AME40541/60541: Finite Element Methods
Final Project

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.
- AME40541: You may assume the number of spatial dimensions (d in the document, `ndim` in the code) is $d = 1, 2$ and the polynomial degree (p in the document, `porder` in the code) is $p = 1, 2, 3$ in all tasks. That is, you only need your finite element code to work in one and two dimensions for polynomial orders one, two, and three to receive full credit.
- AME60541: Your code must work for $d = 1, 2, 3$ and $p = 1, 2, 3$.
- You may work in teams of up to four. Each team needs to submit their code to the instructor and TAs as well as a document that contains the answers to all questions and the required figures. Teams should not be mixed between the two classes.
- Timeline

Thursday, November 10, 2022	Part 1 due
Wednesday, November 23, 2022	Part 2-3 due
Wednesday, December 7, 2022	Part 4-7 due

- The final checkpoint will be accepted until December 18, 2022 at 11:59pm *without penalty*. It will not be accepted after that time.
- The point value for each part is indicated at the beginning of the section. In total, there are 180 points for AME40541 and 285 points for AME60541. This will count as 40% of your final grade.

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 (5 points) 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 polynomial space; $\mathcal{P}^p$ simplex elements 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

$$N_{\text{nd}}^{\text{el}} = \binom{p+d}{d}, \tag{1}$$

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

Since we will use mapped master elements to define the basis functions, we only need to consider the unit right simplex. We assume the nodes $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$ are distributed throughout the element and ordered (ascending)

first in the ξ_1-direction, then in the ξ_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 $N_{\text{fc}}^{\text{el}} = d+1$ 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 $\boldsymbol{\Pi} \in M_{N_{\text{nd}}^{\text{fc}}, N_{\text{fc}}^{\text{el}}}(\mathbb{N})$. where $N_{\text{nd}}^{\text{fc}}$ is the number of nodes on each element face, whose columns define the element nodes that lie on a particular face, i.e., Π_{if} is the element node number corresponding to the ith 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 vertex mapping and normal vectors are

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

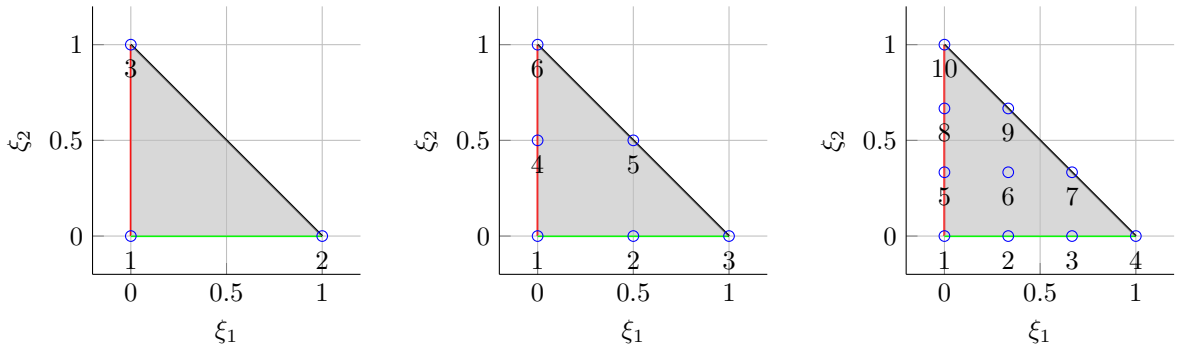

Figure 1: Triangular finite element in reference domain (ξ_1-ξ_2 space) with 3 nodes ($p = 1$) (*left*), 6 nodes ($p = 2$) (*center*), and 10 nodes ($p = 3$) (*right*). The faces are numbered as: face 1 (——), face 2 (——), and face 3 (——).

Tasks for Part 1.1

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 $\{\hat{\boldsymbol{\xi}}_k\}_{k=1}^{N_{\text{nd}}^{\text{el}}}$, the face-to-element vertex mapping $\boldsymbol{\Pi}$, and the unit normal for each face $\{\boldsymbol{N}_i\}_{i=1}^{N_{\text{fc}}^{\text{el}}}$. 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 element of order PORDER.
%
%Input arguments
%---------------
%   NDIM, PORDER : See notation.m
%
%Output arguments
%----------------
%   ZK, F2V, N : See notation.m
```

Read all comments and `notation.m` carefully for instructions regarding the inputs and outputs to the function.

2) 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 your 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 (15 points) Next, we need to define basis functions over the reference domain $\{\psi_i(\boldsymbol{\xi})\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$ for $\boldsymbol{\xi} \in \Omega_{\square}$, where $\Omega_{\square} \subset \mathbb{R}^d$ is the reference/parent element. For this we will use Vandermonde's approach (discussed in lecture and Ch. 6). 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 $N_{\text{nd}}^{\text{el}}$ basis functions as

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

where $\boldsymbol{\Upsilon} \in M_{d,N_{\text{nd}}^{\text{el}}}(\mathbb{N}_0)$ with entries Υ_{ij}, $i = 1, \ldots, d$, $j = 1, \ldots, N_{\text{nd}}^{\text{el}}$, such that $\sum_{i=1}^d \Upsilon_{ij} \leqslant p$ for each $j = 1, \ldots, N_{\text{nd}}^{\text{el}}$ that is used to sweep over all $N_{\text{nd}}^{\text{el}}$ 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 $\omega_i(\boldsymbol{\xi})$, $i = 1, \ldots, N_{\text{nd}}^{\text{el}}$

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

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

Denote the $N_{\text{nd}}^{\text{el}}$ nodes of the pth order simplex element as $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$, where $\hat{\boldsymbol{\xi}}_i = (\hat{\xi}_{1i}, \ldots, \hat{\xi}_{di})^T$. The nodal property is

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

for $i, j = 1, \ldots, N_{\text{nd}}^{\text{el}}$, which leads to

$$\sum_{k=1}^{N_{\text{nd}}^{\text{el}}} \alpha_{ik}\omega_k(\hat{\boldsymbol{\xi}}_j) = \delta_{ij}$$

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, pth order simplex evaluated at $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$, then the above constraints can

be written in matrix form as $\hat{\boldsymbol{V}}\boldsymbol{\alpha}^T = \boldsymbol{I}_{N_{\text{nd}}^{\text{el}}}$, where $\hat{\boldsymbol{V}}$, $\boldsymbol{\alpha}$ are the matrices with indices $\hat{V}_{ij}$, α_{ij}, respectively, and $\boldsymbol{I}_{N_{\text{nd}}^{\text{el}}}$ is the $N_{\text{nd}}^{\text{el}} \times N_{\text{nd}}^{\text{el}}$ 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 $\tilde{\boldsymbol{\xi}}_i = (\tilde{\xi}_{1i}, \dots, \tilde{\xi}_{di})$ to give

$$\psi_i(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_{\text{nd}}^{\text{el}}} \alpha_{ik}\omega_k(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_{\text{nd}}^{\text{el}}} \left(\hat{V}^{-1}\right)_{ki} \omega_k(\tilde{\boldsymbol{\xi}}_j) = \sum_{k=1}^{N_{\text{nd}}^{\text{el}}} \left(\hat{V}^{-1}\right)_{ki} \tilde{V}_{jk} \tag{3}$$

where the last expression used the d-dimensional, pth 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

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

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

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

$$\frac{\partial \psi_i}{\partial \xi_j}(\boldsymbol{\xi}) = \sum_{k=1}^{N_{\text{nd}}^{\text{el}}} \alpha_{ik} \frac{\partial \omega_k}{\partial \xi_j}(\boldsymbol{\xi}),$$

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

$$\frac{\partial \psi_i}{\partial \xi_j}(\tilde{\boldsymbol{\xi}}_k) = \sum_{l=1}^{N_{\text{nd}}^{\text{el}}} \alpha_{il} \frac{\partial \omega_l}{\partial \xi_j}(\tilde{\boldsymbol{\xi}}_k) = \sum_{l=1}^{N_{\text{nd}}^{\text{el}}} \left(\hat{V}^{-1}\right)_{li} \tilde{W}_{klj}, \quad \text{FINAL}$$

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

Tasks for Part 1.2

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

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

```
function [V, dV] = eval_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
```

2) 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] = 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, NDIM+1, NX) : Lagrange interpolation functions (NV) and
%     their derivative evaluated at each point in X. Q(i, 1, j) = value of
%     ith interpolation function evaluated at X(:, j), i.e., Q(i, 1, j) =
%     phi_i(X(j)). Q(i, 1+k, j) = kth partial derivative of ith
%     interpolation function evaluated at X(:, j), i.e., Q(i, 1+k, j) =
%     d(phi_i)/d(x_k))(X(:, j).
```

3) 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 $\boldsymbol{\xi} \in \Omega_\square$

$$\sum_{i=1}^{N_{\text{nd}}^{\text{el}}} \psi_i(\boldsymbol{\xi}) = 1, \qquad \sum_{i=1}^{N_{\text{nd}}^{\text{el}}} \frac{\partial \psi_i}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}) = \mathbf{0}.$$

4) 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 ϵ 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 jth canonical unit vector.

Part 1.3 (15 points) Next we will use the mapped master element 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

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

Reference element to pyysical element

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

$$\begin{aligned} \gamma_f : \Gamma_\square &\to \partial\Omega_{\square,f} \\ \boldsymbol{r} &\mapsto \boldsymbol{\xi} = \gamma_f(\boldsymbol{r}). \end{aligned}$$

Reference domain to reference element

be the bijection that maps $\Gamma_\square$ to the fth face of the master element, $\partial\Omega_{\square,f}$. Finally, for convenience, we introduce the mapping from the reference domain $\Gamma_\square$ to the the physical domain

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

Reference domain to physical element

See Figure 2 for these transformations and their relationships.

