CFD2 assignment 2024-25

December 2024

Abstract

In this assignment, you will be asked to write a Navier-Stokes solver based on Incidence and Mass matrices on arbitrary curvilinear domains. Note that this assignment deviates from the assignments for this course in previous years.

1 Introduction

The goal of this assignment is to solve the 2D incompressible Navier-Stokes equations on arbitrary curvilinear domains. The governing equations of momentum and mass conservation in dimensionless form read as follows:

$$\frac{\partial \underline{u}}{\partial t} + (\underline{u}\cdot\nabla)\underline{u} - \frac{1}{Re}\Delta\underline{u} + \nabla p = \underline{f}, \quad \text{in } \Omega, \tag{1}$$

$$\nabla\cdot\underline{u} = 0, \quad \text{in } \Omega, \tag{2}$$

where $\underline{u}$ is the velocity field, p is the (static) pressure, Re is the Reynolds number, $\Delta\underline{u} = \nabla\cdot\nabla\underline{u}$ is the *Laplacian vector* of $\underline{u}$, and Ω is the geometry on which the problem is posed.

To avoid using second order derivatives, we break (1) in two steps, which leads to two equations with only first-order derivatives. This results in the so-called *mixed formulation.* To end up using two first-order derivatives the vorticity $\underline{\omega} = -\nabla\times\underline{u}$ is introduced and used in the following vector identities:

$$\Delta\underline{u} = \nabla(\nabla\cdot\underline{u}) - \nabla\times\nabla\times\underline{u} = -\nabla\times\nabla\times\underline{u} = \nabla\times\underline{\omega}, \tag{3}$$

$$(\underline{u}\cdot\nabla)\underline{u} = (\nabla\times\underline{u})\times\underline{u} + \frac{1}{2}\nabla(\underline{u}\cdot\underline{u}) = -\underline{\omega}\times\underline{u} + \frac{1}{2}\nabla(\underline{u}\cdot\underline{u}), \tag{4}$$

where we simplify (3) by using $\nabla\cdot\underline{u} = 0$. In this case the Navier-Stokes equations from (1) and (2) are written in the following *mixed formulation*:

$$\underline{\omega} + \nabla\times\underline{u} = 0, \quad \text{in } \Omega, \tag{5}$$

$$\frac{\partial \underline{u}}{\partial t} - \underline{\omega}\times\underline{u} - \frac{1}{Re}\nabla\times\underline{\omega} + \nabla P = \underline{f}, \quad \text{in } \Omega, \tag{6}$$

$$\nabla\cdot\underline{u} = 0, \quad \text{in } \Omega, \tag{7}$$

where $P := p + \frac{1}{2}\underline{u}\cdot\underline{u}$ is the *total* or *Bernoulli pressure.*

This form of the Navier-Stokes equations is only directly applicable for finite Reynolds numbers because of the term $1/Re$. Solving for Stokes (*i.e., $Re = 0$*) therefore requires the definition of a limit case. In that case, the inertia terms become negligible and are left out of the equation:

$$\underline{\omega} + \nabla\times\underline{u} = 0, \quad \text{in } \Omega, \tag{8}$$

$$\nabla\times\underline{\omega} - \nabla\hat{p} = \underline{f}, \quad \text{in } \Omega, \tag{9}$$

$$\nabla\cdot\underline{u} = 0, \quad \text{in } \Omega. \tag{10}$$

In this formulation the pressure is denoted by $\hat{p}$ as this is not the same dimensionless pressure as indicated in (1), since for Stokes the physical pressure must be scaled with a different reference pressure.

2 Case setup

For this assignment, you will simulate a two-dimensional case, the setup of which is illustrated in Figure 1. As shown in the figure, periodic boundary conditions are applied to the left and right sides of the domain. This means that whatever flows outward from the right side re-enters from the left side. Numerically, this couples the elements (and the basis functions) along the right boundary to the elements along the left boundary, just as this is the case over any other grid line.

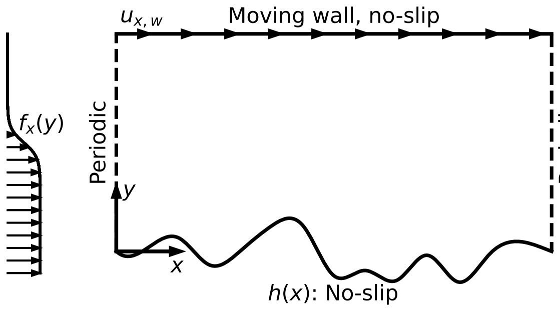

Figure 1: Case definition; the geometry of the lower wall $h(x)$. This geometry will be different for all groups doing this assignment.

