**AME50541: Finite Element Methods**
**Final Project: Due Wednesday, May 8, 2019**

In this project, you will enhance the finite element code you developed throughout the semester in the homework assignments into a general FEM code capable of handling unstructured meshes, non-homogeneous natural boundary conditions, and nonlinear problems. You will then use your code to solve a number of partial differential equations.

**Instructions**

- *All starter code can be found on the course website in the final project code distribution. Be sure to carefully read* `notation.m` *for a description of all relevant variables.* Be sure to run `init.m` each time you start MATLAB to add all required directories to your MATLAB path.
- You may assume the number of spatial dimensions ($d$ in the document, `ndim` in the code) in all tasks *without penalty*. That is, you only need your finite element code to work in one and two dimensions to receive full credit. I will award 5% extra credit on each task if your code works correctly for `ndim` is 1, 2, or 3, i.e., your finite element code works in one, two, and three dimensions.
- *Be sure to email all code to the instructor and TA.*

**Part 1:** (50 points) Since we want our code to be able to handle arbitrarily complex domains, it is necessary to have *simplex* elements available in our FEM code. As we discussed in class, mesh generation with simplex elements (triangles in 2d, tetrahedra in 3d) is a "solved" problem while mesh generation with hypercube elements is much more difficult.

**Part 1.1** First, we need to define the geometry of our element, which is given by the element interior (volume) and its boundary (faces). In addition, nodes are distributed throughout the element based on its degree of polynomial completeness. Simplex elements of order $p$, i.e., polynomial completeness of degree $p$, in $d$-dimension use interpolation functions that include all multinomial terms of order $p$: $\{\xi_1^{\alpha_1}\cdots\xi_d^{\alpha_d} \mid \sum_{i=1}^d \alpha_i \leqslant p\}$. Therefore, the number of nodes in a simplex element must be the number of unique multinomials

$$N_v^e = \sum_{n=0}^{p} \binom{n+d-1}{d-1}, \tag{1}$$

where we use $N_v^e$ throughout the document to denote the number of nodes in an element. For the special case of $d = 2$ (triangle), we have $N_v^e = (p+1)(p+2)/2$ nodes, which can easily be verified by referring to the Pascal triangle.

Since we will use the isoparametric concept to define basis functions, we only need to consider the unit right simplex. We assume the nodes $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$ are distributed uniformly throughout the element and ordered (ascending) first in the $\xi_1$-direction, then in the $\xi_2$-direction, etc., where $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_d)$ are the coordinates in the reference domain (Figure 1). Furthermore, let us number the $N_f = d+1$ faces of the simplex element as: face $f$ is the face with unit normal $\boldsymbol{N}_f = -\boldsymbol{e}_f$ for $f = 1, \ldots, d$, where $\boldsymbol{e}_f \in \mathbb{R}^d$ is the canonical unit vector, and face $d+1$ is the remaining face (unit normal $\boldsymbol{N}_{d+1} = \frac{1}{\sqrt{d}}\mathbf{1}$). Finally, introduce a matrix $\boldsymbol{\Theta} \in \mathbb{N}^{N_v^f \times N_f}$ ($\mathbb{N}^{m\times n}$ is the set of $m \times n$ matrices whose entries are natural numbers) where $N_v^f$ is the number of nodes on each element face, whose columns define the element nodes that lie on a particular face, i.e., $\Theta_{if}$ is the element node number corresponding to the $i$th node on face $f$ of the element.

For the special case of $p = 2$ simplex elements in $d = 2$ dimensions (Figure 1), the nodes are

$$\hat{\boldsymbol{\xi}}_1 = \begin{bmatrix}0\\0\end{bmatrix}, \quad \hat{\boldsymbol{\xi}}_2 = \begin{bmatrix}0.5\\0\end{bmatrix}, \quad \hat{\boldsymbol{\xi}}_3 = \begin{bmatrix}1\\0\end{bmatrix}, \quad \hat{\boldsymbol{\xi}}_4 = \begin{bmatrix}0\\0.5\end{bmatrix}, \quad \hat{\boldsymbol{\xi}}_5 = \begin{bmatrix}0.5\\0.5\end{bmatrix}, \quad \hat{\boldsymbol{\xi}}_6 = \begin{bmatrix}0\\1\end{bmatrix},$$

and the face-to-element vertx mapping and normal vectors are

$$\boldsymbol{\Theta} = \begin{bmatrix}1 & 1 & 3\\ 4 & 2 & 5\\ 6 & 3 & 6\end{bmatrix}, \qquad \boldsymbol{N}_1 = \begin{bmatrix}-1\\0\end{bmatrix}, \qquad \boldsymbol{N}_2 = \begin{bmatrix}0\\-1\end{bmatrix}, \qquad \boldsymbol{N}_3 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}.$$

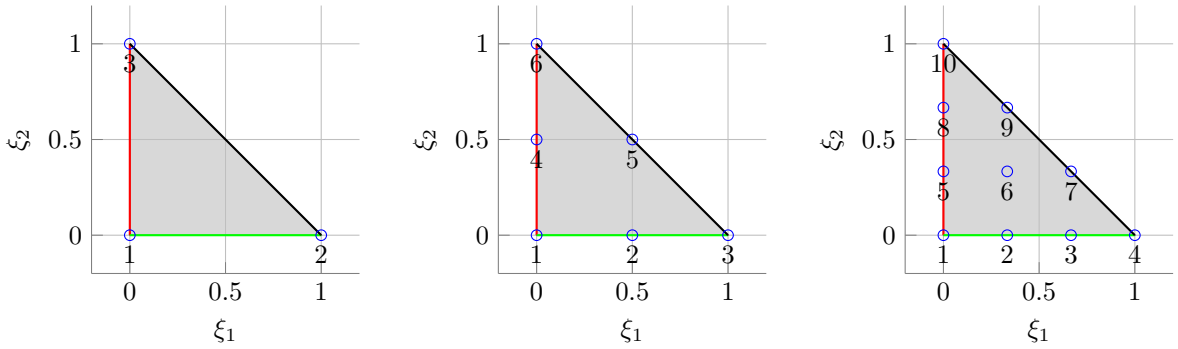


Figure 1: Triangular finite element in reference domain ($\xi_1$-$\xi_2$ space) with 3 nodes ($p = 1$) (*left*), 6 nodes ($p = 2$) (*center*), and 10 nodes ($p = 3$) (*right*). The faces are numbered as: face 1 (——), face 2 (——), and face 3 (——).

**Tasks for Part 1.1**

Write a function that defines the geometry of a simplex element of order $p$ in the reference domain in $d$ spatial dimensions. Your function should define the nodal positions in the reference domain $\{\boldsymbol{\xi}_k\}_{k=1}^{N_v^e}$, the face-to-element vertex mapping $\boldsymbol{\Theta}$, and the unit normal for each face $\{\boldsymbol{N}_i\}_{i=1}^{N_f}$. Your function should have the following signature:

```
function [zk, f2v, N] = create_nodes_bndy_refdom_simp(ndim, porder)
%CREATE_NODES_BNDY_REFDOM_SIMP Create nodal distribution and boundary of
%NDIM-dimensional simplex of order PORDER.
%
%Input arguments
%---------------
%   NDIM, PORDER : See notation.m
%
%Output arguments
%----------------
%   ZK : Array (NDIM, NV) : The nodal positions of the reference simplex
%     element in NDIM-dimensions and polynomial order PORDER.
%
%   F2V, N : See notation.m
```

Read all comments and `notation.m` carefully for instructions regarding the inputs and outputs to the function. For the $d = 2$ case (triangular element), plot all nodes of the element with blue circles. On top of these circles, plot the nodes on face 1 (left edge) with red `x`'s, the nodes of face 2 (bottom edge) with green `□`'s, and the nodes on face 3 (hypotenuse) with black `+`'s. For you reference, for the special case of $p = 2$, you code should return

$$\texttt{zk} = \begin{bmatrix} 0 & 0.5 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 1 \end{bmatrix}, \qquad \texttt{f2v} = \begin{bmatrix} 1 & 1 & 3 \\ 4 & 2 & 5 \\ 6 & 3 & 6 \end{bmatrix}, \qquad \texttt{N} = \begin{bmatrix} -1 & 0 & \frac{1}{\sqrt{2}} \\ 0 & -1 & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

**Part 1.2** Next, we need to define basis functions over the reference domain $\{\psi_i(\boldsymbol{\xi})\}_{i=1}^{N_v^e}$ for $\boldsymbol{\xi} \in \Omega_\square$, where $\Omega_\square \subset \mathbb{R}^d$ is the reference/parent element. For this we will use the direct approach discussed in class. A basis for a simplex element of order $p$ must contain all multinomial terms of order $p$ in $d$ dimensions $\{\xi_1^{\alpha_1} \cdots \xi_d^{\alpha_d} \mid \sum_{i=1}^d \alpha_i \leqslant p\}$, so we can write our $N_v^e$ basis functions as

$$\psi_i(\boldsymbol{\xi}) = \sum_{k=1}^{N_v^e} \alpha_{ik} \prod_{j=1}^{d} \xi_j^{\Upsilon_{jk}} \tag{2}$$

where $\boldsymbol{\Upsilon} \in \mathbb{N}^{d \times N_v^e}$ is a matrix of natural numbers (including zero) with entries $\Upsilon_{ij}$, $i = 1, \ldots, d$, $j = 1, \ldots, N_v^e$, such that $\sum_{i=1}^d \Upsilon_{ij} \leqslant p$ for each $j = 1, \ldots, N_v^e$ that is used to sweep over all $N_v^e$ permissible exponents. For example:

- in the special case of $d = 2$ (triangle) and $p = 1$, we have

$$\boldsymbol{\Upsilon} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \Longrightarrow \quad \psi_i(\xi_1, \xi_2) = \alpha_{i1} + \alpha_{i2}\xi_1 + \alpha_{i3}\xi_2$$

- in the special case of $d = 2$ and $p = 2$, we have

$$\boldsymbol{\Upsilon} = \begin{bmatrix} 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 2 \end{bmatrix} \quad \Longrightarrow \quad \psi_i(\xi_1, \xi_2) = \alpha_{i1} + \alpha_{i2}\xi_1 + \alpha_{i3}\xi_2 + \alpha_{i4}\xi_1^2 + \alpha_{i5}\xi_1\xi_2 + \alpha_{i6}\xi_2^2$$

- in the special case of $d = 3$ (tetrahedron) and $p = 1$, we have

$$\boldsymbol{\Upsilon} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Longrightarrow \quad \psi_i(\xi_1, \xi_2, \xi_3) = \alpha_{i1} + \alpha_{i2}\xi_1 + \alpha_{i3}\xi_2 + \alpha_{i4}\xi_3.$$

For convenience, we introduce the function $\omega_i(\boldsymbol{\xi})$, $i = 1, \ldots, N_v^e$

$$\omega_i(\boldsymbol{\xi}) = \prod_{s=1}^{d} \xi_s^{\Upsilon_{si}},$$

so the basis functions can conveniently be expressed as $\psi_i(\boldsymbol{\xi}) = \sum_{k=1}^{N_v^e} \alpha_{ik}\omega_k(\boldsymbol{\xi})$.

Denote the $N_v^e$ nodes of the $p$th order simplex element as $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$, where $\hat{\boldsymbol{\xi}}_i = (\hat{\xi}_{1i}, \ldots, \hat{\xi}_{di})^T$. The Lagrangian property is

$$\psi_i(\hat{\boldsymbol{\xi}}_j) = \delta_{ij},$$

for $i, j = 1, \ldots, N_v^e$, which leads to

$$\sum_{k=1}^{N_v^e} \alpha_{ik}\omega_k(\hat{\boldsymbol{\xi}}_j) = \delta_{ij}$$

once the expression for $\psi_i(\boldsymbol{\xi})$ is used from (2). Let $\hat{V}_{ij} = \omega_j(\hat{\boldsymbol{\xi}}_i) = \prod_{s=1}^{d} \hat{\xi}_{si}^{\Upsilon_{sj}}$ be the Vandermonde matrix corresponding to the $d$-dimensional, $p$th order simplex evaluated at $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$, then the above constraints can be written in matrix form as $\hat{\boldsymbol{V}}\boldsymbol{\alpha}^T = \boldsymbol{I}_{N_v^e}$, where $\hat{\boldsymbol{V}}$, $\boldsymbol{\alpha}$ are the matrices with indices $\hat{V}_{ij}$, $\alpha_{ij}$, respectively, and $\boldsymbol{I}_{N_v^e}$ is the $N_v^e \times N_v^e$ identity matrix. Once we compute the coefficients, $\boldsymbol{\alpha} = \hat{\boldsymbol{V}}^{-T}$, we substitute this expression into (2) and evaluate at new points $\{\tilde{\boldsymbol{\xi}}_i\}_{i=1}^{m}$ where $\tilde{\boldsymbol{\xi}}_i = (\tilde{\xi}_{1i}, \ldots, \tilde{\xi}_{di})$ to give

$$\psi_i(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_v^e} \alpha_{ik}\omega_k(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_v^e} \hat{V}_{ki}^{-1}\omega_k(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_v^e} \hat{V}_{ki}^{-1}\tilde{V}_{jk} \tag{3}$$

where the last expression used the $d$-dimensional, $p$th order simplex Vandermonde matrix evaluated at $\{\tilde{\boldsymbol{\xi}}_i\}_{i=1}^{m}$: $\tilde{V}_{ij} = \omega_j(\tilde{\boldsymbol{\xi}}_i) = \prod_{s=1}^{d} \tilde{\xi}_{si}^{\Upsilon_{sj}}$. Therefore, if we define $Q_{ij} = \psi_i(\tilde{\boldsymbol{\xi}}_j)$, we have

$$\boldsymbol{Q} = \hat{\boldsymbol{V}}^{-T}\tilde{\boldsymbol{V}}^T,$$

where $\boldsymbol{Q}$, $\tilde{\boldsymbol{V}}$ are the matrices with indices $Q_{ij}$, $\tilde{V}_{ij}$, repsectively.

The partial derivatives of the simplex basis functions are also needed to implement the finite element method. A simple differentiation calculation reveals

$$\frac{\partial \psi_i}{\partial \xi_j}(\boldsymbol{\xi}) = \sum_{k=1}^{N_v^e} \alpha_{ik} \frac{\partial \omega_k}{\partial \xi_j}(\boldsymbol{\xi}),$$

where the partial derivatives of $\omega_i(\boldsymbol{\xi})$ are

$$\frac{\partial \omega_i}{\partial \xi_j}(\boldsymbol{\xi}) = \begin{cases} 0 & \text{if } \Upsilon_{ji} = 0 \\ \Upsilon_{ji} \xi_j^{\Upsilon_{ji}-1} \prod_{s=1, s\neq j}^{d} \xi_s^{\Upsilon_{si}} & \text{if } \Upsilon_{ji} \neq 0. \end{cases}$$

Then, the basis functions evaluated at the points $\{\tilde{\boldsymbol{\xi}}_i\}_{i=1}^m$, take the form

$$\frac{\partial \psi_i}{\partial \xi_j}(\tilde{\boldsymbol{\xi}}_k) = \sum_{l=1}^{N_v^e} \alpha_{il} \frac{\partial \omega_l}{\partial \xi_j}(\tilde{\boldsymbol{\xi}}_k) = \sum_{l=1}^{N_v^e} \hat{V}_{li}^{-1} \tilde{W}_{klj},$$

where $\tilde{W}_{ijk}$ contains the partial derivatives of the Vandermonde matrix evaluated at $\{\tilde{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$, i.e., $W_{ijk} = \frac{\partial \omega_j}{\partial \xi_k}(\tilde{\boldsymbol{\xi}}_i)$.