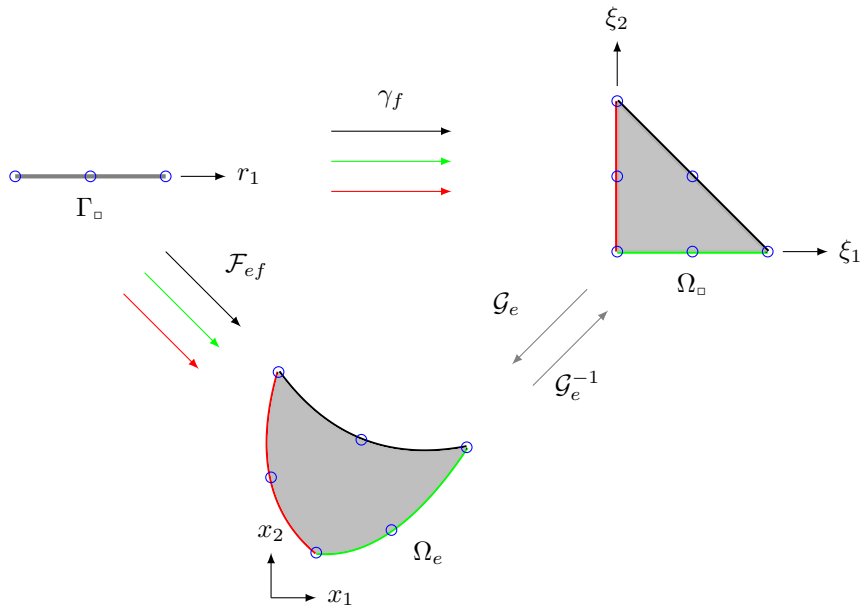

Figure 2: Mapping from $(d-1)$-dimensional reference simplex element ($\Gamma_\square$) to each face of the reference element ($\partial\Omega_{\square,f}$) ($\boldsymbol{\xi} = \gamma_f(\boldsymbol{r})$), the mapping from the d-dimensional reference simplex element ($\Omega_\square$) to the physical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

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 $\mathcal{G}_e$ for the volume integral and $\mathcal{F}_{ef}$ for the boundary integral, they can be transformed to the reference domain $\Omega_\square$ and $\Gamma_\square$, respectively,

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

where

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

where the derivative of $\mathcal{F}_{ef}$ can be expanded as

$$\frac{\partial \mathcal{F}_{ef}}{\partial \boldsymbol{r}}(\boldsymbol{r}) = \frac{\partial \mathcal{G}_e}{\partial \boldsymbol{\xi}}(\gamma_f(\boldsymbol{r}))\frac{\partial \gamma_f}{\partial \boldsymbol{r}}(\boldsymbol{r}) = \boldsymbol{G}_e(\gamma_f(\boldsymbol{r}))\frac{\partial \gamma_f}{\partial \boldsymbol{r}}(\boldsymbol{r}).$$

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

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

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

$$I_v \approx \sum_{k=1}^{N_{\text{qd}}^{\text{el}}} w_k \theta(\mathcal{G}_e(\tilde{\boldsymbol{\xi}}_k)) g_e(\tilde{\boldsymbol{\xi}}_k), \qquad I_s \approx \sum_{k=1}^{N_{\text{qd}}^{\text{fc}}} w_k^f \vartheta(\mathcal{F}_{ef}(\tilde{\boldsymbol{r}}_k))\sigma_{ef}(\tilde{\boldsymbol{r}}_k). \qquad \text{FINAL}$$

Next, we define basis functions over the physical element. Let $\{\psi_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$ define a basis over the d-dimensional reference element, where $\psi_i : \Omega_\square \to \mathbb{R}$ and $N_{\text{nd}}^{\text{el}}$ is the number of nodes in the element $\Omega_\square$. Furthermore, suppose the basis is Lagrangian with nodes $\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$, i.e., $\psi_i(\hat{\boldsymbol{\xi}}_j) = \delta_{ij}$. With these definitions, we define the basis functions over Ω_e as $\{\phi_i^e\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$ where $\phi_i^e : \Omega_e \to \mathbb{R}$ is defined as

$$\phi_i^e(\boldsymbol{x}) = \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x})). \tag{5}$$

From this definition and the integration formulas in (4), any integrals involving the basis ϕ_i^e can be written solely in terms of the reference domain basis $\psi_i(\boldsymbol{\xi})$ 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 Ω_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 $\Gamma_\square$

$$\mathcal{G}_e(\boldsymbol{\xi}) = \sum_{i=1}^{N_{\text{nd}}^{\text{el}}} \hat{\boldsymbol{x}}_i^e \psi_i(\boldsymbol{\xi}), \qquad \mathcal{F}_{ef}(\boldsymbol{r}) = \mathcal{G}_e(\gamma_f(\boldsymbol{r})), \tag{7}$$

where $\{\hat{\boldsymbol{x}}_i^e\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$ are the nodes associated with the element Ω_e and γ_f can be constructed analytically based on the geometry of the reference elements $\Gamma_\square$ and $\Omega_\square$.

Tasks for Part 1.3

Your task is to write a function that evaluates all relevant transformation quantities. Then you will use this functionality to approximate moments (integrals) over elements with regular and non-regular shapes.

1) Derive the expression for the quantities $\boldsymbol{G}_e(\boldsymbol{\xi})$, $\boldsymbol{F}_{ef}(\boldsymbol{r})$, for the transformation given in (7). Your answer should be in terms of the element coordinates $\{\hat{\boldsymbol{x}}_i^e\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$, the basis functions $\{\psi_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$, and the reference volume-to-face mapping $\gamma_f(\boldsymbol{r})$.

2) What is the reference volume-to-face mapping $\gamma_f(\boldsymbol{r})$, $f = 1, \ldots, 3$, for the three faces of the master triangle?

3) (AME60541) What is the reference volume-to-face mapping $\gamma_f(\boldsymbol{r})$, $f = 1, \ldots, 4$, for the four faces of the master tetrahedra?

4) Implement a function that evaluates all relevant transformation quantities. Your function should have the following signature:

```
function [xq, detG, Gi, xqf, sigf, Gif] = eval_transf_quant_ndim(xe, Qv, Qvf, r2z, f2v)
%EVAL_TRANSF_QUANT_DIM Evaluate transformation quantities for a single
%element given the nodal coordinates of the element in physical space (XE)
%and the basis functions (and their derivatives) of the element.
%
%Input arguments
%---------------
%  XE, QV, QVF, R2Z, F2V : See notation.m
%
%Output arguments
%----------------
%  XQ, DETG, GI, XQF, SIGF, GIF : See notation.m
```

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

5) 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 $d = 1, 2$ (AME40541) and $d = 1, 2, 3$ (AME60541).

- Use the function `create_qrule_gaussleg.m` to create quadrature rules for the master simplex and hypercube of any dimension d. A quadrature rule is a MATLAB structure `qrule` that contains quadrature weights/nodes for $\Omega_\square$ ($\{(w_k, \tilde{\boldsymbol{\xi}}_i)\}_{i=1}^{N_{\text{qd}}^{\text{el}}}$) and $\Gamma_\square$ ($\{(w_k^f, \tilde{\boldsymbol{r}}_i)\}_{i=1}^{N_{\text{qd}}^{\text{fc}}}$).
- Use the function `create_polysp_nodal.m` to create a local function space based on nodal simplex or hypercube elements of a given dimension d and polynomial degree p. A local function space is a MATLAB structure `lfcnsp` that contains: the nodal coordinates of $\Omega_\square$ ($\{\hat{\boldsymbol{\xi}}_i\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$) and $\Gamma_\square$ ($\{\hat{\boldsymbol{r}}_i\}_{i=1}^{N_{\text{nd}}^{\text{fc}}}$), the face-to-vertex mapping ($\boldsymbol{\Pi}$) and unit normals of each face ($\{\boldsymbol{N}_i\}_{i=1}^{d}$), the reference volume-to-face mapping $\gamma_f(\boldsymbol{r})$, and the the nodal basis functions evaluated at $\boldsymbol{\xi}$ ($\{\psi_i(\boldsymbol{\xi})\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$) and $\gamma_f(\boldsymbol{r})$ ($\{\psi_i(\gamma_f(\boldsymbol{r}))\}_{i=1}^{N_{\text{nd}}^{\text{el}}}$). Note: for simplex elements (`etype='simp'`), this relies on the code for Parts 1.1-1.2.

6) 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 the curved (quadratic) triangle in Figure 2 with nodes

$$\hat{\boldsymbol{x}}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \hat{\boldsymbol{x}}_2 = \begin{bmatrix} 0.5 \\ 0.15 \end{bmatrix}, \quad \hat{\boldsymbol{x}}_3 = \begin{bmatrix} 1.0 \\ 0.7 \end{bmatrix}, \quad \hat{\boldsymbol{x}}_4 = \begin{bmatrix} -0.3 \\ 0.5 \end{bmatrix}, \quad \hat{\boldsymbol{x}}_5 = \begin{bmatrix} 0.3 \\ 0.75 \end{bmatrix}, \quad \hat{\boldsymbol{x}}_6 = \begin{bmatrix} -0.25 \\ 1.2 \end{bmatrix}$$

using the transformation quantities from `transf_data` and the local function space structure `lfcnsp`. The moments of this element are $V = 0.7858$, $C = (0.1805, 0.4993)$, and $S = 4.0158$. Repeat for the quadratic quadrilateral in Figure 6.17 (Chapter 6 notes) (nodes are given in Equation (6.109)). The moments of this element are $V = 1.6533$, $C = (0.8632, 0.9387)$, and $S = 6.2791$. This will serve as a test for the functions you wrote in this section.

Part 1.4 (15 points) In this section, we will use a finite element mesh (Figure 3) to evaluate integrals over complex domains $\Omega \subset \mathbb{R}^d$:

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

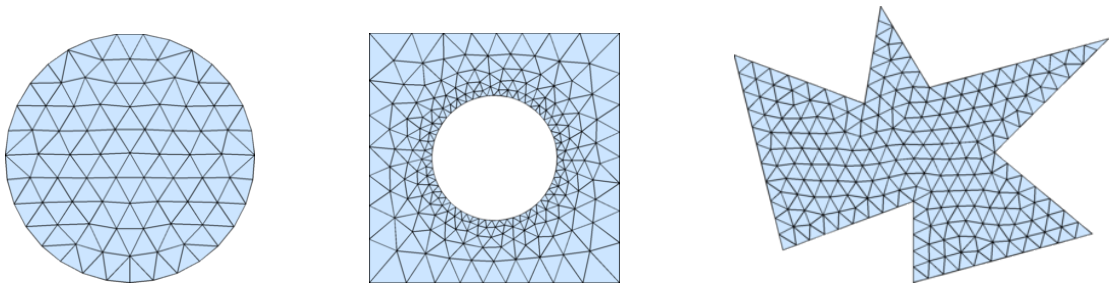

Figure 3: Mesh of a circle, square with circular hole, and arbitrary polygon using triangular elements.