At the top and bottom boundaries, no-slip boundary conditions are imposed. The top wall moves to the right with an imposed velocity, $u_{x,w} = 1$, such that along that boundary

$$\underline{u} = \begin{bmatrix} u_{x,w} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

At the bottom wall, a homogeneous no-slip condition is applied

$$\underline{u} = \underline{0}.$$

To compute the Reynolds number, we use the constant top wall velocity as the reference velocity and we take a reference length of 0.3 characterised by the bottom wall geometry. Moreover, the following body force is imposed throughout the domain:

$$\underline{f} = \begin{bmatrix} f_x(y) \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} + \frac{1}{4}\tanh\left(-15\left(y - \frac{1}{2}\right)\right) \\ 0 \end{bmatrix}, \tag{11}$$

its shape is also shown in Fig. 1. Don't think too much about this force term; it will be pre-programmed for you. We use this somewhat complex expression so you get nice flow visualizations later.

You will be provided with two different geometries for the bottom wall:

- For the mesh convergence study you should use $h(x) = 0$ (i.e. a square domain).
- For the other cases you should use the provided geometry, **which depends on your group number.**

Check carefully which lower-wall geometry you should use for which case, failure to do so will result in the wrong solution!

Task 1: Given these boundary conditions and what you have learned in the course, do you think this problem is well-posed and has a unique solution?

3 Construction of Incidence matrices

To begin, we have to set up the discrete system, by constructing the incidence matrices. When solving a formulation like this in two dimensions, two incidence matrices can be defined in the primal mesh, $\mathbb{E}^{1,0}$ and $\mathbb{E}^{2,1}$. A two-dimensional hybrid primal mesh is shown in Figure 2. Specifically, Figure 2 illustrates an outer-oriented mesh where the degrees of freedom are numbered in lexicographic order. This means the numbering starts from left to right along the x-direction and continues upward in the y-direction. This numbering determines the order of the degrees-of-freedom in the global system and is crucial to assemble the matrices, both the topological ones and the"connecting" one, that determine the way in which individual elements are connected and constitute a global mesh.

Since the mesh that we use is a hybrid one, we solve each element individually and then combine these solutions. Each element is considered part of the global mesh and connected to its neighboring elements. To "glue" the elements in the global system Lagrange multipliers are used: the λ parameters for the velocity fluxes and γ parameters for the vorticity. To provide a simple example, consider the first element (bottom left) in Figure 2:

- This element needs to be "glued" to the one on its right, meaning that the vorticity degrees of freedom ω_2, ω_5 and ω_8 should be set equal to ω_9, ω_{12} and ω_{15}, respectively. Additionally, the outgoing velocity fluxes u_2 and u_5 should be equal to the ingoing velocity fluxes u_{12} and u_{15} in the element to the right.
- The element under consideration should also be "glued" to the element above it. Following the same logic as before, the vorticity degrees of freedom ω_6, ω_7 and ω_8 should be equated to ω_{27}, ω_{28} and ω_{29}, while the velocity fluxes u_{10} and u_{11} should be set equal to u_{42} and u_{43}.

The focus of this assignment is on the construction of the incidence matrices. The part related to this glueing of the elements has already been done for you. As such, your main focus can be the local numbering, which

