned} \int_I \lambda_0^a \lambda_1^b dx &= \int_I \left(\frac{v_1 - x}{|I|} \right)^a \left(\frac{x - v_0}{|I|} \right)^b dx \\ &= \frac{1}{|I|^{a+b}} \int_I (v_1 - x)^a (x - v_0)^b dx. \end{aligned}$$

Then repeated integration by parts yield that

$$\begin{aligned} \int_I \lambda_0^a \lambda_1^b dx &= \frac{1}{|I|^{a+b}} \frac{b}{a+1} \int_{v_0}^{v_1} (v_1 - x)^{a+1} (x - v_0)^{b-1} dx \\ &= \frac{1}{|I|^{a+b}} \frac{b(b-1)}{(a+1)(a+2)} \int_{v_0}^{v_1} (v_1 - x)^{a+2} (x - v_0)^{b-2} dx \\ &\quad \vdots \\ &= \frac{1}{|I|^{a+b}} \frac{b!}{(a+1)(a+2) \cdots (a+b)} \int_{v_0}^{v_1} (v_1 - x)^{a+b} dx \\ &= |I| \frac{a! b!}{(a+b+1)!}. \end{aligned}$$

□

Since the measure of the reference interval I_R is 2, i.e. $|I_R| = 2$, we have

$$\int_{I_R} \tilde{\lambda}_0^a \tilde{\lambda}_1^b dr = 2 \frac{a! b!}{(a+b+1)!}$$

and

$$\int_I \lambda_0^a \lambda_1^b dx = \frac{v_1 - v_0}{2} 2 \frac{a! b!}{(a+b+1)!} = J \int_{I_R} \tilde{\lambda}_1^a \tilde{\lambda}_2^b dr.$$

Therefore, we can compute the integration (1.27) on an arbitrary interval I by using the Jacobian and the integration over the reference interval I_R .

In order to compute derivatives, we now introduce a differentiation matrix D such that

$$(D)_{ij} = \phi'_{j-1}(\xi_{i-1}). \quad (1.28)$$

Clearly, $\phi'_i(x) \in V_h^{k-1} \subset V_h^k$ because $\phi_i(x) \in V_h^k$. Thus, $\phi'_i(x)$ satisfies

$$\phi'_{i-1}(x) = \sum_{j=0}^{N-1} \phi'_{i-1}(\xi_j) \phi_j(x) = (D^t)_i \boldsymbol{\phi}$$

where $(D^t)_i$ is the i -th row of the matrix D^t , i.e. $(D^t)_i = [\phi'_{i-1}(\xi_1) \cdots \phi'_{i-1}(\xi_N)]$, and $\boldsymbol{\phi} = [\phi_0(x) \cdots \phi_{N-1}(x)]^t$. Similar to the stiffness and mass matrices, D can be assembled by the local differentiation matrix

$$D = \sum_{\ell=0}^{M-1} D_{I_\ell}, \quad (D_{I_\ell})_{ij} = \frac{d\phi_{j-1}^\ell}{dx}(\xi_{i-1}^\ell).$$

Thus, the derivative for the local basis function $\phi_i^\ell(x)$ is written as

$$\frac{d}{dx} \phi_{i-1}^\ell(x) = \sum_{j=0}^k \frac{d\phi_{i-1}^\ell}{dx}(\xi_j^\ell) \phi_j^\ell(x) = (D_{I_\ell}^t)_i \boldsymbol{\phi}^\ell \quad (1.29)$$

where $(D_{I_\ell}^t)_i$ is the i -th row of the matrix $D_{I_\ell}^t$ and $\boldsymbol{\phi}^\ell = [\phi_0^\ell(x) \cdots \phi_k^\ell(x)]^t$. Then, we have

$$\begin{aligned} u'_h(\xi_{m-1}) &= \sum_{i=0}^{N-1} u_i \phi'_i(\xi_{m-1}) = \sum_{i=0}^{N-1} u_i \sum_{j=0}^{N-1} \phi'_i(\xi_j) \phi_j(\xi_{m-1}) = \sum_{i=0}^{N-1} u_i \phi'_i(\xi_{m-1}) = (D)_m \mathbf{u} \\ u'_h(\xi_{n-1}^\ell) &= \sum_{i=0}^k u_i^\ell \frac{d\phi_i^\ell}{dx}(\xi_{n-1}^\ell) = \sum_{i=0}^k u_i^\ell \sum_{j=0}^k \frac{d\phi_i^\ell}{dx}(\xi_j^\ell) \phi_j^\ell(\xi_{n-1}^\ell) = \sum_{i=0}^k u_i^\ell \frac{d\phi_i^\ell}{dx}(\xi_{n-1}^\ell) = (D_{I_\ell})_m \mathbf{u}^\ell, \end{aligned}$$