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

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

where N_{el} 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 Ω, except in special cases.

Tasks for Part 1.4

With the transformation function and local function space structure defined in Part 1.3, you can create a

structure array `transf_data` that contains the transformation quantities for every element in a mesh defined by `xcg`, `e2vcg`, `e2bnd` using the function `create_transf_data_ndim` 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 Ω.

1) 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, sa] = compute_domain_metrics(transf_data, qrule)
%COMPUTE_DOMAIN_METRICS Compute the volume, centroid, and surface
%area of a domain (approximated) by the mesh described by TRANSF_DATA.
%
%Input arguments
%———————————————
%  TRANSF_DATA, QRULE : See notation.m
%
%Output arguments
%————————————————
%  V : number : Volume of domain
%
%  C : Array (NDIM,) : Centroid of domain
%
%  SA : number : Surface area of domain
```

2) 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_mesh_hcube` and create the corresponding `transf_data` structure array. Compute the volume, centroid, and surface area of Ω by integrating the appropriate quantities over the mesh and compare to the known volume, centroid, and surface area of a hypercube. Repeat the integral calculation for both simplex and hypercube elements. This will serve as another test for parts of your code (quadrature, transformation of integrals). Use `visualize_fem` to plot the mesh (1d, 2d). Use `visualize_fem3d` to plot in 3d (high-order visualization not supported).

3) 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 elements of a given polynomial degree by mapping a hypercube to a hypersphere. 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. Repeat the integral calculation for both simplex and hypercube elements.

4) 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 (`'batman0'` for the Batman domain, `'nd0'` for the Notre Dame domain), the element type (`'simp'` for simplex elements and `'hcube'` for hypercube elements), the refinement level (only 0 supported for now), 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.

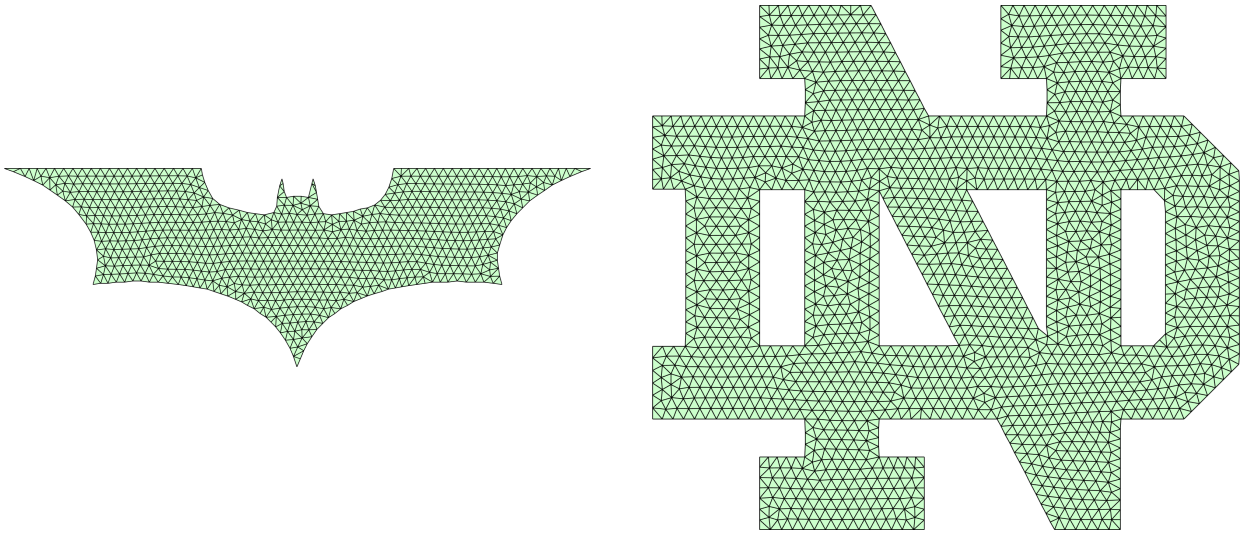

Figure 4: Simplicial mesh ($d = 2$) of the batman symbol and Notre Dame logo.

5) (AME60541) 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.

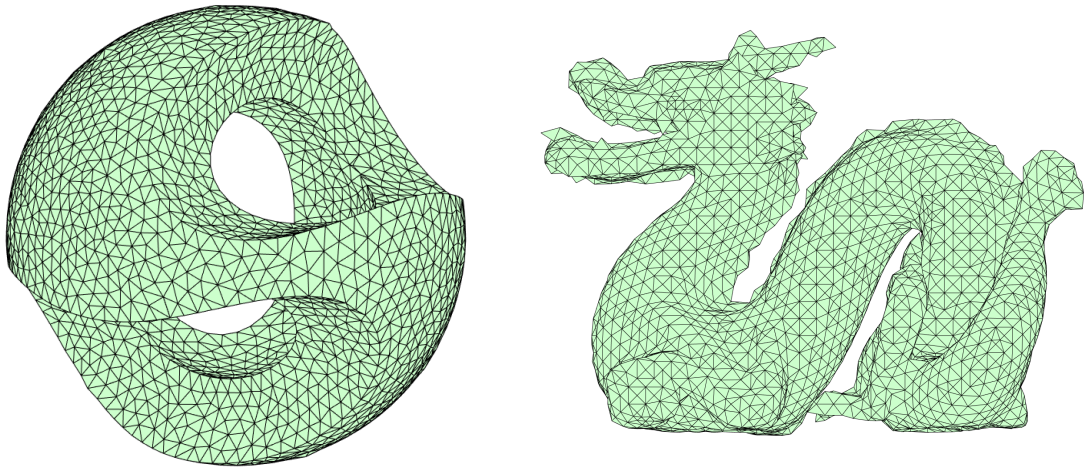

Figure 5: Simplicial mesh ($d = 3$) of a sculpture and dragon.

Part 2: (AME40541: 30 points, AME60541: 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 $\Omega \subset \mathbb{R}^d$

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

where $\boldsymbol{F}(\boldsymbol{U}, \nabla \boldsymbol{U}; \boldsymbol{\iota}) \in \mathbb{R}^{N_c \times d}$ is a nonlinear flux function, $\boldsymbol{S}(\boldsymbol{U}, \nabla \boldsymbol{U}; \boldsymbol{\iota}) \in \mathbb{R}^{N_c}$ is a nonlinear source term, $\boldsymbol{U}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the primary solution variable, $\boldsymbol{\iota}(\boldsymbol{x}) \in \mathbb{R}^m$ 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

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

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

We solely consider a problem with natural boundary conditions to facilitate a convenient and general implementation. For regions 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 the corresponding degree of freedom and results in exactly the same weak formulation as setting the corresponding test function to zero.

In the code, we will describe a PDE of the form (10) using a MATLAB structure (`claw` or `eqn`) with four fields: `nvar` (number of PDE variables, N_c), `ndim` (number of spatial dimension, d), `npars` (number of parameters, m), and `srcflux` (function that returns source term, flux function, and their partial derivatives). We use function handles, called `vol_pars_fcn` and `bnd_pars_fcn`, to specify the parameter function $\boldsymbol{\iota}(\boldsymbol{x})$ and natural boundary conditions $\bar{\boldsymbol{q}}(\boldsymbol{x})$, respectively, in the code. Finally, we wrap the conservation law, parameter function, and natural boundary conditions into a MATLAB structure (`prob`).

We will consider four PDEs in this project: PDE0 from Homework 2, a generic second-order linear PDE, linear elasticity, and the incompressible Navier-Stokes equation. In the following subsections, we introduce the four partial differential equations and their flux formulations. We also provide a concrete example of the data structures used to fully prescribe the PDE and its data.

Part 2.1 PDE0 from Homework 2: (5 points) Recall PDE0 from Homework 2

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

We can immediately identify this as a scalar PDE ($N_c = 1$) in one spatial dimension ($d = 1$). To put this PDE in flux form (10), we identify

$$U_1 = u, \quad F_{11} = -\frac{du}{dx}, \quad S_1 = u - x^2, \quad \bar{q}_1 = -\mathbf{1}_{\{x=1\}}. \tag{11}$$

We take the parameter vector as $\boldsymbol{\iota}(\boldsymbol{x}) = x^2$.

As we will see in Part 4, we will need the partial derivatives of the flux function $\boldsymbol{F}$ and source term $\boldsymbol{S}$ with respect to the PDE state $\boldsymbol{U}$ and its gradient $\nabla\boldsymbol{U}$. It is easy to see these partial derivatives are

$$\frac{\partial F_{11}}{\partial U_1} = 0, \quad \frac{\partial F_{11}}{\partial U_{1,1}} = -1, \quad \frac{\partial S_1}{\partial U_1} = 1, \quad \frac{\partial S_1}{\partial U_{1,1}} = 0. \tag{12}$$

The parameter vector for PDE0 is $\boldsymbol{\iota}(\boldsymbol{x}) = x^2$ and the boundary condition is $\bar{\boldsymbol{q}}(\boldsymbol{x}) = -\mathbf{1}_{\{x=1\}}$, which we specify with the following `prob` structure:

```
prob.eqn = Pde0();
prob.vol_pars_fcn = @(x) x^2;
prob.bnd_pars_fcn = @(x, bnd) -1*(x>=1-1e-12);
```

Tasks for Part 2.1

Implement the flux function, source term, and their partial derivatives in a function with the following signature (see `_eqn/Pde0.m`):