**Tasks for Part 1.2**

Your task is to write a function that evaluates the basis functions (and their derivatives) of a $p$th order, $d$-dimensional simplex element.

- First you need to implement a function that evaluates the Vandermonde matrix and its derivative corresponding to the $d$-dimensional, $p$th order simplex. Your function should have the following signature:

```
function [V, dV] = vander_simp(porder, x)
%VANDER_SIMP Compute NDIM-dimensional Vandermonde matrix of order PORDER
%for a  simplex and its derivative (NDIM determined from shape of X). The
%Vandermonde matrix, V, is the NX x M matrix  of multinomial terms and its
%derivative, dV, is the NX x M x NDIM matrix of the partial derivatives of
%multinomial terms, where M is the number of multinomial terms required for
%polynomial completeness (all combinations such that the sum of the
%exponents is ⩽ porder), and X are the evaluation points.
%
%Input arguments
%---------------
%   PORDER : Polynomial degree of completeness
%
%   X : Array (NDIM, NX) : Points at which to evaluate multinomials in
%     definition of Vandermonde matrix
%
%Output arguments
%----------------
%   V : Array (NX, M) : Vandermonde matrix
%
%   dV : Array (NX, M, NDIM) : Derivative of Vandermonde matrix
```

- Next, you will need to implement a function that evaluates the basis functions and their derivatives for a $d$-dimensional simplex of order $p$ given the coordinates of the element nodes `xk` and points at which to evaluate the basis `x` (these will eventually be quadrature points). Your function should have the following signature:

```
function [Q, dQ] = eval_interp_simp_lagrange(xk, x)
%EVAL_INTERP_SIMP_LAGRANGE Evaluate interpolation functions for simplex
%using Lagrange polynomials (number of spatial dimensions and polynomial
%degree of completeness determined from the nodes of the element XK).
%
%Input arguments
%---------------
%   XK : Array (NDIM, NV): Nodes of simplex element.
```

```
%
%   X : Array (NDIM, NX) : Points at which to evaluate interpolation
%     function.
%
%Output arguments
%------------------
%   Q : Array (NV, NX) : Interpolation functions for the
%     simplex element evaluated at each point in X, i.e.,
%     Q(i, k) = phi_i(X(:, k)) where phi_i(x) is the ith basis fucntion.
%
%   DQ : Array (NV, NDIM, NX) : Derivative of interpolation
%     functions evaluated at each point in X, i.e.,
%     dQ(i, j, k) = (d(phi_i)/d(x_j))(X(:, k))
```

- Check the correctness of your implementation using known properties of a Lagrangian basis:
  - Lagrangian property: $\psi_i(\hat{\boldsymbol{\xi}}_j) = \delta_{ij}$
  - Partition of unity: for any $\boldsymbol{\xi} \in \Omega_\square$

$$\sum_{i=1}^{N_v^e} \psi_i(\boldsymbol{\xi}) = 1, \qquad \sum_{i=1}^{N_v^e} \frac{\partial \psi_i}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}) = \mathbf{0}.$$

- Also check the derivatives of your basis functions are correct by comparing to a finite difference approximation

$$\frac{\partial \psi_i}{\partial \boldsymbol{\xi}_j}(\boldsymbol{\xi}) \approx \frac{\psi_i(\boldsymbol{\xi} + \epsilon \boldsymbol{e}_j) - \psi_i(\boldsymbol{\xi} - \epsilon \boldsymbol{e}_j)}{2\epsilon},$$

where $\epsilon$ is a small number (but not too small to avoid significant floating point errors), e.g., $10^{-6}$, and $\boldsymbol{e}_j \in \mathbb{R}^d$ is the $j$th canonical unit vector.

**Part 1.3** Next we will use the isoparametric concept to define basis functions of elements in the physical domain. Consider an arbitrary $d$-dimensional simplicial domain, $\Omega_e \subset \mathbb{R}^d$ (physical element), and a $d$-dimensional regular simplex, $\Omega_\square \subset \mathbb{R}^d$ (reference or parent element) (Figure 2). Define a bijective mapping between the reference and physical domains as

$$\begin{aligned} \mathcal{G}_e : \ \Omega_\square &\to \Omega_e \\ \boldsymbol{\xi} &\mapsto \boldsymbol{x} = \mathcal{G}_e(\boldsymbol{\xi}), \end{aligned}$$

and let $\mathcal{G}_e^{-1} : \Omega_e \to \Omega_\square$ denote the inverse mapping, i.e. $\boldsymbol{\xi} = \mathcal{G}_e^{-1}(\boldsymbol{x})$. In addition, it will prove convenient to introduce the regular $(d-1)$-dimensional simplex, $\Gamma_\square \subset \mathbb{R}^{d-1}$, that will be used as the reference domain for each face of the reference element, $\Omega_\square$. Let

$$\begin{aligned} \mathcal{F}_{ef} : \ \Gamma_\square &\to \Omega_e \\ \boldsymbol{r} &\mapsto \boldsymbol{x} = \mathcal{F}_{ef}(\boldsymbol{r}). \end{aligned}$$

be the bijection that maps $\Gamma_\square$ to the $f$th face of the physical element, $\partial\Omega_{ef}$ and $\mathcal{F}_{ef}^{-1} : \partial\Omega_{ef} \to \Gamma_\square$ its inverse, i.e., $\boldsymbol{r} = \mathcal{F}_{ef}^{-1}(\boldsymbol{x})$.

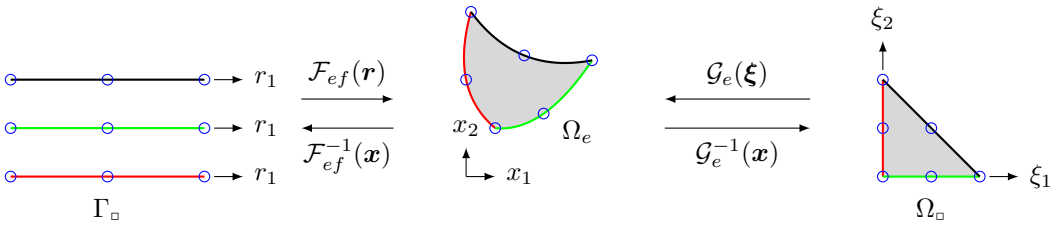


Figure 2: Mapping from $(d-1)$-dimensional reference simplex element ($\Gamma_\square$) to each face of the physical element ($\partial\Omega_{ef}$) ($\boldsymbol{x} = \mathcal{F}_{ef}(\boldsymbol{r})$) and the mapping from the $d$-dimensional reference simplex element ($\Omega_\square$) to the physical element ($\Omega_e$) ($\boldsymbol{x} = \mathcal{G}_e(\boldsymbol{\xi})$).

From this construction, we can define volume and boundary integrals over the physical domain in terms of integrals over the corresponding reference domain using a change of coordinates (volume) and surface parametrization (boundary). Consider the integrals

$$I_v = \int_{\Omega_e} \theta \, dv, \qquad I_s = \int_{\partial\Omega_{ef}} \vartheta \, ds,$$

where $\theta : \Omega_e \to \mathbb{R}$ and $\vartheta : \partial\Omega_{ef} \to \mathbb{R}$. Using the mapping to $\Omega_\square$ for the volume integral and $\Gamma_\square$ for the boundary integral, these integral become

$$\begin{aligned} I_v &= \int_{\Omega_e} \theta \, dv &&= \int_{\Omega_\square} \theta(\mathcal{G}_e(\boldsymbol{\xi})) g_e(\boldsymbol{\xi}) \, d\boldsymbol{\xi} \\ I_s &= \int_{\partial\Omega_{ef}} \vartheta \, ds &&= \int_{\Gamma_\square} \vartheta(\mathcal{F}_{ef}(\boldsymbol{r})) \sigma_{ef}(\boldsymbol{r}) \, d\boldsymbol{r}, \end{aligned} \tag{4}$$

where

$$\begin{aligned} \boldsymbol{G}_e(\boldsymbol{\xi}) &= \frac{\partial \mathcal{G}_e}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}), &\qquad g_e(\boldsymbol{\xi}) &= \det(\boldsymbol{G}_e(\boldsymbol{\xi})) \\ \boldsymbol{F}_{ef}(\boldsymbol{r}) &= \frac{\partial \mathcal{F}_{ef}}{\partial \boldsymbol{r}}(\boldsymbol{r}), &\qquad \sigma_{ef}(\boldsymbol{r}) &= \sqrt{\det\left(\boldsymbol{F}_{ef}(\boldsymbol{r})^T \boldsymbol{F}_{ef}(\boldsymbol{r})\right)}. \end{aligned}$$

Let $\{(w_i, \tilde{\boldsymbol{\xi}}_i)\}_{i=1}^{N_q^e}$ denote a quadrature rule over $\Omega_\square$, where $N_q^e$ is the number of quadrature points, $\{w_i\}_{i=1}^{N_q^e}$ are the quadrature weights, and $\{\tilde{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$ are the quadrature nodes. Similarly, let $\{(w_i^f, \tilde{\boldsymbol{r}}_i)\}_{i=1}^{N_q^f}$ denote a quadrature rule over $\Gamma_\square$, where $N_q^f$ is the number of quadrature points, $\{w_i^f\}_{i=1}^{N_q^f}$ are the quadrature weights, and $\{\tilde{\boldsymbol{r}}_i\}_{i=1}^{N_v^f}$ are the quadrature nodes. Then, integrals over the reference domains are approximated as

$$\int_{\Omega_\square} \gamma(\boldsymbol{\xi}) \, d\boldsymbol{\xi} \approx \sum_{k=1}^{N_q^e} w_k \gamma(\tilde{\boldsymbol{\xi}}_k), \qquad \int_{\Gamma_\square} \lambda(\boldsymbol{r}) \, d\boldsymbol{r} \approx \sum_{k=1}^{N_q^f} w_k^f \lambda(\tilde{\boldsymbol{r}}_k).$$

Therefore the integrals in (4) over the physical domains are approximated as

$$I_v \approx \sum_{k=1}^{N_q^e} \theta(\mathcal{G}_e(\tilde{\boldsymbol{\xi}}_k)) g_e(\tilde{\boldsymbol{\xi}}_k), \qquad I_s \approx \sum_{k=1}^{N_q^f} \vartheta(\mathcal{F}_{ef}(\tilde{\boldsymbol{r}}_k)) \sigma_{ef}(\tilde{\boldsymbol{r}}_k).$$

Next, we define basis functions over the physical element. Let $\{\psi_i\}_{i=1}^{N_v^e}$ define a basis over the $d$-dimensional reference element, where $\psi_i : \Omega_\square \to \mathbb{R}$ and $N_v^e$ is the number of nodes in the element $\Omega_\square$. Furthermore, suppose the basis is Lagrangian with nodes $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$, i.e., $\psi_i(\hat{\boldsymbol{\xi}}_j) = \delta_{ij}$. Also, let $\{\pi_i\}_{i=1}^{N_v^f}$ define a basis over the $(d-1)$-dimensional reference element, where $\pi_i : \Gamma_\square \to \mathbb{R}$ and $N_v^f$ is the number of nodes in the element $\Gamma_\square$. We also require this basis is Lagrangian with nodes $\{\hat{\boldsymbol{r}}_i\}_{i=1}^{N_v^e}$, i.e., $\pi_i(\hat{\boldsymbol{r}}_j) = \delta_{ij}$. With these definitions, we define the basis functions over $\Omega_e$ as $\{\phi_i^e\}_{i=1}^{N_v^e}$ and over $\partial\Omega_{ef}$ as $\{\varphi_i^{ef}\}_{i=1}^{N_v^f}$, where $\phi_i^e : \Omega_e \to \mathbb{R}$ and $\varphi_i^{ef} : \partial\Omega_{ef} \to \mathbb{R}$ are defined as

$$\phi_i^e(\boldsymbol{x}) = \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x})), \qquad \varphi_i^{ef}(\boldsymbol{x}) = \pi_i(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})). \tag{5}$$

From these definitions and the integration formulas in (4), any integrals involving the bases $\phi_i^e$ or $\varphi_i^{ef}$ can be written solely in terms of the corresponding reference domain basis $\psi_i(\boldsymbol{\xi})$ and $\pi_i(\boldsymbol{r})$ because the composition of a mapping and its inverse is the identity map. This implies that we only need to evaluate the basis functions associated with the reference domains $\Omega_\square$ and $\Gamma_\square$ at the quadrature nodes associated with those domains. From a simple application of the chain rule of differentiation (first equality) and the inverse function theorem (second equality), the gradient of the physical basis over $\Omega_e$ is

$$\frac{\partial \phi_i^e}{\partial x_j}(\boldsymbol{x}) = \sum_{k=1}^{d} \frac{\partial \psi_i}{\partial \xi_k}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \frac{\partial (\mathcal{G}_e^{-1})_k}{\partial x_j}(\boldsymbol{x}) = \sum_{k=1}^{d} \frac{\partial \psi_i}{\partial \xi_k}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \left[\boldsymbol{G}_e^{-1}\right]_{kj} \tag{6}$$

Finally, we define the mappings $\mathcal{G}_e$ and $\mathcal{F}_{ef}$ using the basis functions associated with $\Omega_\square$ and $\Gamma_\square$ (isoparametric)

$$\mathcal{G}_e(\boldsymbol{\xi}) = \sum_{i=1}^{N_v^e} \hat{\boldsymbol{x}}_i^e \psi_i(\boldsymbol{\xi}), \qquad \mathcal{F}_{ef}(\boldsymbol{r}) = \sum_{i=1}^{N_v^f} \hat{\boldsymbol{x}}_i^{ef} \pi_i(\boldsymbol{r}), \tag{7}$$

where $\{\hat{\boldsymbol{x}}_i^e\}_{i=1}^{N_v^e}$ are the nodes associated with the element $\Omega_e$ and $\{\hat{\boldsymbol{x}}_i^{ef}\}_{i=1}^{N_v^f}$ are the nodes associated with its face $\partial\Omega_{ef}$. The element and face nodes are related using the face-to-element vertex mapping from Part 1.1: $\hat{\boldsymbol{x}}_i^{ef} = \hat{\boldsymbol{x}}_{\Theta_{if}}^e$.

**Tasks for Part 1.3**

Your task is to write a function that evaluates all relevant isoparametric quantities and one that completely defines a simplex element in the reference domain. Then you will use these functions to approximate moments (integrals) over elements with regular and non-regular shapes.