where $1 \leq m \leq N$, $1 \leq n \leq k+1$, $\mathbf{u} = [u_0, \dots, u_{N-1}]$, and $\mathbf{u}^\ell = [u_0^\ell, \dots, u_k^\ell]$.

Lemma 1.2. *For a given interval $I = [v_0, v_1]$ with barycentric coordinates λ_0 and λ_1 , it holds for $a, b \in \mathbb{N}_0$ that*

$$\frac{d}{dx}(\lambda_0^a(x) \lambda_1^b(x)) = r_x \frac{d}{dr}(\tilde{\lambda}_0^a(r) \tilde{\lambda}_1^b(r)) \quad (1.30)$$

where $r = \Phi^{-1}(x)$, $r_x = 1/J$ and $J = (v_1 - v_0)/2$.

Proof. By the product rule and (1.10),

$$\begin{aligned}
\frac{d}{dx} (\lambda_0^a(x) \lambda_1^b(x)) &= \lambda_0^{a-1}(x) \lambda_1^b(x) \frac{d}{dx} \lambda_0(x) + \lambda_0^a(x) \lambda_1^{b-1}(x) \frac{d}{dx} \lambda_1(x) \\
&= \tilde{\lambda}_0^{a-1}(r) \tilde{\lambda}_1^b(r) r_x \frac{d}{dr} \tilde{\lambda}_0(r) + \tilde{\lambda}_0^a(r) \tilde{\lambda}_1^{b-1}(r) r_x \frac{d}{dr} \tilde{\lambda}_1(r) \\
&= r_x \left(\tilde{\lambda}_0^{a-1}(r) \tilde{\lambda}_1^b(r) \frac{d}{dr} \tilde{\lambda}_0(r) + \tilde{\lambda}_0^a(r) \tilde{\lambda}_1^{b-1}(r) \frac{d}{dr} \tilde{\lambda}_1(r) \right) \\
&= r_x \frac{d}{dr} \left(\tilde{\lambda}_0^a(r) \tilde{\lambda}_1^b(r) \right).
\end{aligned}$$

□

Now we are ready to compute the local stiffness and mass matrices. For convenience, we will use the notations M and S instead of M_{I_ℓ} and A_{I_ℓ} for a fixed element I , respectively. Also, ϕ and $\tilde{\phi}$ are basis functions on the interval I and the reference interval I_R , respectively. By the affine mapping, we have

$$(M)_{ij} = \int_I \phi_{i-1}(x) \phi_{j-1}(x) dx = J \int_{I_R} \tilde{\phi}_{i-1}(r) \tilde{\phi}_{j-1}(r) dr = J(M_R)_{ij} \quad (1.31)$$

where M_R is the mass matrix on I_R . By (1.29) and (1.30),

$$\begin{aligned}
(S)_{ij} &= \int_I \phi'_{i-1}(x) \phi'_{j-1}(x) dx \\
&= \int_I \left(\sum_{\ell=0}^k \frac{d\phi_{i-1}}{dx}(\xi_\ell) \phi_\ell(x) \right) \left(\sum_{m=0}^k \frac{d\phi_{j-1}}{dx}(\xi_m) \phi_m(x) \right) dx \\
&= J \int_{I_R} \left(\sum_{\ell=0}^k r_x \frac{d\tilde{\phi}_{i-1}}{dr}(\Phi^{-1}(\xi_\ell)) \tilde{\phi}_\ell(r) \right) \left(\sum_{m=0}^k r_x \frac{d\tilde{\phi}_{j-1}}{dr}(\Phi^{-1}(\xi_m)) \tilde{\phi}_m(r) \right) dr \\
&= r_x^2 J (S_R)_{ij} = \frac{1}{J} (S_R)_{ij}
\end{aligned} \quad (1.32)$$

and

$$\begin{aligned}
(S_R)_{ij} &= \int_{I_R} \left((D_R^t)_i \tilde{\phi} \right) \left((D_R^t)_j \tilde{\phi} \right) dr \\
&= (D_R^t)_i M_R (D_R)_j \\
&= (D_R^t M_R D_R)_{ij}
\end{aligned} \quad (1.33)$$

where S_R is the local stiffness matrix on I_R , D_R is the differentiation matrix on I_R and $\tilde{\phi} = [\tilde{\phi}_0(r) \cdots \tilde{\phi}_k(r)]^t$. Thus, the local stiffness and mass matrices are obtained from the reference mass and stiffness matrices by using affine mapping.