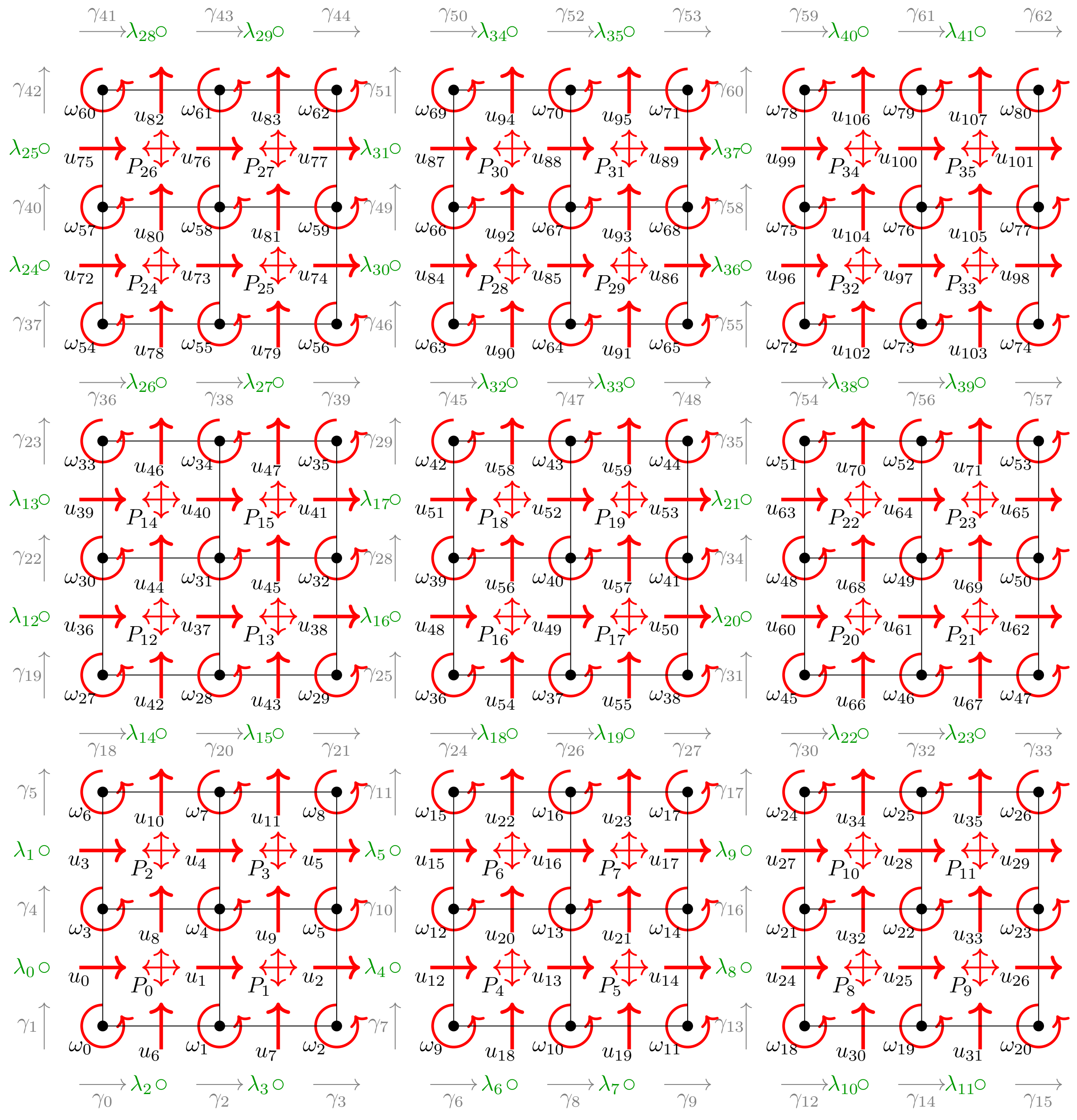

Figure 2: Example hybrid 2D mesh with $p = 2$ and $M = 3 \times N = 3$ elements

plays a vital role in the construction of the incidence matrices. In Figure 3 you can see the numbering of a local element, corresponding to a polynomial degree of $p = 2$.

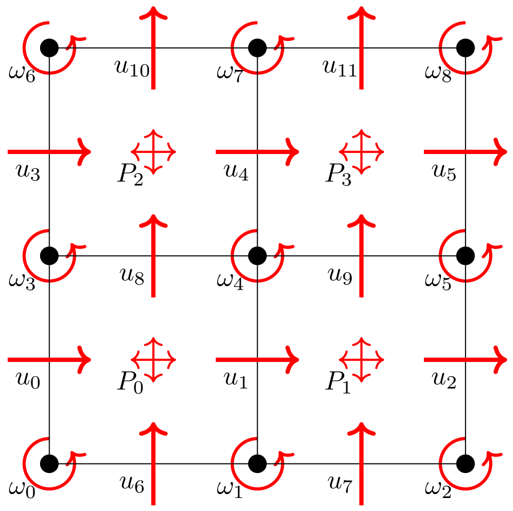

Figure 3: Single element of the mesh for polynomial degree $p = 2$

Task 2: Set up the incidence matrices $\mathbb{E}^{1,0}$ and $\mathbb{E}^{2,1}$ for one element using the geometric approach discussed in the lectures. Start with the $p = 2$ case shown in Figure 3, then generalize for arbitrary p. Verify your implementation by checking if the following identity

$$\mathbb{E}^{2,1}\mathbb{E}^{1,0} = 0. \tag{12}$$

The implementation of the incidence matrices should be made in the file "incidenceMatrices.py". In that file you should find the two functions `assemble_element_E10(p)` and `assemble_element_E21(p)`. In this file, comment out the "raise NotImplementedError" line and add your own implementation for $\mathbb{E}^{1,0}$ and $\mathbb{E}^{2,1}$. ***NOTE***: As these two functions are called and used in the main code with these exact names, the file name and the function names must **not** be altered. However, you are free to add additional functions to the file to aid your implementation.