- Derive the quantities $\boldsymbol{G}_e(\boldsymbol{\xi})$, $\boldsymbol{F}_{ef}(\boldsymbol{r})$, for the isoparametric mapping given in (7). Your answer should be in terms of the element coordinates $\{\hat{\boldsymbol{x}}_i^e\}_{i=1}^{N_v^e}$, the face-to-element vertex mapping $\boldsymbol{\Theta}$, and the basis functions $\{\psi_i\}_{i=1}^{N_v^e}$, $\{\pi_i\}_{i=1}^{N_v^f}$.
- Using the node numbering in Figure 1, write out the isoparametric boundary mapping $\mathcal{F}_{ef}(\boldsymbol{r})$ for each face for the case $d = 2$, $p = 2$. Your answer should be in terms of the nodes of $\Omega_e$, $\{\boldsymbol{x}_i\}_{i=1}^6$, and the basis functions associated with $\Gamma_\square$: $\{\pi_i(\boldsymbol{r})\}_{i=1}^3$, i.e., the basis functions for a face of the element in their reference domain.
- Implement a function that evaluates all relevant isoparametric quantities. Your function should have the following signature:

```
function [xq, detG, Gi, xqf, sigf] = eval_isoparam_quant(xe, Q, dQdz, Qf, dQfdr, f2v)
%EVAL_ISOPARAM_QUANT Evaluate isoparametric quantities for a single element
%given the nodal coordinates of the element in physical space (XE) and the
%basis functions (and their derivatives) over the element and its faces.
%
%Input arguments
%———————————————
%  XE, Q, DQDZ, QF, DQFDR, F2V : See notation.m
%
%Output arguments
%————————————————
%  XQ, DETG, GI, XQF, SIGF : See notation.m
```

  You will test this function below when you use the isoparametric concept to compute the area, centroid, and surface area of known geometries (simplex and hypercube).

- Implement a function that completely defines a simplex element in the reference domain, $\Omega_\square$ and $\Gamma_\square$, as a MATLAB structure. This structure should contain: the nodal coordinates of $\Omega_\square$ ($\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_v^e}$) and $\Gamma_\square$ ($\{\hat{\boldsymbol{r}}_i\}_{i=1}^{N_v^f}$), the face-to-vertex mapping ($\boldsymbol{\Theta}$) and unit normals of each face ($\{\boldsymbol{N}_i\}_{i=1}^d$), a quadrature rule for $\Omega_\square$ ($\{(w_k, \tilde{\boldsymbol{\xi}}_i)\}_{i=1}^{N_q^e}$) and $\Gamma_\square$ ($\{(w_k^f, \tilde{\boldsymbol{r}}_i)\}_{i=1}^{N_q^f}$), and the basis functions evaluated at the appropriate quadrature nodes for $\Omega_\square$ ($\psi_i(\tilde{\boldsymbol{\xi}}_j)$) and $\Gamma_\square$ ($\pi_i(\tilde{\boldsymbol{r}}_j)$). To create the quadrature rule, use the functions `create_quad_onedim_gaussleg` (creates quadrature rule in 1D), `create_quad_hcube_from_onedim` (creates quadrature rule for hypercube from 1D quadrature rule), and `create_quad_simp_from_hcube` (creates quadrature rule for simplex from quadrature rule for hypercube) provided in the project code distribution. To evaluate the basis functions, use the function `eval_interp_simp_lagrange` written in Part 1.2. Your function should have the following signature:

```
function [geom] = create_geom_simp(ndim, porder, nquad_per_dim)
```

```
%CREATE_GEOM_SIMP Create structure defining the simplex element.
%
%Input arguments
%---------------
%   NDIM, PORDER : See notation.m
%
%   NQUAD_PER_DIM : number : Number of quadrature nodes per dimension
%
%Output arguments
%---------------
%   GEOM : See notation.m
```

You may find the function `create_geom_hcube` helpful; it defines the corresponding MATLAB structure defining a hypercube element. Notice that this function uses `eval_interp_hcube_from_onedim` from Homework 3 to define the basis functions.

- Check the quadrature formulas you were provided by computing the following moments of $\Omega_\square$ and $\Gamma_\square$ using quadrature and comparing to known formulas for the area and centroid of simplex and hypercube domains

$$V(\Omega_\square) = \int_{\Omega_\square} d\boldsymbol{\xi}, \qquad \boldsymbol{c}(\Omega_\square) = \frac{1}{V(\Omega_\square)} \int_{\Omega_\square} \boldsymbol{\xi}\, d\boldsymbol{\xi}, \qquad V(\Gamma_\square) = \int_{\Gamma_\square} d\boldsymbol{r}, \qquad \boldsymbol{c}(\Gamma_\square) = \frac{1}{V(\Gamma_\square)} \int_{\Gamma_\square} \boldsymbol{r}\, d\boldsymbol{r}, \qquad .$$

  Make sure your quadrature rule has a sufficient number of points to compute the integrals exactly. Only consider the $p = 1$ elements since these quantities do not depend on the degree of completeness of the element. Consider both simplices and hypercubes elements in spatial dimensions $d = 1, 2, 3$.

- Compute the volume, centroid, and surface area

$$V(\Omega_e) = \int_{\Omega_e} dv, \qquad \boldsymbol{c}(\Omega_e) = \frac{1}{V(\Omega_e)} \int_{\Omega_e} \boldsymbol{x}\, dv, \qquad S(\Omega_e) = \int_{\partial\Omega_e} ds,$$

  of (straight-sided) simplices and hypercubes in $d = 1, 2, 3$ you choose (should be different than the parent element) using the isoparametric quantities from `eval_isoparam_quant` and the `geom` structure. This will serve as a test for the functions you wrote in this section. Once your code returns the correct moments for the straight-sided elements, compute the volume, centroid, and surface area of the curved (quadratic) triangle in Figure 2). The nodes of the triangle (in the order given by Figure 1) are $\{(0.0, 0.0), (0.5, 0.15), (1.0, 0.7), (-0.3, 0.5), (0.3, 0.75), (-0.25, 1.2)\}$.

**Part 1.4** In this section, we will use a finite element mesh (Figure 3), a collection of finite elements such as the simplex and hypercube elements defined in Part 1.3, to evaluate integrals over complex domains $\Omega \subset \mathbb{R}^d$:

$$I_v = \int_\Omega \theta\, dv, \qquad I_s = \int_{\partial\Omega} \vartheta\, ds. \tag{8}$$

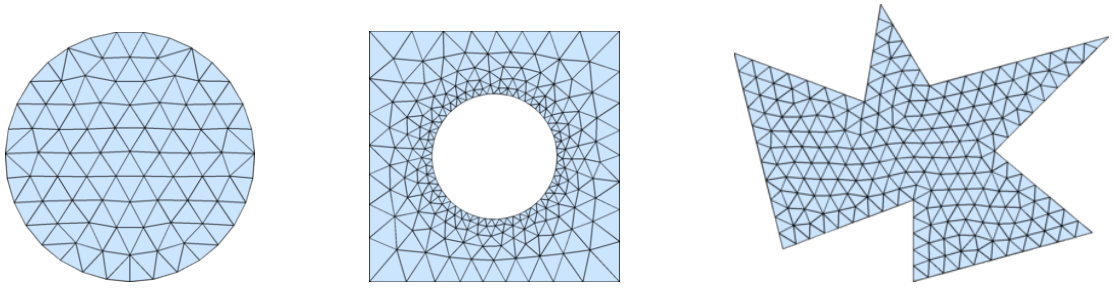

Figure 3: Mesh of a circle, square with circular hole, and arbitrary polygon using triangular elements.

We will describe our unstructured mesh using the arrays: `xcg`, `e2vcg`, and `e2bnd`, defined as

```
%   XCG : 2D array (NDIM, NNODE) : The position of the nodes in the mesh.
%     The (i, j)—entry is the position of global node j in the ith dimension.
%     The global node numbers are defined by the columns of this matrix, e.g.,
%     the node at xcg(:, j) is the jth node of the mesh.
%
%   E2VCG : 2D array (NNODE_PER_ELEM, NELEM): The connectivity of the
%     mesh. The (:, e)—entries are the global node numbers of the nodes
%     that comprise element e. The local node numbers of each element are
%     defined by the columns of this matrix, e.g., e2vcg(i, e) is the
%     global node number of the ith local node of element e.
%
%   E2BND: 2D array (NFACE_PER_ELEM, NELEM): The mapping between element
%     boundaries and global boundaries. The (f, e)—entry is the global
%     boundary tag on which the fth face of element e lies. If face f of
%     element e does not touch the global boundary, e2bnd(f, e) is NaN.
```

With this concept of a mesh, the integrals in (8) can be re-written as

$$I_v = \sum_{e=1}^{N_e} \int_{\Omega_e} \theta \, dv, \qquad I_s = \sum_{e=1}^{N_e} \sum_{f=1}^{N_f} \int_{\partial\Omega_{ef} \cap \partial\Omega} \vartheta \, ds = \sum_{e=1}^{N_e} \sum_{f=1}^{N_f} \mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \varnothing\}} \int_{\partial\Omega_{ef}} \vartheta \, ds,$$

where $N_e$ is the number of elements in the mesh and $\mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \varnothing\}}$ is the indicator function that takes the value of 0 if $\partial\Omega_{ef} \cap \partial\Omega = \varnothing$ and 1 otherwise. Notice that this will, in general, only be an approximation to an integral since the region covered by the union of all finite elements will not exactly overlap with $\Omega$, except in special cases.

**Tasks for Part 1.4**

With the isoparametric function and element structures defined in Part 1.3, you can create a structure array `mesh_data` that contains the isoparametric quantities for every element in a mesh defined by `xcg`, `e2vcg`, `e2bnd` using the function `create_mesh_data` provided. This will create all relevant data for each element of your mesh. Using this structure array, we are interested computing the following integrals

$$V(\Omega) = \int_\Omega dv, \qquad \boldsymbol{c}(\Omega) = \frac{1}{V(\Omega)} \int_\Omega \boldsymbol{x} \, dv, \qquad S(\Omega) = \int_{\partial\Omega} ds,$$

i.e., the volume, centroid, and surface area of $\Omega$.

- Write a function that computes the volume, centroid, and surface area of a domain described by a mesh. Your function should have the following signature:

```
function [V, C, S] = compute_volume_centroid_surfarea(geom, mesh_data)
%COMPUTE_VOLUME_CENTROID_SURFAREA Compute the volume, centroid, and surface
%area of a domain (approximated) by the mesh described by GEOM, MESH_DATA.
%
%Input arguments
%———————————
%  GEOM, MESH_DATA : See notation.m
%
%Output arguments
%———————————
%  V : number : Volume of domain
%
%  C : Array (NDIM,) : Centroid of domain
%
%  S : number : Surface area of domain
```

- Consider a domain $\Omega = [0,1]^d$ for $d = 1, 2, 3$. Create a mesh of this domain with $2^d$ $p = 1$ hypercube elements using `create_unif_mesh_hcube` and create the corresponding `mesh_data` structure array. Compute the volume, centroid, and surface area of $\Omega$ by integrating the appropriate quantities over

the mesh and compare to the known volume, centroid, and surface area of a hypercube. For the case of $d = 2$, use the function `split_quad_mesh_into_tri_mesh` to split quadrilateral elements into triangular ones and repeat the integral calculations. This will serve as a test for parts of your code (quadrature, isoparametric mapping of integrals). Use `visualize_fem` to plot the mesh.

- Repeat the previous task for the unit hypersphere in $d = 2, 3$ dimensions (circle for $d = 2$, sphere for $d = 3$). Since this geometry has a curved boundary, you will need to use more elements and the polynomial degree will have an impact on the accuracy of the integral (provided we use a curved, high-order mesh). The function `create_mesh_hsphere` creates a mesh of curved (hypercube) elements of a given polynomial degree by mapping a hypercube to a hypersphere. Be aware that even though we are using curved elements, the plotting utility `visualize_fem.m` only plots straight-sided elements. Consider polynomial degrees $p = 1, 2, 3, 4$. How many elements are needed to get a high-quality approximation of the volume and surface area? Make sure you use enough quadrature nodes so your element integral computations are exact. For the case of $d = 2$, use the function `split_quad_mesh_into_tri_mesh` to split quadrilateral elements into triangular ones and repeat the integral calculations.

- Compute the volume, centroid, and surface area of the Batman symbol and ND logo (Figure 4). Use a mesh of $p = 1$ simplex elements provided in the `_mesh/_meshes` directory. The function `load_mesh` loads the appropriate mesh given its filename prefix (`'batman'` for the Batman domain, `'nd'` for the Notre Dame domain), the element type (`'simp'` for simplex elements and `'hcube'` for hypercube elements), and the polynomial degree. Use `visualize_fem` to plot the mesh. Plot the centroid of the domain as a sanity check for the centroid computation.

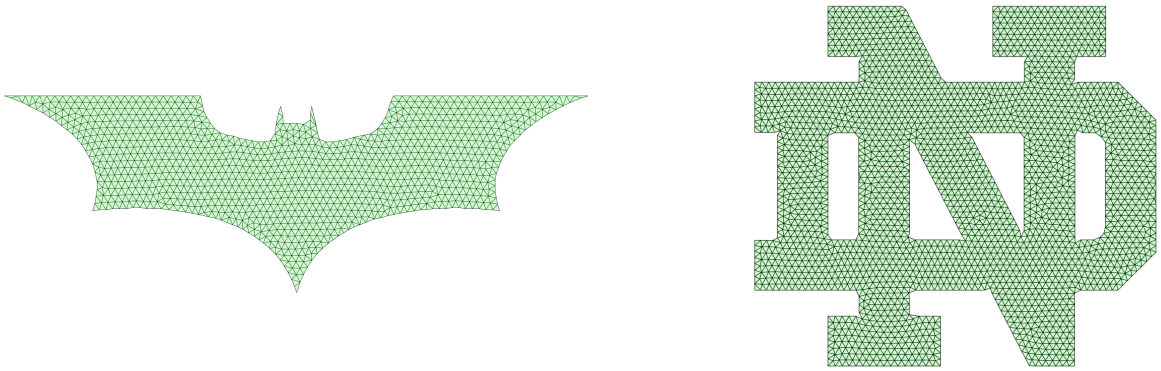

Figure 4: Simplicial mesh ($d = 2$) of the batman symbol and Notre Dame logo.

- Compute the volume, centroid, and surface area of the cow (filename prefix `'cow'`), dragon (filename prefix `'dragon'`), and sculpture (filename prefix `'scultp10kv'`) domains (Figure 5). Use $p = 1$ simplex elements and `visualize_fem` to plot the mesh. Plot the centroid of the domain for the sculpture mesh as a sanity check for the centroid computation.

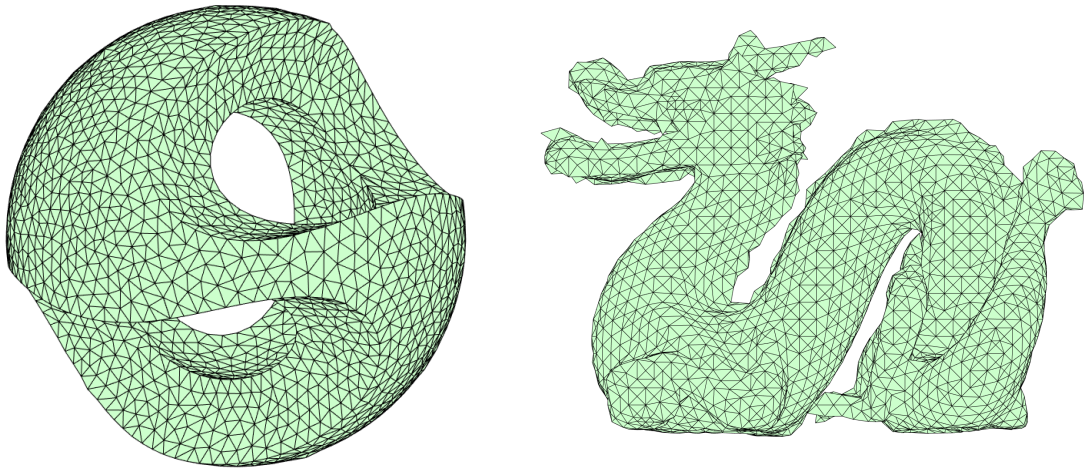

Figure 5: Simplicial mesh ($d = 3$) of a sculpture and dragon.