$$M = J M_R, \quad S = \frac{1}{J} S_R \quad (1.34)$$

Here, 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 $P_1(I_R)$ are the same as the barycentric coordinates. Thus the basis functions and their derivatives are

$$\begin{aligned}\tilde{\phi}_0(r) &= \tilde{\lambda}_0(r), & \tilde{\phi}'_0(r) &= -\frac{1}{2}, \\ \tilde{\phi}_1(r) &= \tilde{\lambda}_1(r), & \tilde{\phi}'_1(r) &= \frac{1}{2}.\end{aligned}$$

Then, (1.26) and (1.27) yield

$$\begin{aligned}M_R &= \frac{1}{3} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \\ \therefore \begin{cases} (M_R)_{11} = \int_{I_R} \tilde{\lambda}_0^2(r) dr = \frac{2!}{3!}|I_R| = \frac{2}{3} \\ (M_R)_{12} = \int_{I_R} \tilde{\lambda}_0(r)\tilde{\lambda}_1(r) dr = \frac{1}{3!}|I_R| = \frac{1}{3} \\ (M_R)_{22} = \int_{I_R} \tilde{\lambda}_1^2(r) dr = \frac{2}{3} \end{cases}\end{aligned} \quad (1.35)$$

The differentiation matrix D_R is obtained from (1.28)

$$D_R = \frac{1}{2} \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix}. \quad (1.36)$$

Therefore, the local stiffness matrix is obtained from (1.33)

$$S_R = D_R^t M_R D_R = \frac{1}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}. \quad (1.37)$$

P_2 matrices The basis functions in $P_2(I_R)$ are obtained from the barycentric coordinates such that

$$\begin{aligned}\tilde{\phi}_0(r) &= \tilde{\lambda}_0(r)(2\tilde{\lambda}_0(r) - 1) \\ \tilde{\phi}_1(r) &= 4\tilde{\lambda}_0(r)\tilde{\lambda}_1(r) \\ \tilde{\phi}_2(r) &= \tilde{\lambda}_1(r)(2\tilde{\lambda}_1(r) - 1).\end{aligned}$$

Then, their derivatives are obtained by using the chain rule

$$\begin{aligned}\tilde{\phi}'_0(r) &= -2\tilde{\lambda}_0(r) + \frac{1}{2} \\ \tilde{\phi}'_1(r) &= 2\tilde{\lambda}_0(r) - 2\tilde{\lambda}_1(r) \\ \tilde{\phi}'_2(r) &= 2\tilde{\lambda}_1(r) - \frac{1}{2}.\end{aligned}$$

Similar to the P_1 matrices, we have the mass matrix

$$M_R = \frac{1}{15} \begin{pmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{pmatrix}, \quad (1.38)$$

$$\therefore \begin{cases} (M_R)_{11} = \int_{I_R} 4\tilde{\lambda}_0^4 - 4\tilde{\lambda}_0^3 + \tilde{\lambda}_0^2 dr = 2 \left(4\frac{4!}{5!} - 4\frac{3!}{4!} + \frac{2!}{3!} \right) = \frac{4}{15} \\ (M_R)_{12} = \int_{I_R} 8\tilde{\lambda}_0^3\tilde{\lambda}_1 - 4\tilde{\lambda}_0^2\tilde{\lambda}_1 dr = 2 \left(8\frac{3!}{5!} - 4\frac{2!}{4!} \right) = \frac{2}{15} \\ (M_R)_{13} = \int_{I_R} 4\tilde{\lambda}_0^2\tilde{\lambda}_1^2 - 2\tilde{\lambda}_0^2\tilde{\lambda}_1 - 2\tilde{\lambda}_0\tilde{\lambda}_1^2 + \tilde{\lambda}_0\tilde{\lambda}_1 dr = 2 \left(4\frac{2!2!}{5!} - 2\frac{2!}{4!} - 2\frac{2!}{4!} + \frac{1}{3!} \right) = \frac{-1}{15} \\ (M_R)_{22} = \int_{I_R} 16\tilde{\lambda}_0^2\tilde{\lambda}_1^2 dr = 32\frac{2!2!}{5!} = \frac{16}{15} \\ (M_R)_{23} = \int_{I_R} 8\tilde{\lambda}_0\tilde{\lambda}_1^3 - 4\tilde{\lambda}_0\tilde{\lambda}_1^2 dr = 2 \left(8\frac{3!}{5!} - 4\frac{2!}{4!} \right) = \frac{2}{15} \\ (M_R)_{33} = \int_{I_R} 4\tilde{\lambda}_1^4 - 4\tilde{\lambda}_1^3 + \tilde{\lambda}_1^2 dr = 2 \left(4\frac{4!}{5!} - 4\frac{3!}{4!} + \frac{2!}{3!} \right) = \frac{4}{15} \end{cases}$$