4 Construction of mass (Hodge) matrices

Following the implementation of the incidence matrices, the next step involves the set up of the mass matrices. These should be implemented in the file named "massMatrices.py". In this file you should find the following functions; `assemble_element_M0()`, `assemble_element_M1()`, and `assemble_element_M2()`.

NOTE: We remind you again, that the file name and the function names should **not** be changed.

The mass matrices will be assembled element by element. For this assignment, it is requested that you implement the algorithm for the mass matrices of ***one individual element*** of general polynomial degree p in both the x and y directions. Subsequently, these functions will be called in a different part of the code to construct the global mass matrices.

Since we want to solve our PDEs in curvilinear domains, we establish a mapping between our computational or reference domain $[-1, 1]^2$ and the physical domain Ω. This mapping is provided for you in the assignment. We do all our computations on the computational domain by pulling back the operations from the physical domain. A key ingredient related to the mapping is the Jacobian which was introduced in the lectures as follows

$$\mathbf{F} := \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}. \tag{13}$$

Note that since we have a curvilinear grid, the components of the Jacobian are actually functions of ξ and η which denote the coordinates in the reference domain $[-1, 1]^2$ (they are not constants). Hence, they need to be evaluated at the Gauss-Lobatto integration points and included in the integral, as discussed in the lectures.

We will start with $\mathbb{M}^{(0)}$ and $\mathbb{M}^{(2)}$ which are defined as follows:

$$\mathbb{M}_{ij}^{(0)} := \int_{\Omega} \psi_i^{(0)}(x,y)\psi_j^{(0)}(x,y)\,\mathrm{d}\Omega = \int_{-1}^{1}\int_{-1}^{1} \psi_i^{(0)}(\xi,\eta)\psi_j^{(0)}(\xi,\eta)\det(\mathbf{F})\,\mathrm{d}\xi\mathrm{d}\eta \tag{14}$$

$$\mathbb{M}_{ij}^{(2)} := \int_{\Omega} \psi_i^{(2)}(x,y)\psi_j^{(2)}(x,y)\,\mathrm{d}\Omega = \int_{-1}^{1}\int_{-1}^{1} \psi_i^{(2)}(\xi,\eta)\psi_j^{(2)}(\xi,\eta)\frac{1}{\det(\mathbf{F})}\,\mathrm{d}\xi\mathrm{d}\eta, \tag{15}$$

where $\psi_i^{(0)}(\xi,\eta)$ and $\psi_i^{(2)}(\xi,\eta)$ are the zero and two form basis functions respectively in the reference domain. Furthermore, $\mathbb{M}^{(1)}$ is defined as follows:

$$\mathbb{M}^{(1)} := \int_{\Omega} \left[\psi_x^{(1)}(x,y), \psi_y^{(1)}(x,y)\right] \begin{bmatrix} \psi_x^{(1)}(x,y) \\ \psi_y^{(1)}(x,y) \end{bmatrix} \mathrm{d}\Omega \tag{16}$$

$$= \int_{-1}^{1}\int_{-1}^{1} \left[\psi_\xi^{(1)}(\xi,\eta), \psi_\eta^{(1)}(\xi,\eta)\right] \mathbf{F}^T\mathbf{F} \begin{bmatrix} \psi_\xi^{(1)}(\xi,\eta) \\ \psi_\eta^{(1)}(\xi,\eta) \end{bmatrix} \frac{1}{\det(\mathbf{F})}\mathrm{d}\xi\mathrm{d}\eta \tag{17}$$

You can construct this matrix by combining four sub-matrices in the following way (for instance with the `numpy.block()` function):

$$\mathbb{M}^{(1)} = \begin{bmatrix} \mathbb{M}^{(\xi,\xi)} & \mathbb{M}^{(\xi,\eta)} \\ \mathbb{M}^{(\eta,\xi)} & \mathbb{M}^{(\eta,\eta)} \end{bmatrix} \tag{18}$$