**Part 2:** (50 points) To support our goal in developing a general FEM code, we will extend the finite element code written in your homework assignment to handle general, second-order partial differential equations, including those with non-homogeneous natural boundary conditions and nonlinearities.

Let us consider a general, second-order, static partial differential equation defined on the domain $\Omega \subset \mathbb{R}^d$

$$\nabla \cdot \boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) + \boldsymbol{S}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) = 0, \quad \text{in } \Omega, \qquad \boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu})\boldsymbol{n} = \bar{\boldsymbol{q}} \quad \text{on } \partial\Omega, \tag{9}$$

where $\boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) \in \mathbb{R}^{N_c \times d}$ is a nonlinear flux function, $\boldsymbol{S}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) \in \mathbb{R}^{N_c}$ is a nonlinear source term, $\boldsymbol{u}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the primary solution variable, $\boldsymbol{\nu}(\boldsymbol{x}) \in \mathbb{R}^m$ are parameters to the flux function and source term, $\bar{\boldsymbol{q}}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the value of the natural boundary condition, $\boldsymbol{n}(\boldsymbol{x}) \in \mathbb{R}^d$ is the outward unit normal, $N_c$ is the number of components in the primary variable $\boldsymbol{u}$, and $\boldsymbol{x} \in \Omega$. Re-writing the PDE and boundary condition using indicial notation, we have

$$F_{ij,j} + S_i = 0 \quad \text{in } \Omega, \qquad F_{ij} n_j = \bar{q}_i \quad \text{on } \partial\Omega \tag{10}$$

for $i = 1, \ldots, N_c$.

We solely consider a problem with natural boundary conditions to facilitate a convenient and general implementation. For positions on the boundary where an essential boundary condition is prescribed, the value of the natural boundary conditions for the corresponding degree of freedom $\bar{q}_i$ will be set to zero and static condensation will be applied. This has the effect of eliminating the boundary term at this position and degree of freedom and results in exactly the same weak formulation as setting the corresponding test function to zero.

We derived the weak formulation for this general PDE (modulo the sign of the source term) in Homework 2 as

$$\int_\Omega (-w_{i,j} F_{ij} + w_i S_i)\, dv + \int_{\partial\Omega} w_i \bar{q}_i \, ds = 0 \tag{11}$$

where $\boldsymbol{w}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the vector of test functions for $\boldsymbol{x} \in \Omega$, arguments are dropped for brevity, and indicial notation is used (sum over $i, j = 1, \ldots, N_c$ is implied by repeated index). Breaking the weak form into a sum of integrals over element domains $\Omega_e$, we have

$$\sum_{e=1}^{N_e} \int_{\Omega_e} (-w_{i,j} F_{ij} + w_i S_i)\, dv + \sum_{e=1}^{N_e} \int_{\partial\Omega_e \cap \partial\Omega} w_i \bar{q}_i \, ds = 0, \tag{12}$$

from which we can identify the element contribution to the weak form as

$$B^e(\boldsymbol{w}, \boldsymbol{u}) = \int_{\Omega_e} (w_i S_i - w_{i,j} F_{ij})\, dv + \int_{\partial\Omega_e \cap \partial\Omega} w_i \bar{q}_i \, ds.$$

Transferring the integrals to the corresponding reference domain using the isoparametric mapping leads to

$$B^e(\boldsymbol{w},\boldsymbol{u}) = \int_{\Omega_\square} \left( w_i S_i - \frac{\partial w_i}{\partial x_j} \right) g_e \, dV + \sum_{f=1}^{N_f} \mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \varnothing\}} \int_{\Gamma_\square} w_i \bar{q}_i \sigma_{ef} \, ds \tag{13}$$

where arguments are dropped for brevity (all terms are evaluated at $\mathcal{G}_e(\boldsymbol{\xi})$).

Let $\{\Omega_e\}_{e=1}^{N_e}$ define a mesh of $\Omega \subset \mathbb{R}^d$, i.e., $\Omega = \bigcup_{e=1}^{N_e} \Omega_e$ and $\Omega_e \cap \Omega_{e'} = \varnothing$ if $e \neq e'$, with $N_e$ elements and $N_v$ nodes. Define the global solution vector $\hat{\boldsymbol{U}} \in \mathbb{R}^{N_d}$ as the vector containing all degrees of freedom (solution variables) associated with a given mesh, where $N_d$ is the number of degrees of freedom in the mesh. Similarly, let $\hat{\boldsymbol{U}}^e \in \mathbb{R}^{N_d^e}$ contain all local degrees of freedom associated with an element, where $N_d^e$ is the number of degrees of freedom in each element. The global and local solution vectors are related by the `ldof2gdof` matrix (discussed in class) as

$$\hat{U}_i^e = \hat{U}_j, \quad \text{where } j = \texttt{ldof2gdof}(i,e)$$

for $i = 1,\dots,N_v^e$, $e = 1,\dots,N_e$. See Figure 6 for a schematic of the global and local degrees of freedom for a scalar PDE in one dimension on a mesh with three $p = 2$ elements. In this case, the `ldof2gdof` matrix is

$$\texttt{ldof2gdof} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 3 & 5 & 7 \end{bmatrix}.$$

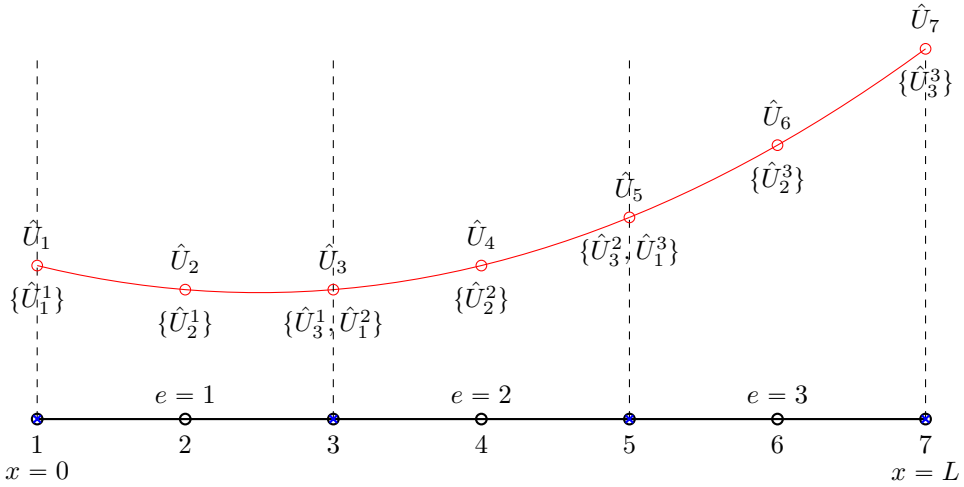


Figure 6: Schematic of finite element mesh and global and local degrees of freedom for a scalar PDE in one dimension.

Define basis functions $\boldsymbol{\Psi}(\boldsymbol{\xi}) \in \mathbb{R}^{N_d^e \times N_c}$ over the reference element $\Omega_\square$ that maps the local degrees of freedom associated with element $e$ to the primary variables evaluated at $\boldsymbol{x} = \mathcal{G}_e(\boldsymbol{\xi})$:

$$\boldsymbol{u}(\boldsymbol{x})|_{\Omega_e} = \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x}))^T \hat{\boldsymbol{U}}^e, \qquad u_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{N_d^e} \Psi_{ji}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \hat{U}_j^e. \tag{14}$$

Similarly, define basis functions $\boldsymbol{\Pi}(\boldsymbol{r}) \in \mathbb{R}^{N_d^f \times N_c}$ over the reference element $\Gamma_\square$ that maps the local degrees of freedom associated with face $f$ of element $e$ to the primary variables evaluated at $\boldsymbol{x} = \mathcal{F}_{ef}(\boldsymbol{r})$:

$$\boldsymbol{u}(\boldsymbol{x})|_{\partial\Omega_{ef}} = \boldsymbol{\Pi}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x}))^T \hat{\boldsymbol{U}}^{ef}, \qquad u_i(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{j=1}^{N_d^f} \Pi_{ji}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})) \hat{U}_j^{ef}, \tag{15}$$

where $\hat{\boldsymbol{U}}^{ef} \in \mathbb{R}^{N_d^f}$ is a vector of all degrees of freedom (primary variables) associated with face $f$ of element $e$ and $N_d^f$ are the number of local degrees of freedom associated with an element face. Since the FEM is a Galerkin approximation, the test functions are interpolated using the same basis functions, i.e.,

$$\boldsymbol{w}(\boldsymbol{x})|_{\Omega_e} = \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x}))^T \hat{\boldsymbol{W}}^e, \qquad \boldsymbol{w}(\boldsymbol{x})|_{\partial\Omega_{ef}} = \boldsymbol{\Pi}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x}))^T \hat{\boldsymbol{W}}^{ef}, \tag{16}$$

where $\hat{\boldsymbol{W}}^e \in \mathbb{R}^{N_d^e}$ is a vector of all degrees of freedom (test variables) associated with element $e$ and $\hat{\boldsymbol{W}}^{ef} \in \mathbb{R}^{N_d^e}$ is a vector of all degrees of freedom (test variables) associated with face $f$ of element $e$. The specific form of $\boldsymbol{\Psi}$ and $\boldsymbol{\Pi}$ will depend on the particular equation under consideration. The degrees of freedom corresponding to the entire element and those specific to face $f$ are related as

$$\hat{U}_i^{ef} = \hat{U}^e_{\Lambda_{if}}, \qquad \hat{W}_i^{ef} = \hat{W}^e_{\Lambda_{if}},$$

for $i = 1, \ldots, N_d^f$, where $\boldsymbol{\Lambda} \in \mathbb{N}^{N_d^f \times N_f}$ is the maxtrix of natural numbers whose columns define the element degrees of freedom associated with a particular face, i.e., $\Lambda_{if}$ is the element degree of freedom corresponding to the $i$th degree of freedom on face $f$. Using this relationship between element and face degrees of freedom, the primary variable and test function on face $f$ can be written in terms of $\hat{\boldsymbol{U}}_i^e$ and $\hat{\boldsymbol{W}}_i^e$, respectively,

$$u_i(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{j=1}^{N_d^f} \Pi_{ji}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x}))\hat{U}^e_{\Lambda_{jf}}, \qquad w_i(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{j=1}^{N_d^f} \Pi_{ji}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x}))\hat{W}^e_{\Lambda_{jf}}. \tag{17}$$

Substitute the expansions in (14)-(17) into the element weak form and identify the term multiplying $\hat{W}_l^e$ as the $l$th component of the element residual, $\hat{\boldsymbol{R}}^e(\hat{\boldsymbol{U}}^e)$

$$\hat{R}_l^e(\hat{\boldsymbol{U}}_e) = \int_{\Omega_\square} \left( \Psi_{li} S_i - \frac{\partial \Psi_{li}}{\partial x_j} F_{ij} \right) g_e \, dV + \begin{cases} \displaystyle\sum_{f=1}^{N_f} \mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \varnothing\}} \int_{\Gamma_\square} \Pi_{ai} \bar{q}_i \sigma_{ef} \, ds & \text{if } \exists a \text{ such that } l = \Lambda_{af} \\ 0 & \text{otherwise,} \end{cases}$$

where $\dfrac{\partial \Psi_{ij}}{\partial x_k} = \dfrac{\partial \Psi_{ij}}{\partial \xi_s} \left( \dfrac{\partial \mathcal{G}_e}{\partial \boldsymbol{\xi}} \right)^{-1}_{sk}$ and summation implied over repeated index. Then, the element Jacobian (derivative of the element residual with respect to $\hat{\boldsymbol{U}}^e$), $\frac{\partial \hat{\boldsymbol{R}}^e}{\partial \hat{\boldsymbol{U}}^e}$, is

$$\frac{\partial \hat{R}_l^e}{\partial \hat{U}_r^e}(\hat{\boldsymbol{U}}^e) = \int_{\Omega_\square} \left( \Psi_{li} \left( \frac{\partial S_i}{\partial u_t} \Psi_{rt} + \frac{\partial S_i}{\partial q_{ts}} \frac{\partial \Psi_{rt}}{\partial x_s} \right) - \frac{\partial \Psi_{li}}{\partial x_j} \left( \frac{\partial F_{ij}}{\partial u_t} \Psi_{rt} + \frac{\partial F_{ij}}{\partial q_{ts}} \frac{\partial \Psi_{rt}}{\partial x_s} \right) \right) g_e \, dV, \tag{18}$$

where $\boldsymbol{q} = \nabla \boldsymbol{u}$, indicial notation is used (sum over repeated indices is implied), and arguments have been dropped.

Once all element residuals and Jacobians have been computed, they are assembled into a global nonlinear system

$$\hat{\boldsymbol{R}}(\hat{\boldsymbol{U}}) = 0$$

where $\hat{\boldsymbol{R}}(\hat{\boldsymbol{U}})$ is the residual of the nonlinear system that results from assembling the element residuals $\hat{\boldsymbol{R}}^e$ into a global vector. Similarly, the Jacobian of the global nonlinear system $\frac{\partial \hat{\boldsymbol{R}}}{\partial \hat{\boldsymbol{U}}}(\hat{\boldsymbol{U}})$ comes from the assembly of the element Jacobian matrices $\frac{\partial \hat{\boldsymbol{R}}^e}{\partial \hat{\boldsymbol{U}}^e}(\hat{\boldsymbol{U}}^e)$. The global solution vector is then partitioned into prescribed components $\hat{\boldsymbol{U}}_d$ (those with an essential boundary condition) and free components $\hat{\boldsymbol{U}}_f$: $\hat{\boldsymbol{U}} = (\hat{\boldsymbol{U}}_f^T, \hat{\boldsymbol{U}}_d^T)^T$. The global residual and Jacobian are partitioned similarly

$$\hat{\boldsymbol{R}}(\hat{\boldsymbol{U}}) = \begin{bmatrix} \hat{\boldsymbol{R}}_f(\hat{\boldsymbol{U}}_f; \hat{\boldsymbol{U}}_d) \\ \hat{\boldsymbol{R}}_d(\hat{\boldsymbol{U}}_d; \hat{\boldsymbol{U}}_f) \end{bmatrix}, \qquad \frac{\partial \hat{\boldsymbol{R}}}{\partial \hat{\boldsymbol{U}}}(\hat{\boldsymbol{U}}) = \begin{bmatrix} \frac{\partial \hat{\boldsymbol{R}}_f}{\partial \hat{\boldsymbol{U}}_f}(\hat{\boldsymbol{U}}_f; \hat{\boldsymbol{U}}_d) & \frac{\partial \hat{\boldsymbol{R}}_f}{\partial \hat{\boldsymbol{U}}_d}(\hat{\boldsymbol{U}}_f; \hat{\boldsymbol{U}}_d) \\ \frac{\partial \hat{\boldsymbol{R}}_d}{\partial \hat{\boldsymbol{U}}_f}(\hat{\boldsymbol{U}}_d; \hat{\boldsymbol{U}}_f) & \frac{\partial \hat{\boldsymbol{R}}_d}{\partial \hat{\boldsymbol{U}}_d}(\hat{\boldsymbol{U}}_d; \hat{\boldsymbol{U}}_f) \end{bmatrix}.$$

The variables $\hat{\boldsymbol{U}}_d$ are *known* since an essential boundary condition is prescribed at these degrees of freedom, we can disregard the corresponding equations in the residual, reducing the nonlinear system to

$$\hat{\boldsymbol{R}}_f(\hat{\boldsymbol{U}}_f; \hat{\boldsymbol{U}}_d) = 0,$$

with Jacobian matrix $\frac{\partial \hat{\boldsymbol{R}}_f}{\partial \hat{\boldsymbol{U}}_f}(\hat{\boldsymbol{U}}_f; \hat{\boldsymbol{U}}_d)$. This system can be solved for the unknown $\hat{\boldsymbol{U}}_f$ using, e.g., the Newton-Raphson method, and the global solution vector re-assembled as $\hat{\boldsymbol{U}} = (\hat{\boldsymbol{U}}_f^T, \hat{\boldsymbol{U}}_d^T)^T$.