```
function [S, dSdU, dSdQ, F, dFdU, dFdQ] = eval_pde0_srcflux(U, Q, pars)
%EVAL_PDE0_SRCFLUX Evaluate PDE0 source term, flux function and their
%partial derivatives at a point.
%
%Input arguments
%_________________
%   U : Array (NVAR, 1) : PDE state vector at a point
%
%   Q : Array (NVAR, NDIM) : Gradient of PDE state vector at a point
%
%   PARS : Array (NPARS, 1) : Vector of parameters at a point
%
%Output arguments
%__________________
%   S, dSdU, dSdQ, F, dFdU, dFdQ : See notation.m
```

Use `test_srcflux_findiff.m` to test your partial derivatives. Be aware that is will only determine if your partial derivatives are consistent with the flux and source expressions; it will NOT test whether the flux/source terms themselves are correct.

Part 2.2 Second-order, linear PDE: (10 points) A generic second-order linear partial differential equation takes the form

$$(-k_{ij}u_{,j})_{,i} = f \quad \text{in } \Omega, \qquad (k_{ij}u_{,j})n_i = \bar{q} \quad \text{on } \partial\Omega, \tag{13}$$

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 $\bar{q}(\boldsymbol{x}) \in \mathbb{R}$. This PDE models a number of diffusion-like processes, most notably heat flow. In Homework 2, you put this into flux form as

$$U_1 = u, \quad F_{1j} = -k_{js}u_{,s}, \quad S_1 = f, \quad \bar{q}_1 = -\bar{q}. \tag{14}$$

Furthermore, for this PDE, we take the parameter vector to be all entries of the coefficient matrix and the source term

$$\boldsymbol{\iota}(\boldsymbol{x}) = (k_{11}(\boldsymbol{x}), k_{21}(\boldsymbol{x}), \ldots, k_{dd}(\boldsymbol{x}), f(\boldsymbol{x})). \tag{15}$$

In the two-dimensional case ($d = 2$), the parameter vector is $\boldsymbol{\iota}(\boldsymbol{x}) = (k_{11}(\boldsymbol{x}), k_{21}(\boldsymbol{x}), k_{12}(\boldsymbol{x}), k_{22}(\boldsymbol{x}), f)$ and the boundary condition is $\bar{\boldsymbol{q}}(\boldsymbol{x}) = -\bar{q}(\boldsymbol{x})$. If we wish to prescribe $\boldsymbol{k}(\boldsymbol{x}) = \boldsymbol{I}_2$ (constant), $f(\boldsymbol{x}) = \sin(x_1)\cos(x_2)$, and $\bar{q}(\boldsymbol{x}) = x_1$ on $\partial\Omega_1$ and $\bar{q}(\boldsymbol{x}) = 1$ on $\partial\Omega_2$, we specify the following `prob` structure:

```
prob.eqn = LinearEllipticScalar(2);
prob.vol_pars_fcn = @(x) [1; 0; 0; 1; sin(x(1))*cos(x(2))];
prob.bnd_pars_fcn = @(x, bnd) -x(1)*(bnd==1) - 1*(bnd==2);
```

Tasks for Part 2.2

Your tasks for this section are to complete the derivation of the flux formulation of the second-order linear PDE and implement the various terms.

1) Derive the partial derivatives of the flux function and source term.

2) Implement the flux function, source term, and their partial derivatives in a function with the following signature (see `_eqn/LinearEllipticScalar.m`):

```
function [S, dSdU, dSdQ, F, dFdU, dFdQ] = eval_linelptc_sclr_srcflux(U, Q, pars)
%EVAL_LINELPTC_SCLR_SRCFLUX Evaluate linear elliptic PDE source term, flux
%function and their partial derivatives at a point.
%
%Input arguments
%_________________
%   U : Array (NVAR, 1) : PDE state vector at a point
%
```

```
%   Q : Array (NVAR, NDIM) : Gradient of PDE state vector at a point
%
%   PARS : Array (NPARS, 1) : Vector of parameters at a point
%
%Output arguments
%------------------
%   S, dSdU, dSdQ, F, dFdU, dFdQ : See notation.m
```

Use `test_srcflux_findiff.m` to test your partial derivatives. Be aware that is will only determine if your partial derivatives are consistent with the flux and source expressions; it will NOT test whether the flux/source terms themselves are correct.

Part 2.3 Linear elasticity: (15 points) The linear elasticity equations govern the deformation of a structure (infinitesimal strains)

$$-\sigma_{ij,j} = f_i \quad \text{in } \Omega, \qquad \sigma_{ij} n_j = \bar{t}_i \quad \text{on } \partial\Omega, \tag{16}$$

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 ith 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 Homework 2, you put this into flux form as

$$U_i = u_i, \quad F_{ij} = -\sigma_{ij}, \quad S_i = f_i, \quad \bar{q}_i = -\bar{t}_i. \tag{17}$$

Furthermore, for this PDE, we take the parameter vector to be the Lamé parameters and source term

$$\boldsymbol{\iota}(\boldsymbol{x}) = (\lambda(\boldsymbol{x}), \mu(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_d(\boldsymbol{x})). \tag{18}$$

In the two-dimensional case ($d = 2$), the parameter vector is $\boldsymbol{\iota}(\boldsymbol{x}) = (\lambda(\boldsymbol{x}), \mu(\boldsymbol{x}), f_1(\boldsymbol{x}), f_2(\boldsymbol{x}))$ and the boundary condition is $\bar{\boldsymbol{q}}(\boldsymbol{x}) = (-\bar{t}_1(\boldsymbol{x}), -\bar{t}_2(\boldsymbol{x}))$. If we wish to prescribe $\lambda(\boldsymbol{x}) = \mu(\boldsymbol{x}) = \mathbf{1}_{\{x_1>0, x_2<0\}}$, $f(\boldsymbol{x}) = (0, -1)$, and $\bar{\boldsymbol{t}} = (0, -1)$, we specify the following problem structure:

```
prob.eqn = LinearElasticity(2);
prob.vol_pars_fcn = @(x) [(x(1)>0).*(x(2)<0); (x(1)>0).*(x(2)<0); 0; -1];
prob.bnd_pars_fcn = @(x, bnd) [0; 1];
```

Tasks for Part 2.3

Your tasks for this section are to complete the derivation of the flux formulation of the linear elasticity PDEs and implement the various terms.

1) Derive the partial derivatives of the flux function and source term.

2) Implement the flux function, source term, and their partial derivatives in a function with the following signature (see `_eqn/LinearElasticity.m`):

```
function [S, dSdU, dSdQ, F, dFdU, dFdQ] = eval_linelast_srcflux(U, Q, pars)
%EVAL_LINELAST_SRCFLUX Evaluate linear elasticity source term, flux
%function and their partial derivatives at a point.
%
%Input arguments
%------------------
%   U : Array (NVAR, 1) : PDE state vector at a point
%
%   Q : Array (NVAR, NDIM) : Gradient of PDE state vector at a point
%
%   PARS : Array (NPARS, 1) : Vector of parameters at a point
```

```
%
%Output arguments
%————————————————
%   S, dSdU, dSdQ, F, dFdU, dFdQ : See notation.m
```

Use `test_srcflux_findiff.m` to test your partial derivatives. Be aware that is will only determine if your partial derivatives are consistent with the flux and source expressions; it will NOT test whether the flux/source terms themselves are correct.

Part 2.4 Incompressible Navier-Stokes: (AME60541) (20 points) The incompressible Navier-Stokes equations model the flow of a viscous fluid with constant density

$$-(\rho\nu v_{i,j})_j + \rho v_j v_{i,j} + P_{,i} = 0, \quad v_{j,j} = 0, \quad \text{in } \Omega \tag{19}$$

for $i = 1, \ldots, d$ with the boundary conditions

$$(\rho\nu v_{i,j} - P\delta_{ij})n_j = \rho\bar{t}_i \quad \text{on } \partial\Omega, \tag{20}$$

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. In Homework 2, you put this into flux form as

$$U_i = \begin{cases} v_i & i < d+1 \\ P & i = d+1 \end{cases}, \quad F_{ij} = \begin{cases} -\rho\nu v_{i,j} + P\delta_{ij} & i < d+1 \\ 0 & i = d+1 \end{cases}, \quad S_i = \begin{cases} -\rho v_j v_{i,j} & i < d+1 \\ -v_{s,s} & i = d+1 \end{cases}$$

$$\bar{q}_i = \begin{cases} -\rho\bar{t}_i & i < d+1 \\ 0 & i = d+1 \end{cases}$$

Furthermore, for this PDE, we take the parameter vector to be the density and viscosity

$$\boldsymbol{\iota}(\boldsymbol{x}) = (\rho(\boldsymbol{x}), \nu(\boldsymbol{x})). \tag{21}$$

In the two-dimensional case ($d = 2$), the parameter vector is $\boldsymbol{\iota}(\boldsymbol{x}) = (\rho(\boldsymbol{x}), \nu(\boldsymbol{x}))$ and the boundary condition is $\bar{\boldsymbol{q}}(\boldsymbol{x}) = (-\bar{t}_1(\boldsymbol{x}), -\bar{t}_2(\boldsymbol{x}), 0)$. If we wish to prescribe $\rho(\boldsymbol{x}) = 1$, $\nu(\boldsymbol{x}) = 0.1$ and $\bar{\boldsymbol{t}} = (1, 2)$, we specify the following `prob` structure:

```
prob.eqn = IncompressibleNavierStokes(2);
prob.vol_pars_fcn = @(x) [1; 0.1];
prob.bnd_pars_fcn = @(x, bnd) [-1; -2; 0];
```

Tasks for Part 2.4

Your tasks for this section are to complete the derivation of the flux formulation of the incompressible Navier-Stokes equations and implement the various terms.

1) Derive the partial derivatives of the flux function and source term.

2) Implement the flux function, source term, and their partial derivatives in a function with the following signature (see `_eqn/IncompressibleNavierStokes.m`).