These four sub-matrices are defined as follows:

$$\mathbb{M}_{ij}^{(\xi,\xi)} = \int_{-1}^{1}\int_{-1}^{1} \psi_{\xi,i}^{(1)}(\xi,\eta)[\mathbf{F}^T\mathbf{F}]_{0,0}\psi_{\xi,j}^{(1)}(\xi,\eta)\frac{1}{\det(\mathbf{F})}\mathrm{d}\xi\mathrm{d}\eta, \tag{19}$$

$$\mathbb{M}_{ij}^{(\xi,\eta)} = \int_{-1}^{1}\int_{-1}^{1} \psi_{\xi,i}^{(1)}(\xi,\eta)[\mathbf{F}^T\mathbf{F}]_{0,1}\psi_{\eta,j}^{(1)}(\xi,\eta)\frac{1}{\det(\mathbf{F})}\mathrm{d}\xi\mathrm{d}\eta, \tag{20}$$

$$\mathbb{M}_{ij}^{(\eta,\xi)} = \int_{-1}^{1}\int_{-1}^{1} \psi_{\eta,i}^{(1)}(\xi,\eta)[\mathbf{F}^T\mathbf{F}]_{1,0}\psi_{\xi,j}^{(1)}(\xi,\eta)\frac{1}{\det(\mathbf{F})}\mathrm{d}\xi\mathrm{d}\eta, \tag{21}$$

$$\mathbb{M}_{ij}^{(\eta,\eta)} = \int_{-1}^{1}\int_{-1}^{1} \psi_{\eta,i}^{(1)}(\xi,\eta)[\mathbf{F}^T\mathbf{F}]_{1,1}\psi_{\eta,j}^{(1)}(\xi,\eta)\frac{1}{\det(\mathbf{F})}\mathrm{d}\xi\mathrm{d}\eta, \tag{22}$$

where $\psi_i^{(1)}(\xi,\eta)$ is the one form basis function and $\left[\mathbf{F}^T\mathbf{F}\right]_{[n,m]}$ is the nth row mth column of the 2×2 $\mathbf{F}^T\mathbf{F}$ matrix. In the code the following indications are used: $\psi^{(0)}(\xi,\eta) =$ `psi0_i`, $\psi_\xi^{(1)}(\xi,\eta) =$ `psi1_xi_i`, $\psi_\eta^{(1)}(\xi,\eta) =$ `psi1_eta_i` and $\psi^{(2)}(\xi,\eta) =$ `psi2_i`. These variables are 3-dimensional arrays, please keep note of what each axis represents:

Axis 0 represents the different basis functions.

Axis 1 represents the increment in η-direction.

Axis 2 represents the increment in ξ-direction.

Thus, each value `psi0_i[k,m,n]` in the array represents the basis function $\psi_k^{(0)}$ evaluated at the point (ξ_n, η_m). These 3-dimensional arrays are also visually represented in Eqs. 23-25:

$$\psi^{(0)}(\xi,\eta) = \left[\begin{array}{c}
\left.\begin{bmatrix} h_0(\xi_0)h_0(\eta_0) & h_0(\xi_1)h_0(\eta_0) & \dots & h_0(\xi_p)h_0(\eta_0) \\ h_0(\xi_0)h_0(\eta_1) & h_0(\xi_1)h_0(\eta_1) & \dots & h_0(\xi_p)h_0(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_0(\xi_0)h_0(\eta_p) & h_0(\xi_1)h_0(\eta_p) & \dots & h_0(\xi_p)h_0(\eta_p) \end{bmatrix}\right\} \psi_0^{(0)}(\xi,\eta) \\
\left.\begin{bmatrix} h_1(\xi_0)h_0(\eta_0) & h_1(\xi_1)h_0(\eta_0) & \dots & h_1(\xi_p)h_0(\eta_0) \\ h_1(\xi_0)h_0(\eta_1) & h_1(\xi_1)h_0(\eta_1) & \dots & h_1(\xi_p)h_0(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_1(\xi_0)h_0(\eta_p) & h_1(\xi_1)h_0(\eta_p) & \dots & h_1(\xi_p)h_0(\eta_p) \end{bmatrix}\right\} \psi_1^{(0)}(\xi,\eta) \\
\vdots \\
\left.\begin{bmatrix} h_p(\xi_0)h_p(\eta_0) & h_p(\xi_1)h_p(\eta_0) & \dots & h_p(\xi_p)h_p(\eta_0) \\ h_p(\xi_0)h_p(\eta_1) & h_p(\xi_1)h_p(\eta_1) & \dots & h_p(\xi_p)h_p(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_p(\xi_0)h_p(\eta_p) & h_p(\xi_1)h_p(\eta_p) & \dots & h_p(\xi_p)h_p(\eta_p) \end{bmatrix}\right\} \psi_{(p+1)(p+1)}^{(0)}(\xi,\eta)
\end{array}\right] \tag{23}$$