Before specializing this general formulation to specific PDEs, we introduce another critical assembled quantity that is useful in integrating quantities over the domain and for unsteady problems: the mass matrix. Suppose we would like to compute the following integral

$$I = \int_\Omega \rho(\boldsymbol{x}) \boldsymbol{u}(\boldsymbol{x})^T \boldsymbol{u}(\boldsymbol{x}) \, dv, \tag{19}$$

where $\rho(\boldsymbol{x}) \in \mathbb{R}$ for $\boldsymbol{x} \in \Omega$ is a weighting functions and $\boldsymbol{u}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the PDE solution in (10). Using the finite element approximation in (14), we have

$$I = \sum_{e=1}^{N_e} \int_{\Omega_e} \rho(\boldsymbol{x}) (\hat{\boldsymbol{U}}^e)^T \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x}))^T \hat{\boldsymbol{U}}^e \, dv = \sum_{e=1}^{N_e} (\hat{\boldsymbol{U}}^e)^T \boldsymbol{M}^e \hat{\boldsymbol{U}}^e, \tag{20}$$

where

$$\boldsymbol{M}^e = \int_{\Omega_e} \rho(\boldsymbol{x}) \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x}))^T \, dv = \int_{\Omega_\square} \rho(\mathcal{G}_e(\boldsymbol{\xi})) \boldsymbol{\Psi}(\boldsymbol{\xi}) \boldsymbol{\Psi}(\boldsymbol{\xi})^T g_e(\boldsymbol{\xi}) \, dV$$

is the element mass matrix. After assembly, this reduces to

$$I = \hat{\boldsymbol{U}}^T \boldsymbol{M} \hat{\boldsymbol{U}}, \tag{21}$$

where $\hat{\boldsymbol{U}} \in \mathbb{R}^{N_d}$ is the global (assembled) solution vector and $\boldsymbol{M} \in \mathbb{R}^{N_d \times N_d}$ is the assembled mass matrix.

To this point, we have operated in a fairly abstract setting to ensure our framework will be applicable to a variety of PDEs. For each PDE considered, we must define the following quantities:

- the PDE variable $\boldsymbol{u}(\boldsymbol{x})$, flux function $\boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu})$, source term $\boldsymbol{S}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu})$, and parameter $\boldsymbol{\nu}(\boldsymbol{x})$,
- the local element solution vector $\hat{\boldsymbol{U}}^e \in \mathbb{R}^{N_d^e}$ and the local element face solution vector $\hat{\boldsymbol{U}}^{ef} \in \mathbb{R}^{N_d^f}$,
- the basis functions $\boldsymbol{\Psi}(\boldsymbol{\xi})$ and $\boldsymbol{\Pi}(\boldsymbol{r})$ over the reference domain, and
- the face-to-element degree of freedom mapping $\boldsymbol{\Lambda}$.

For concreteness, we will consider PDE0 from Homework 3

$$-\frac{d^2u}{dx^2} - u + x^2 = 0, \quad 0 < x < 1$$
$$u(0) = 0, \left(\frac{du}{dx}\right)\Big|_{x=1} = 1.$$

We can immediately identify this as a scalar PDE ($N_c = 1$) in one spatial dimension ($d = 1$). To put this PDE in the form of (10), we identify the PDE variables as $\boldsymbol{u} = u$ and its gradient as $\nabla \boldsymbol{u} = \frac{du}{dx}$. The flux function and source term are

$$\boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) = -\frac{du}{dx}, \qquad \boldsymbol{S}(\boldsymbol{u}, \nabla \boldsymbol{u}; \boldsymbol{\nu}) = -u + x^2,$$

the natural boundary condition is $\bar{\boldsymbol{q}}(\boldsymbol{x}) = \mathbf{1}_{\{x=1\}}$, and the parameter vector is $\boldsymbol{\nu}(\boldsymbol{x}) = x^2$. Let $\hat{u}_i^e$ denote the solution at the $i$th node of element $e$ (element node numbering) for $i = 1, \ldots, N_v^e$ and let $\hat{\boldsymbol{U}}^e = (\hat{u}_1^e, \ldots, \hat{u}_{N_v^e}^e)^T$ be the local (element) solution vector. Similarly, let $\hat{u}_i^{ef}$ denote the solution at the $i$th node of face $f$ of element $e$ (face node numbering) for $i = 1, \ldots, N_v^f$ and let $\hat{\boldsymbol{U}}^{ef} = (\hat{u}_1^{ef}, \ldots, \hat{u}_{N_v^f}^{ef})^T$. These quantities are related by $\hat{u}_j^{ef} = \hat{u}^e_{\Theta_{jf}}$ for $j = 1, \ldots, N_v^f$, which implies $\boldsymbol{\Lambda} = \boldsymbol{\Theta}$. For this problem, we approximate the element-wise solution as

$$u(\boldsymbol{x})|_{\Omega_e} = \sum_{i=1}^{N_v^e} \hat{u}_i^e \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x})), \qquad u(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{i=1}^{N_v^f} \hat{u}_i^{ef} \pi_i(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})),$$

where $\psi_i(\boldsymbol{\xi})$ and $\pi_i(\boldsymbol{r})$ are the basis functions defined over a particular geometric element, which implies

$$\boldsymbol{\Psi}(\boldsymbol{\xi}) = \begin{bmatrix} \psi_1(\boldsymbol{\xi}) \\ \vdots \\ \psi_{N_v^e}(\boldsymbol{\xi}) \end{bmatrix}, \qquad \boldsymbol{\Pi}(\boldsymbol{r}) = \begin{bmatrix} \pi_1(\boldsymbol{r}) \\ \vdots \\ \pi_{N_v^f}(\boldsymbol{r}) \end{bmatrix}. \tag{22}$$

**Tasks for Part 2**

Your task is to generalize the FEM code you developed in your homework for the general, second-order PDE in (10) and test your code using PDE0.

- Derive the partial derivatives of the flux function and source term $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}}$, $\frac{\partial \boldsymbol{F}}{\partial \nabla \boldsymbol{u}}$, $\frac{\partial \boldsymbol{S}}{\partial \boldsymbol{u}}$, $\frac{\partial \boldsymbol{S}}{\partial \nabla \boldsymbol{u}}$ for PDE0.
- Implement a function with the following signature that returns the flux function, the source term, and their partial derivatives given the primary variable, its gradient, and parameters:

```
function [F, dFdu, dFdq, S, dSdu, dSdq] = eval_pntws_contrib_pde0(u, q, eqn_pars)
%EVAL_PNTWS_CONTRIB_PDE0 Evaluate the flux function and source term (and
%their derivatives) that define PDE0.
%
% Input arguments
% ———————————
%   U : Array (NC,) : Primary variables
%
%   Q : Array (NC, NDIM) : Gradient of primary variables
%
%   EQN_PARS : Array (M,) : Parameters to flux function and source term
%
% Output arguments
% ————————————————
%   F, DFDU, DFDQ, S, DSDU, DSDQ : See notation.m
```

- Implement a function with the following signature that creates the solution bases $\boldsymbol{\Psi}(\boldsymbol{\xi})$, $\boldsymbol{\Pi}(\boldsymbol{r})$ and the derivative $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}$, evaluated at the appropriate quadrature points, and the face-to-element degree of freedom mapping $\boldsymbol{\Lambda}$:

```
function [T, dTdz, Tf, ldof2fdof] = create_soln_basis_ldof2fdof_pde0(Q, dQdz, Qf, f2v)
%CREATE_SOLN_BASIS_LDOF2FDOF_PDE0 Create solution basis (element and face)
%and its derivative w.r.t. reference coordinates and the face—to—element
%degree of freedom mapping for PDE0.
%
% Input arguments
% ———————————
%   Q, DQDZ, QF, F2V : See notation.m
%
% Output arguments
% ————————————————
%   T, DTDZ, TF, LDOF2FDOF : See notation.m
```

Once you have this function, you will be able to create the `elem` structure and `mesh_data` structure array (see `notation.m`), which will be central to your finite element code since they contain all required information (both generic and equation-specific information) to evaluate the element residual and Jacobian. You will construct these structures using three functions. First, the function `create_mesh_data` will initialize the `mesh_data` structure array with relevant isoparametric quantities from the element geometry `geom` and description of the mesh `xcg`, `e2vcg`, `e2bnd`. Then, the PDE-specific function `create_elem_update_mesh_data_pde0` will be used to create the `elem` structure (see `notation.m`) and update the `mesh_data` structure array with the derivative of the solution basis $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{x}}$. Notice that that `create_elem_update_mesh_data_pde0` will call the function you just wrote. Finally, the function

`update_mesh_data_pars` will be used to evaluate spatially dependent parameters (such as $\rho(\boldsymbol{x})$, $\boldsymbol{\nu}(\boldsymbol{x})$, and $\bar{\boldsymbol{q}}(\boldsymbol{x})$) at each quadrature node of each element and update `mesh_data` accordingly. See the comments of the function for a complete description. The following snippet of code would create `elem` and `mesh_data` given the element geometry `geom` and mesh `xcg`, `e2vcg`, `e2bnd`

```
mass_par_fcn = @(x) 1;
eqn_pars_fcn = @(x) [x(1)^2*x(2); exp(-x(1)^2)*cos(x(2)); 1; 2];
nbc_val_fcn = @(x, bnd) sin(x(1))*(bnd==1) + (-1)*(bnd==2);

mesh_data = create_mesh_data(geom, xcg, e2vcg, e2bnd);
[elem, mesh_data] = create_elem_update_mesh_data_pde0(geom, mesh_data);
mesh_data = update_mesh_data_pars(mesh_data, mass_par_fcn, eqn_pars_fcn, nbc_val_fcn);
```

where, in this ficticious example,

$$\rho(\boldsymbol{x}) = 1, \qquad \boldsymbol{\nu}(\boldsymbol{x}) = \begin{bmatrix} x_1^2 x_2 \\ e^{-x_1^2}\cos(x_2) \\ 1 \\ 2 \end{bmatrix}, \qquad \bar{\boldsymbol{q}}(\boldsymbol{x}) = \begin{cases} \sin(x_1) & \text{if } \boldsymbol{x} \in \partial\Omega_1 \\ -1 & \text{if } \boldsymbol{x} \in \partial\Omega_2 \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

- Implement a function that evaluates the discrete element residual $\hat{\boldsymbol{R}}^e(\hat{\boldsymbol{U}}^e)$ and its Jacobian $\frac{\partial \hat{\boldsymbol{R}}^e}{\partial \hat{\boldsymbol{U}}^e}(\hat{\boldsymbol{U}}^e)$ given the solution coefficients for a single element $\hat{\boldsymbol{U}}^e$ and information about the element (geometry, basis functions, quadrature rules). Your functions should have the following signature:

```
function [Re, dRe] = eval_elem_resjac_isoparam(Ue, elem, mesh_data)
%EVAL_ELEM_RESJAC_ISOPARAM Evaluate element residual and Jacobian for
%isoparametric elements.
%
% Input arguments
% ---------------
%   UE : Array (NDOF_PER_ELEM,) : Element solution (primary variables)
%
%   ELEM, MESH_DATA : See notation.m
%
% Output arguments
% ----------------
%   RE : Array (NDOF_PER_ELEM,) : Element residual
%
%   DRE : Array (NDOF_PER_ELEM, NDOF_PER_ELEM) : Element Jacobian
```

The `elem` structure contains a field `eval_pntws_contrib` that evaluates the the flux function, source term, and their derivatives, e.g., `eval_pntws_contrib_pde0` (see `notation.m`).

- In the same spirit, implement a function that evaluates the element mass matrix $\boldsymbol{M}^e$. Your function should have the following signature:

```
function [Me] = eval_unassembled_mass(elem, mesh_data)
%EVAL_UNASSEMBLED_MASS Evaluate/store element mass matrices for each
%element.
%
% Input arguments
% ---------------
%   ELEM, MESH_DATA : See notation.m
%
% Output arguments
% ----------------
%   ME : Array (NDOF_PER_ELEM, NDOF_PER_ELEM, NELEM): Element matrix for
%     all elements in mesh
```

Recall that the element mass matrix can be used to integrate the inner product of the solution with itself over the element. This can be used to check the implementation of the element mass matrix (construct a simple case).