```
function [S, dSdU, dSdQ, F, dFdU, dFdQ] = eval_ins_srcflux(U, Q, pars)
%EVAL_INS_SRCFLUX Evaluate incompressible Navier-Stokes source term, flux
%function and their partial derivatives at a point.
%
%Input arguments
%————————————————
%   U : Array (NVAR, 1) : PDE state vector at a point
```

```
%
%   Q : Array (NVAR, NDIM) : Gradient of PDE state vector at a point
%
%   PARS : Array (NPARS, 1) : Vector of parameters at a point
%
%Output arguments
%------------------
%   S, dSdU, dSdQ, F, dFdU, dFdQ : See notation.m
```

Use `test_srcflux_findiff.m` to test your partial derivatives. Be aware that is will only determine if your partial derivatives are consistent with the flux and source expressions; it will NOT test whether the flux/source terms themselves are correct.

Part 3: (AME40541: 10 points, AME60541: 25 points) Next we turn to building up a basis for a finite element using the local function space and transformation from Part 1. Recall that, in addition to the element geometry, a finite element is defined by the associated function space and degrees of freedom (nodal values in our case). Let $\hat{\boldsymbol{U}}^e \in \mathbb{R}^{N^{\text{el}}_{\text{dof}}}$ be the collection of nodal degrees of freedom and define the matrix of basis functions $\boldsymbol{\Psi}(\boldsymbol{\xi}) \in \mathbb{R}^{N^{\text{el}}_{\text{dof}} \times N_c}$ over the reference element $\Omega_\square$ that maps the local degrees of freedom associated with element e to the primary variables evaluated at $\boldsymbol{x} = \mathcal{G}_e(\boldsymbol{\xi})$:

$$\boldsymbol{U}(\boldsymbol{x})|_{\Omega_e} = \boldsymbol{\Phi}^e(\boldsymbol{x})^T \hat{\boldsymbol{U}}^e, \qquad U_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{N^{\text{el}}_{\text{dof}}} \Phi^e_{ji}(\boldsymbol{x}) \hat{U}^e_j. \tag{22}$$

where the basis functions over the physical domain are

$$\boldsymbol{\Phi}^e(\boldsymbol{x}) := \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x})), \qquad \frac{\partial \boldsymbol{\Phi}^e}{\partial \boldsymbol{x}}(\boldsymbol{x}) = \frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \cdot \left[\frac{\partial \mathcal{G}_e}{\partial \boldsymbol{\xi}}(\mathcal{G}_e^{-1}(\boldsymbol{x})) \right]^{-1}. \tag{23}$$

In the following subsections, we consider two different families of vector-valued finite elements, a standard element where all solution components are approximated in the same local function space, and a mixed element where different components are approximated in different function spaces.

Part 3.1 (10 points) First we consider the a standard element where all solution components are approximated using the same function space, i.e.,

$$U_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{N^{\text{el}}_{\text{nd}}} \phi^e_j(\boldsymbol{x}) \hat{U}^e_{ij} \tag{24}$$

where $\{\phi_1, \ldots, \phi_{N^{\text{el}}_{\text{nd}}}\}$ is a basis of the local function space over Ω_e defined in terms of the master element basis $\{\psi_1, \ldots, \psi_{N^{\text{el}}_{\text{nd}}}\}$ as $\phi^e_i(\boldsymbol{x}) = \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x}))$ and the element degrees of freedom are

$$\hat{\boldsymbol{U}}^e = \begin{bmatrix} \hat{\boldsymbol{U}}^e_1 \\ \vdots \\ \hat{\boldsymbol{U}}^e_{N^{\text{el}}_{\text{nd}}} \end{bmatrix}, \qquad \hat{\boldsymbol{U}}^e_i = \begin{bmatrix} \hat{U}^e_{1i} \\ \vdots \\ \hat{U}^e_{N_c i} \end{bmatrix}, \tag{25}$$

where $\hat{U}^e_{ji}$ is the degree of freedom at node i corresponding to $U_j(\boldsymbol{x})|_{\Omega_e}$. From this numbering of the degrees of freedom, it can be seen the matrix of master basis functions must be

$$\boldsymbol{\Psi}(\boldsymbol{\xi}) = \begin{bmatrix} \psi_1(\boldsymbol{\xi}) \boldsymbol{I}_{N_c} \\ \vdots \\ \psi_{N^{\text{el}}_{\text{nd}}}(\boldsymbol{\xi}) \boldsymbol{I}_{N_c} \end{bmatrix} \tag{26}$$

where $\boldsymbol{I}_{N_c} \in \mathbb{R}^{N_c \times N_c}$ is the identity matrix.

Tasks for Part 3.1
Your task for this part is to implement a function with the following signature that creates the element basis $\boldsymbol{\Psi}(\boldsymbol{\xi})$ and its partial derivatives $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi})$ from the scalar local function space basis $\psi_1(\boldsymbol{\xi}), \ldots, \psi_{N_{\text{nd}}^{\text{el}}}(\boldsymbol{\xi})$, evaluated at the appropriate quadrature points:

```
function [Tv, Tvf] = create_elem_basis(nvar, Qv, Qvf)
%CREATE_ELEM_BASIS Create element basis evaluated at point throughout
%volume (TV) and on each face (TVF) from basis of local function space
%evaluated at corresponding points (QV, QVF).
%
% Input arguments
% ---------------
%   NVAR, QV, QVF : See notation.m
%
% Output arguments
% ----------------
%   TV, TVF : See notation.m
```

Part 3.2 (AME60541) (15 points) Next we consider the a mixed element where the solution components are partitioned into two groups,

$$\boldsymbol{U}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{V}(\boldsymbol{x}) \\ \boldsymbol{W}(\boldsymbol{x}) \end{bmatrix}, \tag{27}$$

where $\boldsymbol{V}(\boldsymbol{x}) \in \mathbb{R}^M$ contains the first M components of $\boldsymbol{U}(\boldsymbol{x})$ and $\boldsymbol{W}(\boldsymbol{x}) \in \mathbb{R}^{N_c - M}$ contains the last $N_c - M$ components of $\boldsymbol{U}(\boldsymbol{x})$. Each group of variables is approximated using a different function space, i.e.,

$$V_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{N_{\text{nd}}^{\text{el}}} \phi_j^e(\boldsymbol{x}) \hat{V}_j^e \quad \text{for } i = 1, \ldots, M, \qquad W_i(\boldsymbol{x})|_{\Omega_e} = \sum_{j=1}^{\tilde{N}_{\text{nd}}^{\text{el}}} \tilde{\phi}_j^e(\boldsymbol{x}) \hat{W}_j^e \quad \text{for } i = M+1, \ldots, N_c \tag{28}$$

where $\{\phi_1, \ldots, \phi_{N_{\text{nd}}^{\text{el}}}\}$ and $\{\tilde{\phi}_1, \ldots, \tilde{\phi}_{\tilde{N}_{\text{nd}}^{\text{el}}}\}$ are bases for the two local function spaces considered. They are defined in terms of master element bases $\{\psi_1, \ldots, \psi_{N_{\text{nd}}^{\text{el}}}\}$ and $\{\tilde{\psi}_1, \ldots, \tilde{\psi}_{\tilde{N}_{\text{nd}}^{\text{el}}}\}$, respectively, as

$$\phi_i^e(\boldsymbol{x}) = \psi_i(\mathcal{G}_e^{-1}(\boldsymbol{x})) \quad \text{for } i = 1, \ldots, N_{\text{nd}}^{\text{el}}, \qquad \tilde{\phi}_i^e(\boldsymbol{x}) = \tilde{\psi}_i(\mathcal{G}_e^{-1}(\boldsymbol{x})) \quad \text{for } i = 1, \ldots, \tilde{N}_{\text{nd}}^{\text{el}}. \tag{29}$$

The element degrees of freedom are defined as

$$\hat{\boldsymbol{U}}^e = \begin{bmatrix} \hat{\boldsymbol{V}}^e \\ \hat{\boldsymbol{W}}^e \end{bmatrix}, \quad \hat{\boldsymbol{V}}^e = \begin{bmatrix} \hat{\boldsymbol{V}}_1^e \\ \vdots \\ \hat{\boldsymbol{V}}_{N_{\text{nd}}^{\text{el}}}^e \end{bmatrix}, \quad \hat{\boldsymbol{V}}_i^e = \begin{bmatrix} \hat{V}_{1i}^e \\ \vdots \\ \hat{V}_{Mi}^e \end{bmatrix}, \quad \hat{\boldsymbol{W}}^e = \begin{bmatrix} \hat{\boldsymbol{W}}_1^e \\ \vdots \\ \hat{\boldsymbol{W}}_{\tilde{N}_{\text{nd}}^{\text{el}}}^e \end{bmatrix}, \quad \hat{\boldsymbol{W}}_i^e = \begin{bmatrix} \hat{W}_{1i}^e \\ \vdots \\ \hat{W}_{(N_c-M)i}^e \end{bmatrix}, \tag{30}$$

where $\hat{V}_{ji}^e$ is the degree of freedom at node i corresponding to $V_j(\boldsymbol{x})|_{\Omega_e}$ and $\hat{W}_{ji}^e$ is the degree of freedom at node i corresponding to $W_j(\boldsymbol{x})|_{\Omega_e}$. From this arrangement the degrees of freedom, the master basis function matrix must be

$$\boldsymbol{\Psi}(\boldsymbol{\xi}) = \begin{bmatrix} \psi_1(\boldsymbol{\xi}) \boldsymbol{I}_M & & \\ \vdots & & \\ \psi_{N_{\text{nd}}^{\text{el}}}(\boldsymbol{\xi}) \boldsymbol{I}_M & & \\ & \tilde{\psi}_1(\boldsymbol{\xi}) \boldsymbol{I}_{N_c - M} & \\ & \vdots & \\ & \tilde{\psi}_{\tilde{N}_{\text{nd}}^{\text{el}}}(\boldsymbol{\xi}) \boldsymbol{I}_{N_c - M} \end{bmatrix}. \tag{31}$$

Tasks for Part 3.2
Your task for this part is to implement a function with the following signature that creates the element basis $\boldsymbol{\Psi}(\boldsymbol{\xi})$ and its partial derivatives $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi})$ from the scalar local function space basis $\psi_1(\boldsymbol{\xi}), \ldots, \psi_{N_{\text{nd}}^{\text{el}}}(\boldsymbol{\xi})$ and $\tilde{\psi}_1(\boldsymbol{\xi}), \ldots, \tilde{\psi}_{\tilde{N}_{\text{nd}}^{\text{el}}}(\boldsymbol{\xi})$, evaluated at the appropriate quadrature points:

```
function [Tv, Tvf] = create_elem_basis_mixed2(nvar1, Qv1, Qvf1, ...
                                              nvar2, Qv2, Qvf2)
%CREATE_ELEM_BASIS_MIXED2 Create element basis for mixed element (two
%different local function spaces) evaluated at point throughout volume (TV)
%and on each face (TVF) from basis of local function spaces evaluated at
%corresponding points (QV1, QVF1, QV2, QVF2).
%
% Input arguments
% ---------------
%   NVAR1, QV1, QVF1 : NVAR, QV, QVF (see notation.m) for the first local
%     function space, e.g., velocity in Navier-Stokes.
%
%   NVAR2, QV2, QVF2 : NVAR, QV, QVF (see notation.m) for the second local
%     function space, e.g., pressure in Navier-Stokes.
%
% Output arguments
% ----------------
%   TV, TVF : See notation.m
```

Part 4: (40 points) Now that we have all ingredients in place, we turn to the core of our finite element code. In Homework 2, we derived the weak formulation for the general PDE in (10) as

$$\int_{\Omega} (-w_{i,j} F_{ij} - w_i S_i)\, dv + \int_{\partial\Omega} w_i \bar{q}_i\, ds = 0 \tag{32}$$

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 Ω_e, we have

$$\sum_{e=1}^{N_{\text{el}}} \int_{\Omega_e} (-w_{i,j} F_{ij} - w_i S_i)\, dv + \sum_{e=1}^{N_{\text{el}}} \int_{\partial\Omega_e \cap \partial\Omega} w_i \bar{q}_i\, ds = 0, \tag{33}$$

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 master element transformation leads to

$$B^e(\boldsymbol{w}, \boldsymbol{U}) = \int_{\Omega_\square} \left(-w_i S_i - \frac{\partial w_i}{\partial x_j} F_{ij} \right) g_e\, dV + \sum_{f=1}^{N_{\text{fc}}^{\text{el}}} \mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \varnothing\}} \int_{\Gamma_\square} w_i \bar{q}_i \sigma_{ef}\, dS \tag{34}$$

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

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

$$\hat{U}_i^e = \hat{U}_{\Xi_{ie}}, \quad \text{where } \Xi_{ie} = \texttt{ldof2gdof}(i, e)$$

for $i = 1, \ldots, N_{\text{dof}}^{\text{el}}$, $e = 1, \ldots, N_{\text{el}}$.

Since the FEM is a Galerkin approximation, the trial and test functions are interpolated using the same basis functions, i.e.,

$$\boldsymbol{U}(\boldsymbol{x})|_{\Omega_e} = \boldsymbol{\Phi}^e(\boldsymbol{x})^T \hat{\boldsymbol{U}}^e, \qquad \boldsymbol{w}(\boldsymbol{x})|_{\Omega_e} = \boldsymbol{\Phi}^e(\boldsymbol{x})^T \hat{\boldsymbol{w}}^e, \tag{35}$$

where $\hat{\boldsymbol{w}}^e \in \mathbb{R}^{N_{\text{dof}}^{\text{el}}}$ is a vector of all degrees of freedom (test variables) associated with element e. The physical basis is written in terms of the master basis as (23): $\boldsymbol{\Phi}^e(\boldsymbol{x}) = \boldsymbol{\Psi}(\mathcal{G}_e^{-1}(\boldsymbol{x}))$. The specific form of $\boldsymbol{\Psi}$ will depend on the type of element used, i.e., (26) for a standard element and (31) for a mixed element.

Substitute the expansions in (35) into the element weak form and identify the term multiplying $\hat{w}_l^e$ as the lth component of the element residual, $\hat{\boldsymbol{R}}^e(\hat{\boldsymbol{U}}^e)$

$$\hat{R}_l^e(\hat{\boldsymbol{U}}_e) = \int_{\Omega_\square} \left(-\Psi_{li} S_i - \frac{\partial \Psi_{li}}{\partial x_j} F_{ij} \right) g_e \, dV + \sum_{f=1}^{N_{\text{fc}}^{\text{el}}} \mathbf{1}_{\{\partial\Omega_{ef} \cap \partial\Omega \neq \emptyset\}} \int_{\Gamma_\square} \Psi_{li} \bar{q}_i \sigma_{ef} \, dS$$

where $\frac{\partial \Psi_{li}}{\partial x_j} = \frac{\partial \Psi_{li}}{\partial \xi_s} [G_e^{-1}]_{sj}$ and summation is implied over repeated indices. 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

$$\frac{\partial \hat{R}_l^e}{\partial \hat{U}_r^e}(\hat{\boldsymbol{U}}^e) = \int_{\Omega_\square} \left(-\Psi_{li} \left(\frac{\partial S_i}{\partial u_t} \Psi_{rt} + \frac{\partial S_i}{\partial q_{ts}} \frac{\partial \Psi_{rt}}{\partial x_s} \right) - \frac{\partial \Psi_{li}}{\partial x_j} \left(\frac{\partial F_{ij}}{\partial u_t} \Psi_{rt} + \frac{\partial F_{ij}}{\partial q_{ts}} \frac{\partial \Psi_{rt}}{\partial x_s} \right) \right) g_e \, dV, \tag{36}$$

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 constrained degrees of freedom $\hat{\boldsymbol{U}}_c$ (those with an essential boundary condition) and unconstrained degrees of freedom $\hat{\boldsymbol{U}}_u$: $\hat{\boldsymbol{U}} = (\hat{\boldsymbol{U}}_u^T, \hat{\boldsymbol{U}}_c^T)^T$. The global residual and Jacobian are partitioned similarly

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

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

$$\hat{\boldsymbol{R}}_u(\hat{\boldsymbol{U}}_u; \hat{\boldsymbol{U}}_c) = 0,$$

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

To close this section, 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

$$I = \int_\Omega \boldsymbol{U}(\boldsymbol{x})^T \boldsymbol{U}(\boldsymbol{x}) \, dv, \tag{37}$$

where $\boldsymbol{U}(\boldsymbol{x}) \in \mathbb{R}^{N_c}$ is the PDE solution in (10). Using the finite element approximation in (22), we have

$$I = \sum_{e=1}^{N_{\text{el}}} \int_{\Omega_e} (\hat{\boldsymbol{U}}^e)^T \boldsymbol{\Phi}^e(\boldsymbol{x}) \boldsymbol{\Phi}^e(\boldsymbol{x})^T \hat{\boldsymbol{U}}^e \, dv = \sum_{e=1}^{N_{\text{el}}} (\hat{\boldsymbol{U}}^e)^T \boldsymbol{M}^e \hat{\boldsymbol{U}}^e, \tag{38}$$

where

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

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

$$I = \hat{\boldsymbol{U}}^T \boldsymbol{M} \hat{\boldsymbol{U}}, \tag{39}$$

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

Tasks for Part 4

Your task is to complete the core of your FEM code for general, second-order PDEs in (10) and test it using PDE0.

1) Implement a function that evaluates the *volume contribution* of 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 rule). Your functions should have the following signature:

```
function [Re, dRe] = intg_elem_claw_vol(Ue, transf_data, elem, elem_data)
%INTG_ELEM_CLAW_VOL Integrate element Galerkin form (volume term) to
%form the volume contribution to the element residual and Jacobian.
%
% Input arguments
% ---------------
%   UE : Array (NDOF_PER_ELEM,) : Element solution (primary variables)
%
%   TRANSF_DATA, ELEM, ELEM_DATA : See notation.m
%
% Output arguments
% ----------------
%   RE : Array (NDOF_PER_ELEM,) : Element residual (volume contribution)
%
%   DRE : Array (NDOF_PER_ELEM, NDOF_PER_ELEM) : Element Jacobian (volume contribution)
```

2) Implement a function that evaluates the *boundary contribution* of the discrete element residual $\hat{\boldsymbol{R}}^e(\hat{\boldsymbol{U}}^e)$ given information about the element (geometry, basis functions, quadrature rule). Your functions should have the following signature:

```
function [Re] = intg_elem_claw_extface(transf_data, elem, elem_data)
%INTG_ELEM_CLAW_EXTFACE Integrate element Galerkin form (boundary term) to
%form the boundary contribution to the element residual and Jacobian.
%
% Input arguments
% ---------------
%   TRANSF_DATA, ELEM, ELEM_DATA : See notation.m
%
% Output arguments
% ----------------
%   RE : Array (NDOF_PER_ELEM,) : Element residual (boundary contribution)
```

3) 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_claw_cg(U, transf_data, elem, elem_data, ...
    ldof2gdof)
%EVAL_UNASSEMBLED_RESJAC Evaluate/store element residual vector and
%Jacobian matrices for each element.
%
% Input arguments
% ---------------
%   U : Array (NDOF,) : Global (assembled) solution vector
%
%   TRANSF_DATA, ELEM, ELEM_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
```

4) Implement a function that evaluates the assembled element residual and Jacobian. Your function should have the following signature:

```
function [R, dR] = eval_assembled_resjac_claw_cg(U, transf_data, elem, elem_data, ...
                                                 ldof2gdof, spmat)
%EVAL_ASSEMBLED_RESJAC_CLAW_CG Evaluate assembled residual vector and
%Jacobian matrix
%
%Input arguments
%----------------
%   U : Array (NDOF,) : Global (assembled) solution vector
%
%   TRANSF_DATA, ELEM, ELEM_DATA, SPMAT : See notation.m
%
%Output arguments
%-----------------
%   R : Array (NDOF,) : Assembled residual vector PRIOR to static condensation
%
%   dR : Array (NDOF, NDOF) : Assembled Jacobian matrix PRIOR to static condensation
```

5) Implement a function that evaluates the global (assembled) finite element residual and Jacobian. This function will depend on the finite element space (`femsp`) structure; see `create_femsp_cg.m`. The `femsp` structure is central to your finite element code as it contains all required information (both generic and equation-specific) to evaluate the element residual and Jacobian; `create_femsp_cg.m` is a wrapper for many of the functions you wrote in previous parts of the project. Your function should have the following signature:

```
function [Ru, dRu] = create_fem_resjac(Uu, femsp)
%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 FEMSP).
%
% Input arguments
% ----------------
%   UU : Array (NDOF-NDBC,) : Global (assembled) solution vector,
%     restricted to the free degrees of freedom (via static condensation).
%
%   FEMSP : See notation.m
%
% Output arguments
% -----------------
%   RU : Array (NDOF-NDBC,) : Finite element residual, restricted to free
%     degrees of freedom
%
%   DRU : Sparse matrix (NDOF-NDBC, NDOF-NDBC) : Finite element Jacobian
%     restricted to free degrees of freedom
```

This function will be used by `solve_fem.m` that uses the Newton-Raphson method to solve the final finite element system after static condensation.