the differentiation matrix

$$D_R = \frac{1}{2} \begin{pmatrix} -3 & 4 & -1 \\ -1 & 0 & 1 \\ 1 & -4 & 3 \end{pmatrix}, \quad (1.39)$$

and the stiffness matrix

$$S_R = D_R^t M_R D_R = \frac{1}{6} \begin{pmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7 \end{pmatrix}. \quad (1.40)$$

get_matrices_1d The following python code generates the mass matrix M_R , the stiffness matrix S_R and the differentiation matrix D_R for continuous k -th order polynomial approximations on the reference interval I_R .

```

1 def get_matrices_1d(k=1):
2     if k == 1:
3         M_R = np.array([[2, 1],[1, 2]], dtype=np.float64) / 3.
4         S_R = np.array([[1, -1],[-1, 1]], dtype=np.float64) / 2.
5         D_R = np.array([[-1, 1],[-1, 1]], dtype=np.float64) / 2.
6     elif k == 2:
7         M_R = np.array([[4, 2, -1],[2, 16, 2],[-1, 2, 4]], dtype=np.float64) / 15.
8         S_R = np.array([[7, -8, 1],[-8, 16, -8],[1, -8, 7]], dtype=np.float64) / 6.
9         D_R = np.array([[-3, 4, -1],[-1, 0, 1],[1, -4, 3]], dtype=np.float64) / 2.
10    elif: ...
11    return (M_R, S_R, D_R)

```

1.5 Python codes in 1D

Now we are ready to assemble the global stiffness matrix A and the global load vector \mathbf{b} in (1.19). In the matlab code, A and \mathbf{b} are assembled by using the local stiffness matrix (1.32) and the local mass matrix (1.31).

fem_for_poisson_1d The following python code solves the Poisson problem. In order to use this code, mesh information ($c4n$, $n4e$, $n4db$, $ind4e$), matrices (M_R , S_R), the source f , and the boundary condition u_D are necessary. Then, the results of this code are the numerical solution u , the global stiffness matrix A , the global load vector \mathbf{b} and the freenodes.

```

1 def fem_for_poisson_1d(c4n,n4e,n4db,ind4e,k,M_R,S_R,f,u_D):
2     number_of_nodes = len(c4n)
3     number_of_elements = len(n4e)
4     b = np.zeros(number_of_nodes)
5     u = np.zeros(number_of_nodes)
6     J = np.array([(c4n[n4e[i],1]]-c4n[n4e[i],0])/2 for i in range(number_of_elements)])
7     Aloc = np.array([S_R.flatten()/J[i] for i in range(number_of_elements)])
8     for i in range(number_of_elements):
9         b[ind4e[i]] += J[i]*np.matmul(M_R, f(c4n[ind4e[i]]))
10    row_ind = np.tile(ind4e.flatten(),(k+1,1)).T.flatten()
11    col_ind = np.tile(ind4e,(1,k+1)).flatten()
12    A_C00 = coo_matrix((Aloc.flatten(), (row_ind, col_ind)),
13                      shape=(number_of_nodes, number_of_nodes))
14    A = A_C00.tocsr()
15    dof = np.setdiff1d(range(0,number_of_nodes), n4db)
16    u[dof] = spsolve(A[dof, :].tocsc()[:, dof].tocsr(), b[dof])
17    return u