- Re-structure the finite element code you wrote during your homework to be able to handle both linear and nonlinear problems. You are given the functions `assemble_nobc_vec` and `assemble_nobc_mat` that perform the assembly operation for vector and matrix quantities, respectively.
  - Implement a function that evaluates the element residual and Jacobian for each element in the domain and stores them in an unassembled (element-wise) format. Your function should have the following signature:

```
function  [Re, dRe] = eval_unassembled_resjac(U, elem, mesh_data, ldof2gdof)
%EVAL_UNASSEMBLED_RESJAC Evaluate/store element residual vector and
%Jacobian matrices for each element.
%
% Input arguments
% ---------------
%   U : Array (NDOF,) : Solution vector
%
%   ELEM, MESH_DATA, LDOF2GDOF : See notation.m
%
% Output arguments
% ----------------
%   RE : Array (NDOF_PER_ELEM, NELEM): Element residual vector for
%     all elements in mesh
%
%   DRE : Array (NDOF_PER_ELEM, NDOF_PER_ELEM, NELEM): Element Jacobian
%     matrix for all elements in mesh
```

  - Next, we need to generalize the application of our boundary conditions via static condensation and, most importantly, decouple it from the solve. Implement a function that restricts a vector defined over all global degrees of freedom to only its free entries (degrees of freedom without an essential boundary condition). Your function should have the following signature:

```
function  [vr] = rstr_to_free_dof_vec(v, dbc_idx)
%RSTR_TO_FREE_DOF_VEC Restrict vector defined over all global degrees of
%freedom to only the free degrees of freedom (without prescribed primary
%variable), i.e., remove all entries corresponding to indices in DBC_IDX.
%
%Input arguments
%---------------
%   V : Array (NDOF,) : Vector defined over all global degrees of freedom
%
%   DBC_IDX : See notation.m
%
%Output arguments
%----------------
%   VR : Array (NDOF-NDBC,) : V restricted to free degrees of freedom
```

    Implement a separate function that restricts the rows/columns of a matrix defined over all global degrees of freedom to only its free entries (degrees of freedom without an essential boundary condition). Your function should have the following signature:

```
function  [Mr] = rstr_to_free_dof_mat(M, dbc_idx)
%RSTR_TO_FREE_DOF_MAT Restrict matrix defined over all global degrees of
%freedom to only the free degrees of freedom (without prescribed primary
%variable), i.e., remove all rows/cols corresponding to indices in DBC_IDX.
%
%Input arguments
%---------------
```

```
%   M : Array (NDOF, NDOF) : Matrix defined over all global degrees of
%     freedom
%
%   DBC_IDX : See notation.m
%
%Output arguments
%------------------
%   MR : Array (NDOF-NDBC, NDOF-NDBC) : M with rows/columns restricted to
%     free degrees of freedom
```

– Implement a function that evaluates the global (assembled) finite element residual and Jacobian. Your function should have the following signature:

```
function [Rf, dRf] = create_fem_resjac(Uf, elem, mesh_data, dbc_idx, ...
                                       dbc_val, ldof2gdof, cooidx, lmat2gmat)
%CREATE_FEM_RESJAC Create the finite element residual and Jacobian,
%restricted to the free degrees of freedom. When combined with a nonlinear
%solver, this will approximate the solution of a PDE (described in ELEM,
%MESH_DATA) on the mesh (defined by ELEM, MESH_DATA).
%
% Input arguments
% ---------------
%   ELEM, MESH_DATA, DBC_IDX, DBC_VAL : See notation.m
%
%   LDOF2GDOF, COOIDX, LMAT2GMAT : See notation.m
%
% Output arguments
% ----------------
%   RF : Array (NDOF-NDBC,) : Finite element residual, restricted to free
%     degrees of freedom
%
%   DRF : Sparse matrix (NDOF-NDBC, NDOF-NDBC) : Finite element Jacobian
%     restricted to free degrees of freedom
```

– Implement a function that solves a static finite element problem using a Newton-Raphon nonlinear solver (you wrote during Homework 5). Your function should have the following signature:

```
function [U, info] = solve_fem_static(elem, mesh_data, dbc_idx, dbc_val, ...
                                      ldof2gdof, cooidx, lmat2gmat, Uf0, ...
                                      tol, maxit)
%SOLVE_FEM_STATIC Solve nonlinear finite element problem given complete
%description of mesh, PDE, and boundary conditions.
%
% Input arguments
% ---------------
%   ELEM, MESH_DATA, DBC_IDX, DBC_VAL: See notation.m
%
%   LDOF2GDOF, COOIDX, LMAT2GMAT : See notation.m
%
%   UF0 : Array (NDOF-NDBC,) : Initial guess for vector of primary
%     variables at free indices (where essential boundary condition not
%     given).
%
%   TOL, MAXIT : See SOLVE_NEWTRAPH
%
% Output arguments
% ----------------
%   U : Array (NDOF,) : Approximate solution of PDE (primary variables) for
%     all global degrees of freedom.
%
%   INFO : See SOLVE_NEWTRAPH
```

- Test your finite element code with PDE0 and verify optimal convergence rates. That is, for a sequence of meshes with $2^k$ elements for $k = 0, 1, 2, 3$ and polynomial orders $p = 1, \ldots, 5$, verify that your finite

element solution is approaching the exact solution with the optimal convergence rate $\mathcal{O}(h^{p+1})$, where $h$ is the finite element size. This can be done by plotting the finite element error versus the element size on a log-log plot. Once the mesh is sufficiently fine, this should be a straight line. The slope of this line is the convergence rate.

The exact solution of PDE0 is

$$u(x) = \frac{2\cos(1-x) - \sin(x)}{\cos(1)} + x^2 - 2$$

and the $L^2$ finite element error is

$$e = \sqrt{(\hat{\boldsymbol{U}} - \boldsymbol{U}^*)^T \boldsymbol{M} (\hat{\boldsymbol{U}} - \boldsymbol{U}^*)},$$

where $\hat{\boldsymbol{U}} \in \mathbb{R}^{N_d}$ is the global (assembled) finite element solution and $\boldsymbol{U}^* \in \mathbb{R}^{N_d}$ is the exact solution interpolated on the finite element mesh. The function `solve_pde0` is provided for your convenience; however, it will not run to completion until all the coding tasks in this section are complete.

**Part 3:** (50 points) In this part of the project, we will extend our finite element code to solve the Poisson equation

$$\begin{aligned} (-k_{ij} u_{,j})_{,i} &= f \quad \text{in } \Omega \\ (-k_{ij} u_{,j}) n_i &= \bar{q} \quad \text{on } \partial\Omega \end{aligned} \tag{24}$$

where $\Omega \subset \mathbb{R}^d$ and for each $\boldsymbol{x} \in \mathbb{R}^d$, the coefficient matrix $\boldsymbol{k}(\boldsymbol{x}) \in \mathbb{R}^{d \times d}$, source term $f(\boldsymbol{x}) \in \mathbb{R}$, outward normal $\boldsymbol{n}(\boldsymbol{x})$ to $\partial\Omega$, and natural boundary condition $\bar{q}(\boldsymbol{x}) \in \mathbb{R}$.

Similar to PDE0, we let $\hat{u}_i^e$ denote the solution at the $i$th node of element $e$ (element node numbering) for $i = 1, \ldots, N_v^e$ and let $\hat{\boldsymbol{U}}^e = (\hat{u}_1^e, \ldots, \hat{u}_{N_v^e}^e)^T$ be the local (element) solution vector. Similarly, let $\hat{u}_i^{ef}$ denote the solution at the $i$th node of face $f$ of element $e$ (face node numbering) for $i = 1, \ldots, N_v^f$ and let $\hat{\boldsymbol{U}}^{ef} = (\hat{u}_1^{ef}, \ldots, \hat{u}_{N_v^f}^{ef})$. The quantities are related by $\hat{u}_j^{ef} = \hat{u}_{\Theta_{jf}}^e$ for $j = 1, \ldots, N_v^f$, which implies $\boldsymbol{\Lambda} = \boldsymbol{\Theta}$. For this problem, we approximate the element-wise solution as

$$u(\boldsymbol{x})|_{\Omega_e} = \sum_{i=1}^{N_v^e} \hat{u}_i^e \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x})), \qquad u(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{i=1}^{N_v^f} \hat{u}_i^e \pi_i(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})). \tag{25}$$

where $\psi_i(\boldsymbol{\xi})$ and $\pi_i(\boldsymbol{r})$ are the basis functions defined over a particular geometric element, which implies the solution basis functions are identical to the basis functions of the geometric element (22). Finally, for the Poisson problem we take the parameter vector to be $\boldsymbol{\nu}(\boldsymbol{x}) = (k_{11}(\boldsymbol{x}), k_{21}(\boldsymbol{x}), \ldots, k_{dd}(\boldsymbol{x}), f(\boldsymbol{x}))^T$.

**Tasks for Part 3**

Your tasks for this part of the project are to convert the Poisson equation into the form of the general, second-order PDE (10), implement it in the FEM code from Part 2, and solve a number of problems. The function `solve_poi` is provided for your convenience; it will be useful for a number of the tasks in this section.

- Identify the PDE variables, its gradient, the flux function and source term, and natural boundary condition for the Poisson equation based on the general, second-order PDE in (10).
- Derive the partial derivatives of the flux function and source term.
- Implement a function with the following signature that returns the flux function, source term, and their partial derivatives given the primary variables, its gradient, and the parameters:

```
function [F, dFdu, dFdq, S, dSdu, dSdq] = eval_pntws_contrib_poi(u, q, eqn_pars)
%EVAL_PNTWS_CONTRIB_POI Evaluate the flux function and source term (and
%their derivatives) that define the Poisson equation.
```

```
%
% Input arguments
% ———————————————
%   U : Array (NC,) : Primary variables
%
%   Q : Array (NC, NDIM) : Gradient of primary variables
%
%   EQN_PARS : Array (M,) : Parameters to flux function and source term
%
% Output arguments
% ————————————————
%   F, DFDU, DFDQ, S, DSDU, DSDQ : See notation.m
```

- Implement a function with the following signature that creates the solution bases $\boldsymbol{\Psi}(\boldsymbol{\xi})$, $\boldsymbol{\Pi}(\boldsymbol{r})$ and the derivative $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}$, evaluated at the appropriate quadrature points, and the face-to-element degree of freedom mapping $\boldsymbol{\Lambda}$:

```
function [T, dTdz, Tf, ldof2fdof] = create_soln_basis_ldof2fdof_poi(Q, dQdz, Qf, f2v)
%CREATE_SOLN_BASIS_LDOF2FDOF_POI Create solution basis (element and face)
%and its derivative w.r.t. reference coordinates and the face-to-element
%degree of freedom mapping for the Poisson equation.
%
% Input arguments
% ———————————————
%   Q, DQDZ, QF, F2V : See notation.m
%
% Output arguments
% ————————————————
%   T, DTDZ, TF, LDOF2FDOF : See notation.m
```

- Consider the Poisson problem on the unit disk

$$-\Delta u = 1 \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega, \tag{26}$$

where $\Omega \subset \mathbb{R}^2$ is the unit disk. The exact solution is

$$u(x, y) = \frac{1 - x^2 - y^2}{4}.$$

  - Use your FEM code to solve the above Poisson problem. Plot the solution $u(\boldsymbol{x})$ over the domain $\Omega$ and along any line that passes through the center of the disk. Use a mesh consisting of $20 \times 20$ hypercube elements of order $p = 2$.
  - Complete a convergence study for both simplex and hypercube meshes for polynomial orders $p = 1, 2, 3$. Be sure to plot the element size $h$ versus the error in your finite element solution. Assume the elements are uniformly sized, i.e., $h = \sqrt{V(\Omega)/N_e}$. Discuss similarities and differences between the convergence rates and absolute error for a given element size between the simplex and hypercube meshes.

- Use your FEM code to solve the Poisson problem (24) on the Batman domain (Figure 7) with boundary conditions: natural boundary condition with $\bar{q}(\boldsymbol{x}) = -10 \sin(x_1)$ on $\partial\Omega_1 \cup \partial\Omega_2$ and an essential boundary condition $u = 0$ on $\partial\Omega_3$. Take the coefficient matrix to be $k_{11} = 1$, $k_{22} = 10$, and $k_{12} = k_{21} = 0$. In the notation of (24), this corresponds to

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}, \qquad f(\boldsymbol{x}) = 0, \qquad \bar{q}(\boldsymbol{x}) = \begin{cases} -10 \sin(x_1) & \boldsymbol{x} \in \partial\Omega_1 \cup \partial\Omega_2 \\ 0 & \text{otherwise,} \end{cases} \tag{27}$$

and $u(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \partial\Omega_3$. Use the $p = 2$ simplicial mesh provided (`_mesh/_meshes/batman-simp-p2.mat`). Plot the solution over the entire domain $\Omega$ and along the line $\Gamma$ (Figure 7).

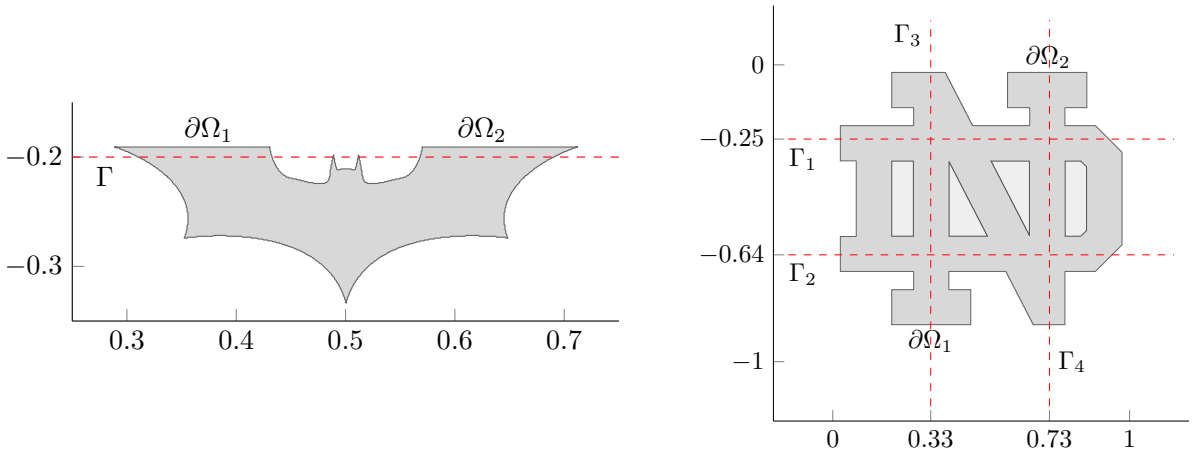


Figure 7: Domain ($\Omega$), boundaries ($\partial\Omega_i$), and lines along which to evaluate quantities ($\Gamma_i$) for Batman symbol and Notre Dame logo. For each domain, the first two boundaries are shown above and the third boundary is $\partial\Omega_3 = \partial\Omega \backslash (\partial\Omega_1 \cup \partial\Omega_2)$.

- Use your FEM code to solve the Poisson problem (24) on the ND logo domain (Figure 7) with boundary conditions: prescribed solution $u = 0$ on $\partial\Omega_1$, prescribed solution $u = 10$ on $\partial\Omega_2$, and homogeneous natural boundary conditions $\bar{q} = 0$ on $\partial\Omega_3$. Take the coefficient matrix to be $\boldsymbol{k} = \boldsymbol{I}_2$ ($2 \times 2$ identity matrix). In the notation of (24), this corresponds to

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad f(\boldsymbol{x}) = 0, \qquad \bar{q}(\boldsymbol{x}) = 0, \qquad \boldsymbol{u}(\boldsymbol{x}) = \begin{cases} 0 & \boldsymbol{x} \in \partial\Omega_1 \\ 10 & \boldsymbol{x} \in \partial\Omega_2. \end{cases} \tag{28}$$

  Use the $p = 2$ simplicial mesh provided (`_mesh/_meshes/nd-simp-p2.mat`). Plot the solution over the entire domain $\Omega$ and along the lines $\Gamma_1$, $\Gamma_2$, $\Gamma_3$, $\Gamma_4$ (Figure 7).