6) Test your finite element code with PDE0 and verify optimal convergence rates. That is, for a sequence of meshes with 2^k elements for $k = 1, 2, 3, 4$ 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 $\hat{\boldsymbol{U}} \in \mathbb{R}^{N_{\text{dof}}}$ is the global (assembled) finite element solution and $\boldsymbol{U}^* \in \mathbb{R}^{N_{\text{dof}}}$ 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 5: (AME40541: 30 points, AME60541: 40 points) In this part of the project, we will use your finite element code to solve several second-order linear PDEs, e.g., heat flow, over various domains.

Tasks for Part 5

Your tasks for this section are to solve second-order linear PDEs over a disk (and verify convergence rates), the Batman domain, the ND domain, and a cube (AME60541).

1) Consider the Poisson equation on the unit disk

$$-\Delta u = 1 \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega, \tag{40}$$

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

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

Starter code is provided in `solve_linelptc_sclr_disk0.m`.

- Use your FEM code to solve the above Poisson problem. Plot the solution $u(\boldsymbol{x})$ over the domain Ω and along any line that passes through the center of the disk. Use a mesh consisting of 20×20 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., $h = \sqrt{V(\Omega)/N_{\text{el}}}$. Discuss similarities and differences between the convergence rates and absolute error for a given element size between the simplex and hypercube meshes.

2) Use your FEM code to solve a second-order linear PDE (13) on the Batman domain (Figure 6) 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 (13), this corresponds to

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}, \qquad f(\boldsymbol{x}) = 0, \qquad \bar{q}(\boldsymbol{x}) = \begin{cases} 10\sin(x_1) & \boldsymbol{x} \in \partial\Omega_1 \cup \partial\Omega_2 \\ 0 & \text{otherwise,} \end{cases} \tag{41}$$

and $u(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \partial\Omega_3$. Use the $p = 2$ simplicial mesh provided (`msh = load_mesh('batman0', ... 'simp', 0, 2)`). Plot the solution over the entire domain Ω and along the line Γ (Figure 6). Starter code is provided in `solve_linelptc_sclr_batman0.m`.

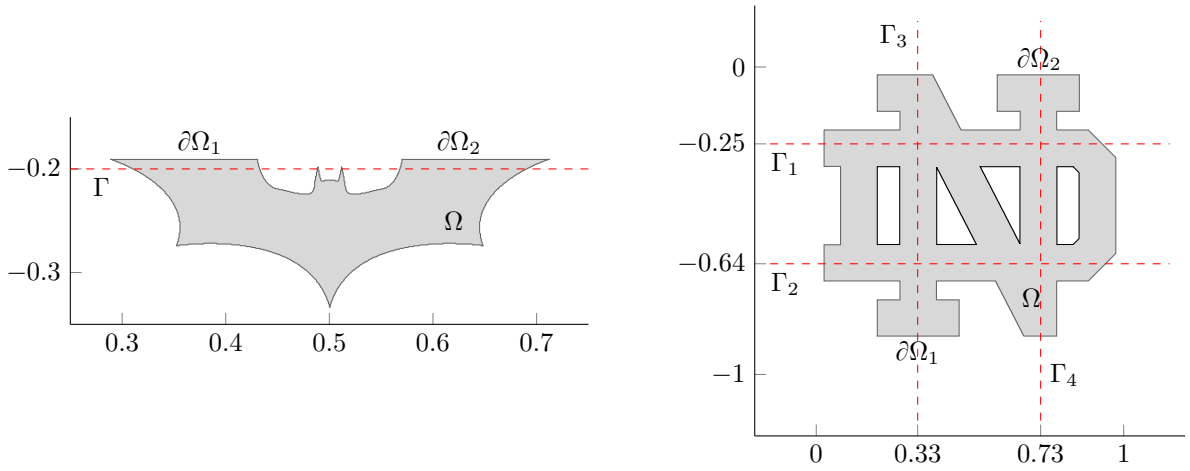

Figure 6: Domain (Ω), boundaries ($\partial\Omega_i$), and lines along which to evaluate quantities (Γ_i) for Batman symbol and Notre Dame logo. For each domain, the first two boundaries are shown above and the third boundary is $\partial\Omega_3 = \partial\Omega \backslash (\partial\Omega_1 \cup \partial\Omega_2)$.

3) Use your FEM code to solve the Poisson equation (13) on the ND logo domain (Figure 6) 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 $\boldsymbol{k}(\boldsymbol{x}) = \boldsymbol{I}_2$ (2×2 identity matrix). In the notation of (13), this corresponds to

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad f(\boldsymbol{x}) = 0, \qquad \bar{q}(\boldsymbol{x}) = 0, \qquad \boldsymbol{U}(\boldsymbol{x}) = \begin{cases} 0 & \boldsymbol{x} \in \partial\Omega_1 \\ 10 & \boldsymbol{x} \in \partial\Omega_2. \end{cases} \tag{42}$$

Use the $p = 2$ simplicial mesh provided (`msh = load_mesh('nd0', 'simp', 0, 2)`). Plot the solution over the entire domain Ω and along the lines Γ_1, Γ_2, Γ_3, Γ_4 (Figure 6). Starter code is provided in `solve_linelptc_sclr_nd0.m`.

4) (AME60541) Use your FEM code to solve the second-order linear PDE in (13) defined over the unit cube ($\Omega = [0,1]^3$) with coefficient matrix

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

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 $\Gamma = \{(x,y,z) \mid 0 \leqslant x \leqslant 1, 0 \leqslant y \leqslant 1, z = 0.5\}$.

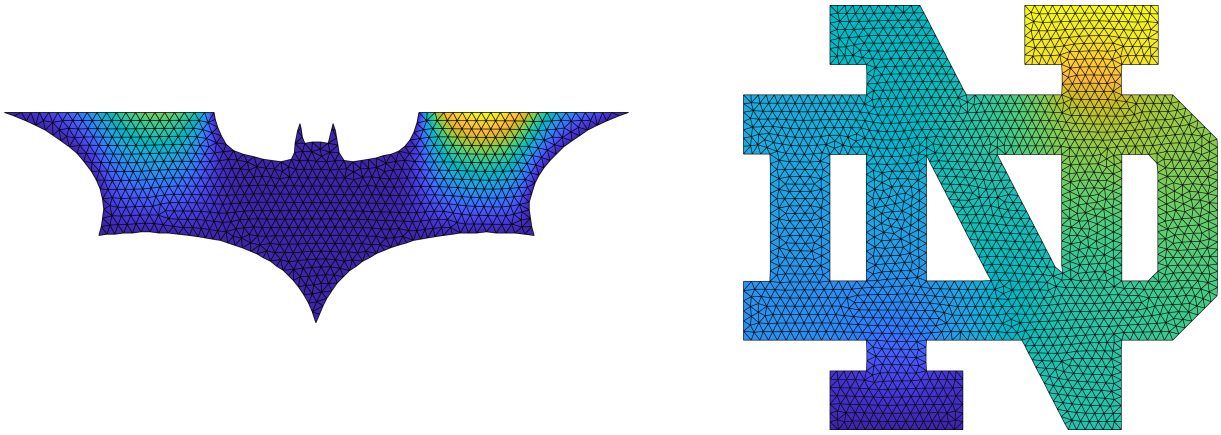

Figure 7: Solution of Poisson equation on Batman and ND logo domains.

Part 6: (AME40541: 20 points, AME60541: 40 points) In this part of the project, we will use our finite element code to solve several linear elasticity, e.g., structural deformation, problems over various domains.

Tasks for Part 6

Your tasks for this section are to solve the linear elasticity equations for a multi-material beam and hollow cylinder (AME60541).

1) 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. Starter code is provided in `solve_ ... linelast_beam0.m`.
   - Evaluate the displacements u_1, u_2 along the line Γ shown in Figure 8 and plot the magnitude of the displacement on the deformed geometry.

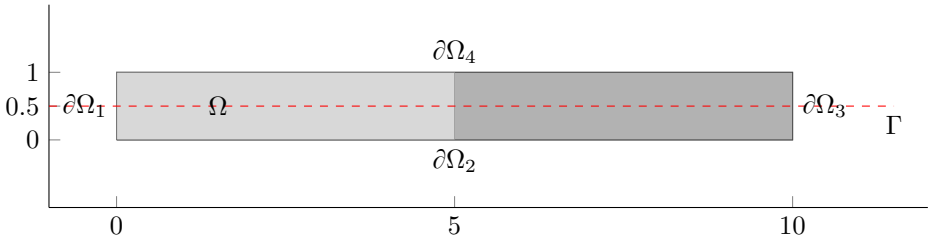

Figure 8: Multimaterial beam (Ω), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities (Γ).

2) (AME60541) 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 magnitude of the displacement on the surface of the deformed cylinder.
   - Plot the displacements u_1, u_2, u_3 along the line Γ shown in Figure 9.