```

compute_error_fem_1d The following python code computes the semi H1 error between the exact solution and the numerical solution.

```

1 def compute_error_fem_1d(c4n,ind4e,M_R,D_R,u,Du):
2     error = 0
3     number_of_elements = len(ind4e)
4     J = np.array([(c4n[n4e[i],1]]-c4n[n4e[i],0])/2 for i in range(number_of_elements)])
5     for i in range(number_of_elements):
6         u_x = np.matmul(D_R, u[ind4e[i]]) / J[i]
7         D_e = Du(c4n[ind4e[i]]) - u_x
8         error += J[i] * np.matmul(D_e,np.matmul(M_R,D_e))
9     return np.sqrt(error)

```

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

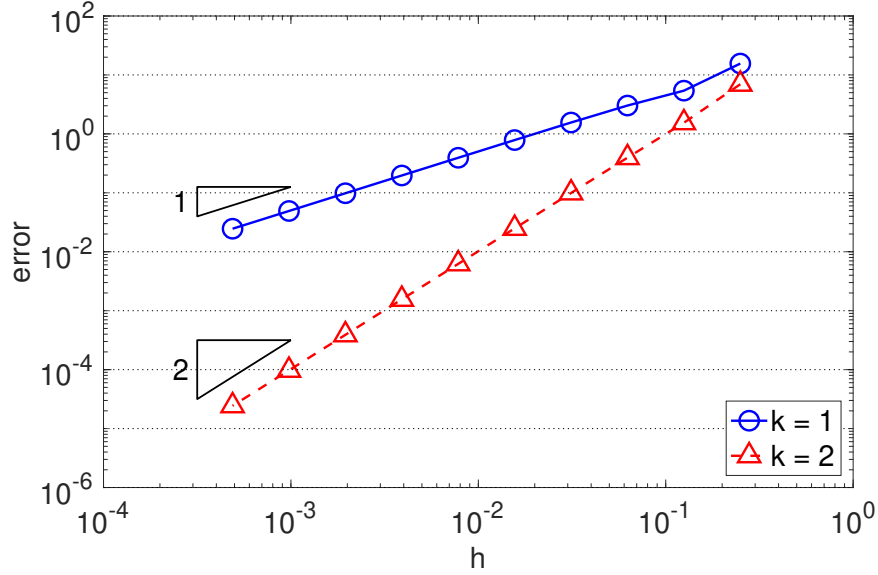


Figure 1.5: Convergence history for Example 1.1

Example 1.1. Consider the domain $\Omega = [0, 1]$. The source term f is chosen such that

$$u = \sin(5\pi x) \quad (1.41)$$

is the analytical solution to (1.1).

The results of this example are displayed in Figure 1.5. Here the optimal rates of convergence are obtained. The matlab code for this example is as follows.

main_1d The following python code solves the Poisson problem by using several matlab codes such as mesh_fem_1d.m, get_matrices_1d.m, fem_for_poisson_1d.m and compute_error_fem_1d.m.

```

1  iter = 10
2  a = 0
3  b = 1
4  k = 1
5  M = 2 ** np.arange(2, iter+2)
6  f = lambda x: 25 * np.pi**2 * np.sin(5 * np.pi * x)
7  u_D = lambda x: 0 * x
8  Du = lambda x: 5 * np.pi * np.cos(5 * np.pi * x)
9

```

```

10 error = np.zeros(iter)
11 h = 1 / M
12
13 M_R, S_R, D_R = get_matrices_1d(k)
14 for i in range(iter):
15     c4n, n4e, n4db, ind4e = mesh_fem_1d(a,b,M[i],k)
16     u = fem_for_poisson_1d(c4n,n4e,n4db,ind4e,k,M_R,S_R,f,u_D)
17     error[i] = compute_error_fem_1d(c4n,ind4e,M_R,D_R,u,Du)
18
19 rate = (np.log(error[1:])-np.log(error[:-1]))/(np.log(h[1:])-np.log(h[:-1]))
20 print(rate)

```

Exercises

1. Add the matrices for the cubic approximations ($k = 3$) in `get_matrices_1d` and check the convergence rate.
2. Modify `fem_for_poisson_1d_ex2` to solve the Poisson problem with non-homogeneous Dirichlet boundary condition,

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

3. Modify `fem_for_poisson_1d_ex3` to solve the Poisson problem with mixed boundary condition,

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

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