- Use your FEM code to solve the Poisson problem in (24) defined over the unit cube ($\Omega = [0,1]^3$) with coefficient matrix

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

  and boundary conditions

$$\bar{q}(\boldsymbol{x}) = \begin{cases} x_1 + x_2 + x_3 & \boldsymbol{x} \in \partial\Omega_1 \cup \partial\Omega_2 \cup \partial\Omega_4 \cup \partial\Omega_5 \\ 0 & \text{otherwise}, \end{cases} \qquad u(\boldsymbol{x}) = \begin{cases} 0 & \boldsymbol{x} \in \partial\Omega_3 \\ \sin(2\pi x_1)\cos(2\pi x_2) & \boldsymbol{x} \in \partial\Omega_6 \end{cases}.$$

  Boundaries $\partial\Omega_1$ and $\partial\Omega_4$ are defined as the boundaries with unit normals $\boldsymbol{e}_1$, $-\boldsymbol{e}_1$, respectively. Boundaries $\partial\Omega_2$ and $\partial\Omega_5$ are defined as the $\boldsymbol{e}_2$, $-\boldsymbol{e}_2$, respectively. Boundaries $\partial\Omega_3$ and $\partial\Omega_6$ are defined as the $\boldsymbol{e}_3$, $-\boldsymbol{e}_3$, respectively. Plot the solution over the surface over the domain $\partial\Omega$ and along the plane $\Gamma = \{(x, y, z) \mid 0 \leqslant x \leqslant 1, 0 \leqslant y \leqslant 1, z = 0.5\}$.

**Part 4:** (50 points) In this part of the project, we will extend our finite element code to solve the linear elasticity equations

$$\begin{aligned} \sigma_{ji,j} + f_i &= 0 \quad \text{in } \Omega, \\ \sigma_{ij} n_j &= \bar{t}_i \quad \text{on } \partial\Omega, \end{aligned} \tag{29}$$

for $i = 1, \ldots, d$, where the stress tensor $\boldsymbol{\sigma} \in \mathbb{R}^{d \times d}$ and strain tensor $\boldsymbol{\epsilon} \in \mathbb{R}^{d \times d}$ are defined as

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}, \qquad \epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}),$$

and $u_i \in \mathbb{R}$ is the displacement in the $i$th direction for $i = 1, \ldots, d$. We will solely consider a homogeneous, isotropic material with $C_{ijkl}(\boldsymbol{x}) = \lambda(\boldsymbol{x})\delta_{ij}\delta_{kl} + \mu(\boldsymbol{x})(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$, where $\lambda(\boldsymbol{x})$ and $\mu(\boldsymbol{x})$ are the Lamé parameters. In the notation of (10), the solution vector for this problem is $\boldsymbol{u} = (u_1, \ldots, u_d)^T$.

Let $\hat{u}_{ij}$ denote the displacement in the $i$th direction at node $j$ of the mesh (global node numbering) for $i = 1, \ldots, d$, $j = 1, \ldots, N_v$ and $\hat{\boldsymbol{U}} = (u_{11}, \ldots, u_{d1}, \ldots, u_{dN_v})$ be the global solution vector. In addition, let $\hat{u}^e_{ij}$ denote the displacement in the $i$th direction at node $j$ of element $e$ (element node numbering) for $i = 1, \ldots, d$, $j = 1, \ldots, N^e_v$ and let $\hat{\boldsymbol{U}}^e = (\hat{u}_{11}, \ldots, \hat{u}_{d1}, \ldots, \hat{u}_{dN_v})^T$ be the local (element) solution vector. The global and local solution vectors are related by the `ldof2gdof` matrix discussed in class. Similarly, let $\hat{u}^{ef}_{ij}$ denote the displacement in the $i$th direction at node $j$ of face $f$ of element $e$ (face node numbering) for $i = 1, \ldots, d$ and $j = 1, \ldots, N^f_v$ and let $\hat{\boldsymbol{U}}^{ef} = (\hat{u}^{ef}_1, \ldots, \hat{u}^{ef}_{N^f_v})^T$. The element and face quantities are related by $\hat{u}^{ef}_{ij} = \hat{u}^e_{i\Theta_{jf}}$ for $i = 1, \ldots, d$ and $j = 1, \ldots, N^f_v$, which implies $\Lambda_{kf} = i + d(\Theta_{jf} - 1)$, where $k = i + d(j-1)$ for $i = 1, \ldots, d$, $j = 1, \ldots, N^f_v$, and $f = 1, \ldots, N_f$. For this problem, we approximate the element-wise solution as

$$u_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{N^e_v} \hat{u}^e_{ij}\psi_j(\mathcal{G}_e^{-1}(\boldsymbol{x})), \qquad u_i(\boldsymbol{x})|_{\partial\Omega_{ef}} = \sum_{j=1}^{N^f_v} \hat{u}^{ef}_{ij}\pi_j(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})),$$

for $i = 1, \ldots, d$, where $\psi_j(\boldsymbol{\xi})$ and $\pi_j(\boldsymbol{r})$ are the basis functions defined over a particular geometric element, which implies

$$\boldsymbol{\Psi}(\boldsymbol{\xi}) = \begin{bmatrix} \psi_1(\boldsymbol{\xi})\boldsymbol{I}_d \\ \vdots \\ \psi_{N^e_v}(\boldsymbol{\xi})\boldsymbol{I}_d \end{bmatrix}, \qquad \boldsymbol{\Pi}(\boldsymbol{r}) = \begin{bmatrix} \pi_1(\boldsymbol{r})\boldsymbol{I}_d \\ \vdots \\ \pi_{N^f_v}(\boldsymbol{r})\boldsymbol{I}_d \end{bmatrix}, \tag{30}$$

where $\boldsymbol{I}_d \in \mathbb{R}^{d\times d}$ is the identity matrix. Finally, the linear elasticity parameter vector is taken to be $\boldsymbol{\nu}(\boldsymbol{x}) = (\lambda(\boldsymbol{x}), \mu(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_d(\boldsymbol{x}))^T$.

**Tasks for Part 4**

Your tasks for this part of the project are to convert the linear elasticity equations into the form of the general, second-order PDE (10), implement it in the FEM code from Part 2, and solve a number of elasticity problems. The function `solve_linelast` is provided for your convenience; it will be useful for a number of the tasks in this section.

- Identify the PDE variables, its gradient, the flux function and source term, and natural boundary condition for linear elasticity based on the general, second-order PDE in (10).
- Derive the partial derivatives of the flux function and source term.
- Implement a function with the following signature that returns the flux function, source term, and their partial derivatives given the primary variable, its gradient, and the PDE parameters:

```
function [F, dFdu, dFdq, S, dSdu, dSdq] = eval_pntws_contrib_linelast(u, q, eqn_pars)
%EVAL_PNTWS_CONTRIB_LINELAST Evaluate the flux function and source term
%(and their derivatives) that define the linear elasticity equation.
%
% Input arguments
% ---------------
%   U : Array (NC,) : Primary variables
%
%   Q : Array (NC, NDIM) : Gradient of primary variables
%
%   EQN_PARS : Array (M,) : Parameters to flux function and source term
%
% Output arguments
% ----------------
%   F, DFDU, DFDQ, S, DSDU, DSDQ : See notation.m
```

- Verify the face-to-element degree of freedom mapping $\mathbf{\Lambda}$ for the special case of the $p = 2$ simplex element in $\mathbb{R}^2$.
- Implement a function with the following signature that creates the solution bases $\mathbf{\Psi}(\boldsymbol{\xi})$, $\mathbf{\Pi}(\boldsymbol{r})$ and the derivative $\frac{\partial \mathbf{\Psi}}{\partial \boldsymbol{\xi}}$, evaluated at the appropriate quadrature points, and the face-to-element degree of freedom mapping $\mathbf{\Lambda}$:

```
function [T, dTdz, Tf, ldof2fdof] = create_soln_basis_ldof2fdof_linelast(Q,dQdz,Qf,f2v)
%CREATE_SOLN_BASIS_LDOF2FDOF_LINELAST Create solution basis (element and face)
%and its derivative w.r.t. reference coordinates and the face-to-element
%degree of freedom mapping for linear elasticity.
%
% Input arguments
% ---------------
%   Q, DQDZ, QF, F2V : See notation.m
%
% Output arguments
% ----------------
%   T, DTDZ, TF, LDOF2FDOF : See notation.m
```

- Consider a multimaterial beam (Figure 8) with boundary conditions: clamped on $\partial\Omega_1$ ($u_1 = u_2 = 0$), no traction on $\partial\Omega_2 \cup \partial\Omega_4$ ($\bar{t}_1 = \bar{t}_2 = 0$), and a distributed force in the $-y$ direction of 0.1 on $\partial\Omega_3$ ($\bar{t}_1 = 0, \bar{t}_2 = -0.1$). Take the Lamé parameters for material 1 to be $\lambda_1(\boldsymbol{x}) = 365$, $\mu_1(\boldsymbol{x}) = 188$ and those for material 2 to be $\lambda_1(\boldsymbol{x}) = 36.5$, $\mu_1(\boldsymbol{x}) = 18.8$.
  - Solve for the deformation of the beam using your FEM code and COMSOL on any sufficiently refined mesh.
  - Using the solution from both codes, evaluate the displacements $u_1$, $u_2$ along the line $\Gamma$ shown in Figure 8 and plot the von Mises stress on the deformed geometry.
  - Check that the solutions from both code qualitatively look similar (by examing the stress field and deformed domain) and match quantitatively (by comparing the pointwise evaluations).

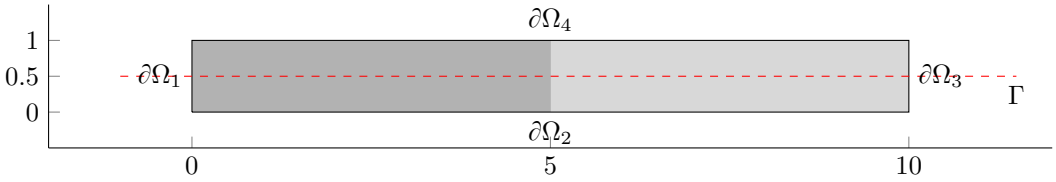


Figure 8: Multimaterial beam ($\Omega$), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities ($\Gamma$).

- Consider a hollow cylinder (Figure 9) with boundary conditions: clamped on $\partial\Omega_2$ ($u_1 = u_2 = u_3 = 0$), no traction on $\partial\Omega_3$ ($\bar{t}_1 = \bar{t}_2 = \bar{t}_3 = 0$), a distributed force in the $-z$ direction of 0.25 on $\partial\Omega_4$ ($\bar{t}_1 = \bar{t}_2 = 0$, $\bar{t}_3 = -0.25$), and a pressure load of 1 on $\partial\Omega_1$ ($\bar{\boldsymbol{t}} = -\boldsymbol{n}$, where $\boldsymbol{n}$ is the outward unit normal). Take the Lamé parameters to be $\lambda(\boldsymbol{x}) = 0.73$, $\mu(\boldsymbol{x}) = 0.376$.
  - Solve for the deformation of the hollow cylinder using your FEM code. Use $p = 2$ hypercube elements. Make sure your mesh is fine enough that your solution is converged.
  - Plot the von Mises stress on the surface of the deformed cylinder.
  - Plot the displacements $u_1$, $u_2$, $u_3$ along the line $\Gamma$ shown in Figure 9.

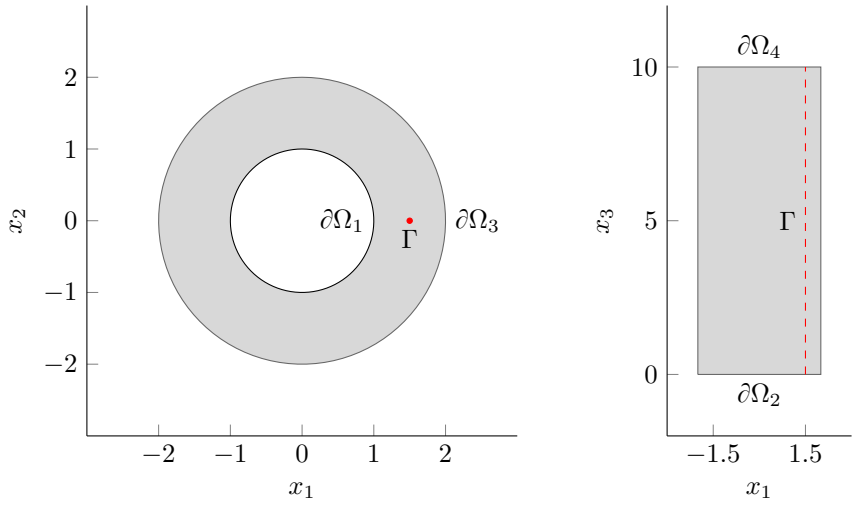


Figure 9: Hollow cylinder domain ($\Omega$), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities ($\Gamma$); $x_1 - x_2$ view (*left*) and $x_1 - x_3$ view (*right*). The boundaries $\partial\Omega_1$, $\partial\Omega_3$ are the inner, outer cylindrical surfaces, respectively. The boundaries $\partial\Omega_2$, $\partial\Omega_4$ are the bottom, top "caps" of the cylinder, respectively.

**Part 5:** (50 points) In this part of the project, we will extend our finite element code to solve the incompressible Navier-Stokes equations (viscous fluid flow)

$$\begin{aligned} -(\rho\nu v_{i,j})_j + \rho v_j v_{i,j} + P_{,i} = 0, \quad i = 1, \dots, d \\ v_{j,j} = 0 \end{aligned} \tag{31}$$

for all $\boldsymbol{x} \in \Omega$ with the boundary conditions $\rho\nu(v_{i,j}n_j - Pn_i) = \rho\bar{t}_i$ on $\partial\Omega$, where $\boldsymbol{v}(\boldsymbol{x}) \in \mathbb{R}^d$ is the velocity vector, $P(\boldsymbol{x}) \in \mathbb{R}$ is the pressure, $\rho(\boldsymbol{x}) \in \mathbb{R}$ is the density of the fluid, $\nu(\boldsymbol{x}) \in \mathbb{R}$ is the kinematic viscosity of the fluid, $\boldsymbol{n}(\boldsymbol{x}) \in \mathbb{R}^d$ is the outward normal to $\partial\Omega$, and $\bar{\boldsymbol{t}}(\boldsymbol{x}) \in \mathbb{R}^d$ is the traction boundary condition.

An important non-dimensional quantity in the study of fluid flow is the Reynolds number

$$Re = \frac{UL}{\nu}, \tag{32}$$

where $U$ is the velocity of the fluid with respect to an object, $L$ is the characteristic linear dimension, and $\nu$ is the kinematic viscosity of the fluid. The Reynolds number is the ratio of inertial-to-viscous forces and is used to predict flow patterns in different fluid flow situations.