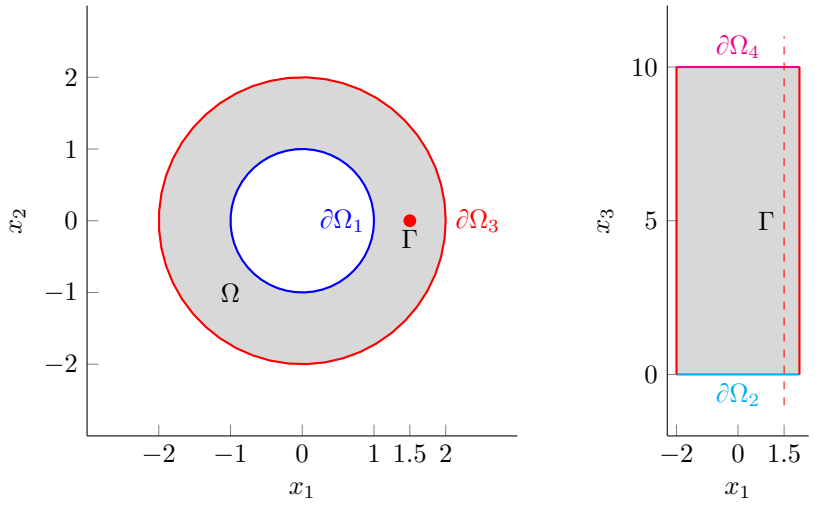

Figure 9: Hollow cylinder domain (Ω), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities (Γ); $x_1 - x_2$ view (*left*) and $x_1 - x_3$ view (*right*). The boundaries $\partial\Omega_1$, $\partial\Omega_3$ are the inner, outer cylindrical surfaces, respectively. The boundaries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

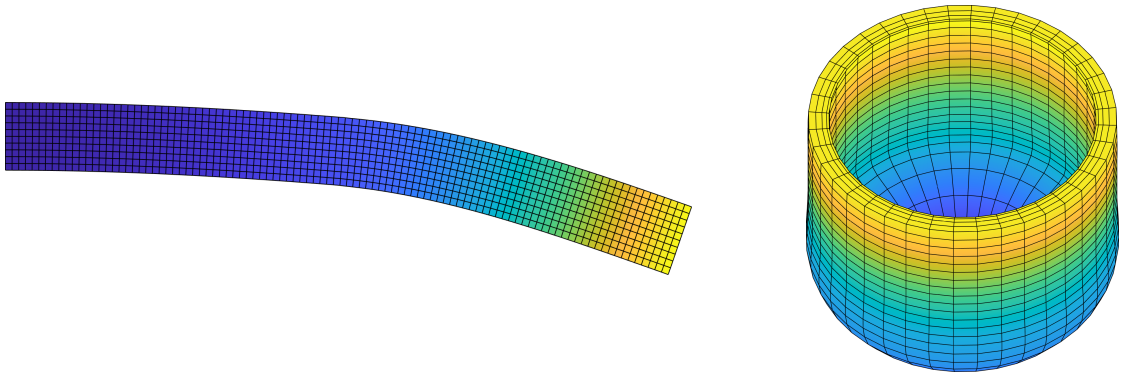

Figure 10: Solution of linear elasticity equations of multimaterial beam and hollow cylinder.

Part 7: (AME60541) (40 points) In this part of the project, we will use our finite element code to solve several incompressible Navier-Stokes equations, e.g., viscous fluid flow, through various domains. An important non-dimensional quantity in the study of fluid flow is the Reynolds number

$$Re = \frac{UL}{\nu}, \tag{43}$$

where U is the velocity of the fluid with respect to an object, L is the characteristic linear dimension, and ν 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.

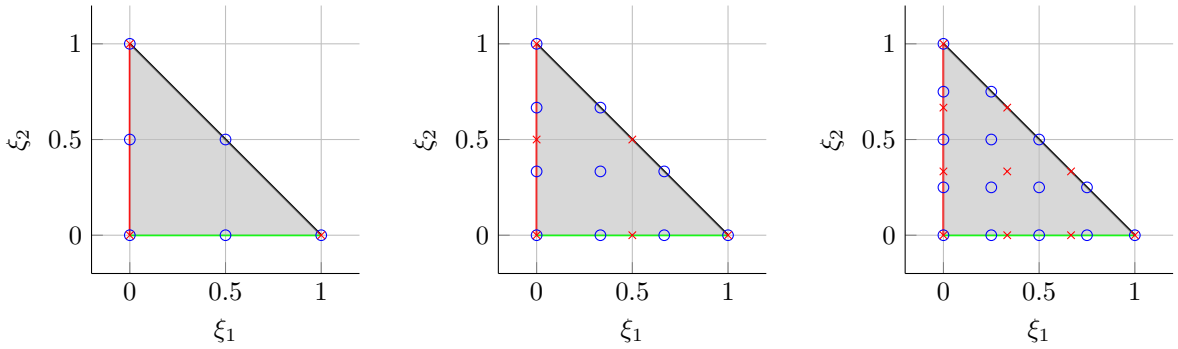

Figure 11: Mixed (velocity-pressure) triangular finite element in reference domain (ξ_1-ξ_2 space): $\mathcal{P}^2$-$\mathcal{P}^1$ (quadratic approximation of velocity, linear approximation of pressure) (*left*), $\mathcal{P}^3$-$\mathcal{P}^2$ (cubic approximation of velocity, quadratic approximation of pressure) (*center*), $\mathcal{P}^4$-$\mathcal{P}^3$ (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 $\boldsymbol{v}_i^e$ ($i = 1, \ldots, N_{\text{nd}}^{\text{el}}$) at each (○) and a pressure degree of freedom P_i^e ($i = 1, \ldots, \tilde{N}_{\text{nd}}^{\text{el}}$) 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 11). In the notation of Part 3, this mean we take $M = d$ and $N_c = d+1$. 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.

Tasks for Part 7

Your tasks for this section are to solve the lid-driven cavity problem and flow through the ND logo.

1) Use your code to solve the lid-driven cavity problem. The lid-driven cavity is defined on a square domain (Figure 12) 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 $\partial\Omega_4$. Since the pressure is only determined up to a constant, we need to prescribe it at one point on the boundary, e.g., take $P = 0$ at the point ($x_1 = 0$, $x_2 = 0$). 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 $Re = 100$. Starter code is provided in `solve_ins_ldc0.m`.
   - Plot the velocities v_1, v_2 along the line Γ shown in Figure 12 and the magnitude of the velocity throughout the domain.
   - 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 throughout the domain and superimpose a quiver plot that shows the direction of the flow.
     - Plot both components of the velocity along the the line Γ defined in Figure 12.

2) Use your code to solve for flow through the ND logo (Figure 6) with boundary conditions: vertical inflow on $\partial\Omega_1$ ($v_1 = 0$, $v_2 = 1$), traction-free outflow on $\partial\Omega_2$ ($\bar{t}_1 = \bar{t}_2 = 0$), and a no slip wall on $\partial\Omega_3$ ($v_1 = v_2 = 0$). Because there is a natural boundary condition, you do not need to explicitly fix the pressure along the boundary. Take the characteristic length scale to be $L = 1$, the characteristic velocity to be the inlet speed $U = 1$, and the density to be $\rho(\boldsymbol{x}) = 1$. Starter code provided in `solve_ins_nd0`.

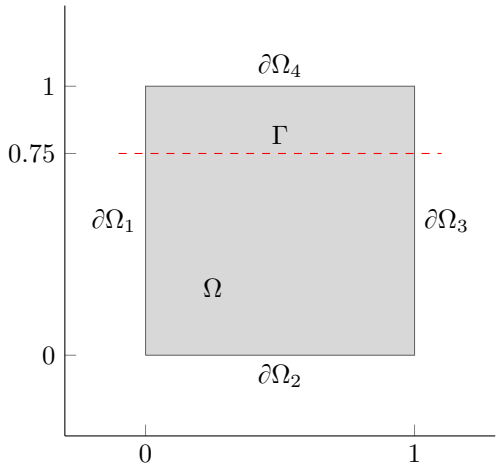

Figure 12: Lid-driven cavity domain (Ω), boundaries ($\partial\Omega_i$), and line along which to evaluate quantities (Γ).

- Solve for the flow velocity and pressure at $Re = 1300$; you will need to use continuation to solve for this Reynolds number. Use $\mathcal{P}^2$-$\mathcal{P}^1$ 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 throughout the domain and superimpose a quiver plot that shows the direction of the flow.
- Plot both components of the velocity along the four lines Γ_1, Γ_2, Γ_3, Γ_4 defined in Figure 6.

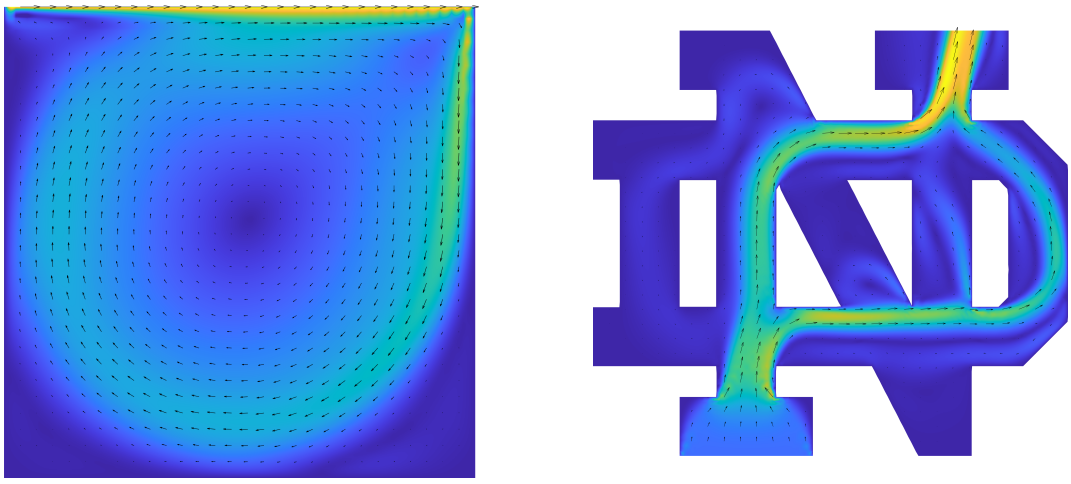

Figure 13: Solution of the incompressible Naiver-Stokes equations: lid-driven cavity flow ($Re = 2000$) and flow through ND logo ($Re = 1300$).