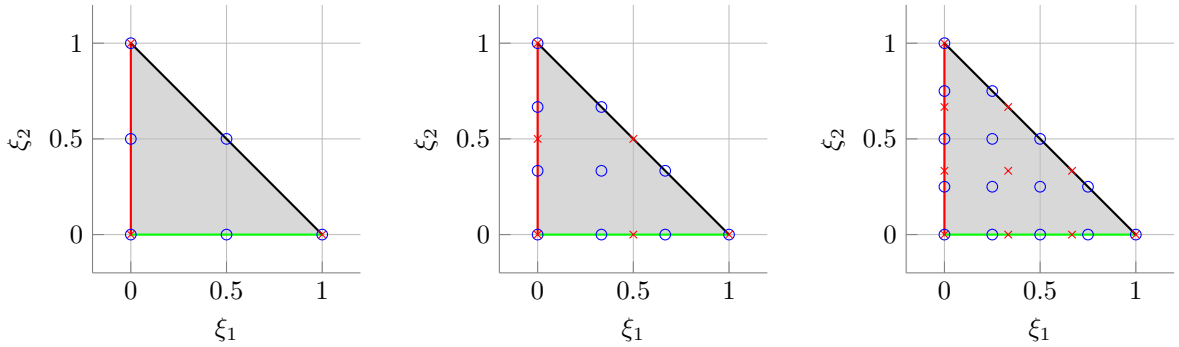


Figure 10: Mixed (velocity-pressure) triangular finite element in reference domain ($\xi_1$-$\xi_2$ space): P2-P1 (quadratic approximation of velocity, linear approximation of pressure) (*left*), P3-P2 (cubic approximation of velocity, quadratic approximation of pressure) (*center*), P4-P3 (quartic approximation of velocity, cubic approximation of pressure) (*right*). The faces are numbered as: face 1 (——), face 2 (——), and face 3 (——). There is a velocity degree of freedom $\boldsymbol{v}_i^e$ ($i = 1, \dots, N_v^{e,\boldsymbol{v}}$) at each (○) and a pressure degree of freedom $P_i^e$ ($i = 1, \dots, N_v^{e,P}$) at each (×). For both sets of nodes, the numbering is inherited from the standard numbering in Figure 1.

For stability reasons, we need to consider a mixed element approximation of the solution vector $\boldsymbol{u}(\boldsymbol{x}) = (v_1(\boldsymbol{x}), \ldots, v_d(\boldsymbol{x}), P)^T$. In particular, we will use an element of order $p$ to approximate the velocity ($\boldsymbol{v}(\boldsymbol{x})$) and an element of order $p-1$ to approximate the pressure field ($P(\boldsymbol{x})$) (Figure 10). Let $\{\psi_i^{\boldsymbol{v}}(\boldsymbol{\xi})\}_{i=1}^{N_v^{e,\boldsymbol{v}}}$ and $\{\pi_i^{\boldsymbol{v}}(\boldsymbol{r})\}_{i=1}^{N_v^{f,\boldsymbol{v}}}$ denote the basis functions used to approximate the velocity $\boldsymbol{v}(\boldsymbol{x})$ within an element, i.e., $p$th order Lagrangian polynomials, where $N_v^{e,\boldsymbol{v}}$ is the number of nodes associated with the velocity element and $N_v^{f,\boldsymbol{v}}$ is the number of nodes associated with each face of the velocity element. Similarly let $\{\psi_i^{P}(\boldsymbol{\xi})\}_{i=1}^{N_v^{e,P}}$ and $\{\pi_i^{P}(\boldsymbol{r})\}_{i=1}^{N_v^{f,P}}$ denote the basis functions used to approximate the pressure $P(\boldsymbol{x})$ within an element, i.e., $(p-1)$th order Lagrangian polynomials, where $N_v^{e,P}$ is the number of nodes associated with the pressure element and $N_v^{f,P}$ is the number of nodes associated with each face of the pressure element. In addition, let $\Theta^{\boldsymbol{v}}$, $\Theta^{P}$ denote the face-to-element vertex mapping for the velocity and pressure elements, respectively. This mixed element representation of the solution implies that we must have two meshes, one for the velocity degrees of freedom and one for the pressure degrees of freedom. The elements of both meshes will exactly align, but the nodal positions will not. Let $N_v^{\boldsymbol{v}}$ denote the number of vertices in the velocity mesh and $N_v^{P}$ denote the number of vertices in the pressure mesh. This implies there are $d \cdot N_v^{\boldsymbol{v}}$ velocity degrees of freedom and $N_v^{P}$ pressure degrees of freedom.

Let $\hat{v}_{ij}$ denote the fluid velocity in the $i$th direction at node $j$ of the velocity mesh (global node numbering) for $i = 1, \ldots, d$, $j = 1, \ldots, N_v^{\boldsymbol{v}}$, let $\hat{P}_j$ denote the fluid pressure at node $j$ of the pressure mesh (global node numbering) for $j = 1, \ldots, N_v^{P}$, and $\hat{\boldsymbol{U}} = (\hat{v}_{11}, \ldots, \hat{v}_{d1}, \ldots, \hat{v}_{dN_v^{\boldsymbol{v}}}, \hat{P}_1, \ldots, \hat{P}_{N_v^{P}})$ be the global solution vector. In addition, let $\hat{v}_{ij}^e$ denote the fluid velocity in the $i$th direction at node $j$ of velocity element $e$ (element node numbering) for $i = 1, \ldots, d$, $j = 1, \ldots, N_v^{e,\boldsymbol{v}}$ and let $\hat{P}_j^e$ denote the pressure at the $j$th node of pressure element $e$ (element node numbering) for $j = 1, \ldots, N_v^{e,P}$. Define the local (element) solution vector as $\hat{\boldsymbol{U}}^e = (\hat{v}_{11}^e, \ldots, \hat{v}_{d1}^e, \ldots, \hat{v}_{dN_v^{e,\boldsymbol{v}}}^e, \hat{P}_1^e, \ldots, \hat{P}_{N_v^{e,P}}^e)^T$. The global and local solution vectors are related by the `ldof2gdof` matrix discussed in class. Similarly, let $\hat{v}_{ij}^{ef}$ denote the $i$th component of the fluid velocity at node $j$ of face $f$ of velocity element $e$ for $i = 1, \ldots, d$ and $j = 1, \ldots, N_v^{f,\boldsymbol{v}}$ and $\hat{P}_j^{ef}$ denote the pressure at node $j$ of face $f$ of pressure element $e$ for $j = 1, \ldots, N_v^{f,P}$. Define the local (element face) solution vector as $\hat{\boldsymbol{U}}^{ef} = (\hat{v}_{11}^{ef}, \ldots, \hat{v}_{d1}^{ef}, \ldots, \hat{v}_{dN_v^{f,\boldsymbol{v}}}^{ef}, \hat{P}_1^{ef}, \ldots, \hat{P}_{N_v^{f,P}}^{ef})^T$. The element and face quantities are related by $\hat{v}_{ij}^{ef} = \hat{v}_{i\Theta_{jf}^{\boldsymbol{v}}}^e$ for $i = 1, \ldots, d$, $j = 1, \ldots, N_v^{f,\boldsymbol{v}}$ and $\hat{P}_j^{ef} = \hat{P}_{\Theta_{jf}^{P}}^e$ for $j = 1, \ldots, N_v^{f,P}$. From this information, the face-to-element degree of freedom mapping $\boldsymbol{\Lambda}$ can be determined; however, we omit it and instead refer to the code in `create_soln_basis_ldof2fdof_ins`.

For this problem, we approximate the element-wise solution as

$$
\begin{aligned}
v_i(\boldsymbol{x})|_{\Omega_e} &= \sum_{j=1}^{N_v^{e,\boldsymbol{v}}} \hat{v}_{ij}^e \psi_j^{\boldsymbol{v}}(\mathcal{G}_e^{-1}(\boldsymbol{x})), & v_i(\boldsymbol{x})|_{\partial\Omega_{ef}} &= \sum_{j=1}^{N_v^{f,\boldsymbol{v}}} \hat{v}_{ij}^{ef} \pi_j^{\boldsymbol{v}}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x})) \\
P(\boldsymbol{x})|_{\Omega_e} &= \sum_{i=1}^{N_v^{e,P}} \hat{P}_i^e \psi_j^{P}(\mathcal{G}_e^{-1}(\boldsymbol{x})), & P(\boldsymbol{x})|_{\partial\Omega_{ef}} &= \sum_{i=1}^{N_v^{f,P}} \hat{P}_i^{ef} \pi_j^{P}(\mathcal{F}_{ef}^{-1}(\boldsymbol{x}))
\end{aligned}
$$

where $\psi_j(\boldsymbol{\xi})$ and $\pi_j(\boldsymbol{r})$ are the basis functions defined over a particular geometric element, which implies

$$
\boldsymbol{\Psi}(\boldsymbol{\xi}) = \begin{bmatrix} \psi_1^{\boldsymbol{v}}(\boldsymbol{\xi})\boldsymbol{I}_d & & \\ \vdots & & \\ \psi_{N_v^{e,\boldsymbol{v}}}^{\boldsymbol{v}}(\boldsymbol{\xi})\boldsymbol{I}_d & & \\ & \psi_1^{P}(\boldsymbol{\xi}) & \\ & \vdots & \\ & \psi_{N_v^{e,P}}^{P}(\boldsymbol{\xi}) & \end{bmatrix}, \qquad \boldsymbol{\Pi}(\boldsymbol{r}) = \begin{bmatrix} \pi_1^{\boldsymbol{v}}(\boldsymbol{r})\boldsymbol{I}_d & & \\ \vdots & & \\ \pi_{N_v^{f,\boldsymbol{v}}}^{\boldsymbol{v}}(\boldsymbol{r})\boldsymbol{I}_d & & \\ & \pi_1^{P}(\boldsymbol{r}) & \\ & \vdots & \\ & \pi_{N_v^{f,P}}^{P}(\boldsymbol{r}) & \end{bmatrix}. \tag{33}
$$

Finally, the incompressible Navier-Stokes parameter vector is taken to be $\boldsymbol{\nu}(\boldsymbol{x}) = (\rho(\boldsymbol{x}), \nu(\boldsymbol{x}))^T$.

**Tasks for Part 5**

Your tasks for this part of the project are to convert the incompressible Navier-Stokes equations into the

form of the general, second-order PDE (10), implement it in the FEM code from Part 2, and use your code to solve two fluid flow problems. The function `solve_ins` is provided for your convenience; it will be useful for a number of the tasks in this section.

- Identify the PDE variables, its gradient, the flux function and source term, and natural boundary condition for incompressible Navier-Stokes equations based on the general, second-order PDE in (10).
- Derive the partial derivatives of the flux function and source term.
- Implement a function with the following signature that returns the flux function, source term, and their partial derivatives given the primary variable, its gradient, and the PDE parameters:

```
function [F, dFdu, dFdq, S, dSdu, dSdq] = eval_pntws_contrib_ins(u, q, eqn_pars)
%EVAL_PNTWS_CONTRIB_INS Evaluate the flux function and source term
%(and their derivatives) that define the incompressible Navier-Stokes
%equations.
%
% Input arguments
% ---------------
%   U : Array (NC,) : Primary variables
%
%   Q : Array (NC, NDIM) : Gradient of primary variables
%
%   EQN_PARS : Array (M,) : Parameters to flux function and source term
%
% Output arguments
% ----------------
%   F, DFDU, DFDQ, S, DSDU, DSDQ : See notation.m
```

- Implement a function with the following signature that creates the solution bases $\boldsymbol{\Psi}(\boldsymbol{\xi})$, $\boldsymbol{\Pi}(\boldsymbol{r})$ and the derivative $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}$, evaluated at the appropriate quadrature points; the face-to-element degree of freedom mapping $\boldsymbol{\Lambda}$ is provided for you:

```
function [T, dTdz, Tf, ldof2fdof] = create_soln_basis_ldof2fdof_ins(Q, dQdz, Qf, f2v)
%CREATE_SOLN_BASIS_LDOF2FDOF_INS Create solution basis (element and face)
%and its derivative w.r.t. reference coordinates and the face-to-element
%degree of freedom mapping for the incompressible Navier-Stokes equations.
%
% Input arguments
% ---------------
%   Q, DQDZ, QF, F2V : See notation.m
%
% Output arguments
% ----------------
%   T, DTDZ, TF, LDOF2FDOF : See notation.m
```

- Use your code to solve the lid-driven cavity problem. The lid-driven cavity is defined on a square domain (Figure 11) with boundary conditions: stationary, no-slip walls ($v_1 = v_2 = 0$) on $\partial\Omega_1 \cup \partial\Omega_2 \cup \partial\Omega_3$ and a moving, no-slip wall ($v_1 = 1$, $v_2 = 0$) on $\partial\Omega_4$. Take the characteristic length scale to be $L = 1$ (length/height of domain), the characteristic velocity to be the velocity of the moving wall $U = 1$, and the density to be $\rho(\boldsymbol{x}) = 1$.
  - Solve for the flow velocity and pressure at $Re = 100$ using both your code and COMSOL on any sufficiently refined mesh.
    - Using the solution from both codes, plot the velocities $v_1$, $v_2$ along the line $\Gamma$ shown in Figure 11 and the magnitude of the velocity throughout the domain.
    - Check that the solutions from both codes qualitatively look similar (by examining the velocity magnitude plot) and match quantatively (by comparing the pointwise evaluations).

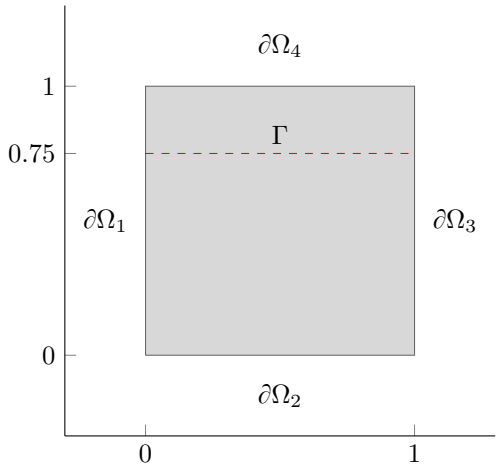


Figure 11: Lid-driven cavity domain ($\Omega$), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities ($\Gamma$).

  - Solve for flow velocity and pressure at $Re = 2000$ using your code. To solve for this Reynolds number you will need to use continuation, i.e., use the solution corresponding to a Reynolds number of $Re_k$ as the initial guess for the Newton solver for Reynolds number $Re_{k+1}$, where $k = 0, 1, \ldots$, $Re_k < Re_{k+1}$, and $Re_0$ is sufficiently small that the solution can be found easily from a zero initial guess ($Re_0 = 100$ is usually sufficient).
    * Plot the velocity magnitude and vorticity throughout the domain. On each plot superimpose a quiver plot that shows the direction of the flow.
    * Plot both components of the velocity along the the line $\Gamma$ defined in Figure 11.

- Use your code to solve for flow through the ND logo (Figure 7) with boundary conditions: vertical inflow on $\partial\Omega_1$ ($v_1 = 0$, $v_2 = 1$), traction-free outflow on $\partial\Omega_3$ ($\bar{t}_1 = \bar{t}_2 = 0$), and a no slip wall on $\partial\Omega_3$ ($v_1 = v_2 = 0$). Take the characteristic length scale to be $L = 1$, the characteristic velocity to be the inlet speed $U = 1$, and the density to be $\rho(\boldsymbol{x}) = 1$.
  - Solve for the flow velocity and pressure at $Re = 1300$; you will need to use continuation to solve for this Reynolds number. Use P2-P1 elements, i.e., triangular elements with quadratic ($p = 2$) basis functions for the velocity field and linear ($p = 1$) basis functions for the pressure field.
  - Plot the velocity magnitude and vorticity throughout the domain. On each plot superimpose a quiver plot that shows the direction of the flow.
  - Plot both components of the velocity along the four lines $\Gamma_1$, $\Gamma_2$, $\Gamma_3$, $\Gamma_4$ defined in Figure 7.