$$\psi_\xi^{(1)}(\xi,\eta) = \left[\begin{array}{c}
\left.\begin{bmatrix} h_0(\xi_0)e_1(\eta_0) & h_0(\xi_1)e_1(\eta_0) & \dots & h_0(\xi_p)e_1(\eta_0) \\ h_0(\xi_0)e_1(\eta_1) & h_0(\xi_1)e_1(\eta_1) & \dots & h_0(\xi_p)e_1(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_0(\xi_0)e_1(\eta_p) & h_0(\xi_1)e_1(\eta_p) & \dots & h_0(\xi_p)e_1(\eta_p) \end{bmatrix}\right\} \psi_{\xi,0}^{(1)}(\xi,\eta) \\
\left.\begin{bmatrix} h_1(\xi_0)e_1(\eta_0) & h_1(\xi_1)e_1(\eta_0) & \dots & h_1(\xi_p)e_1(\eta_0) \\ h_1(\xi_0)e_1(\eta_1) & h_1(\xi_1)e_1(\eta_1) & \dots & h_1(\xi_p)e_1(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_1(\xi_0)e_1(\eta_p) & h_1(\xi_1)e_1(\eta_p) & \dots & h_1(\xi_p)e_1(\eta_p) \end{bmatrix}\right\} \psi_{\xi,1}^{(1)}(\xi,\eta) \\
\vdots \\
\left.\begin{bmatrix} h_p(\xi_0)e_p(\eta_0) & h_p(\xi_1)e_p(\eta_0) & \dots & h_p(\xi_p)e_p(\eta_0) \\ h_p(\xi_0)e_p(\eta_1) & h_p(\xi_1)e_p(\eta_1) & \dots & h_p(\xi_p)e_p(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ h_p(\xi_0)e_p(\eta_p) & h_p(\xi_1)e_p(\eta_p) & \dots & h_p(\xi_p)e_p(\eta_p) \end{bmatrix}\right\} \psi_{\xi,p(p+1)}^{(1)}(\xi,\eta)
\end{array}\right] \tag{24}$$

$$\psi_\eta^{(1)}(\xi,\eta) = \left[\begin{array}{c}
\left.\begin{bmatrix} e_1(\xi_0)h_0(\eta_0) & e_1(\xi_1)h_0(\eta_0) & \dots & e_1(\xi_p)h_0(\eta_0) \\ e_1(\xi_0)h_0(\eta_1) & e_1(\xi_1)h_0(\eta_1) & \dots & e_1(\xi_p)h_0(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ e_1(\xi_0)h_0(\eta_p) & e_1(\xi_1)h_0(\eta_p) & \dots & e_1(\xi_p)h_0(\eta_p) \end{bmatrix}\right\} \psi_{\eta,0}^{(1)}(\xi,\eta) \\
\left.\begin{bmatrix} e_2(\xi_0)h_0(\eta_0) & e_2(\xi_1)h_0(\eta_0) & \dots & e_2(\xi_p)h_0(\eta_0) \\ e_2(\xi_0)h_0(\eta_1) & e_2(\xi_1)h_0(\eta_1) & \dots & e_2(\xi_p)h_0(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ e_2(\xi_0)h_0(\eta_p) & e_2(\xi_1)h_0(\eta_p) & \dots & e_2(\xi_p)h_0(\eta_p) \end{bmatrix}\right\} \psi_{\eta,1}^{(1)}(\xi,\eta) \\
\vdots \\
\left.\begin{bmatrix} e_p(\xi_0)h_p(\eta_0) & e_p(\xi_1)h_p(\eta_0) & \dots & e_p(\xi_p)h_p(\eta_0) \\ e_p(\xi_0)h_p(\eta_1) & e_p(\xi_1)h_p(\eta_1) & \dots & e_p(\xi_p)h_p(\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ e_p(\xi_0)h_p(\eta_p) & e_p(\xi_1)h_p(\eta_p) & \dots & e_p(\xi_p)h_p(\eta_p) \end{bmatrix}\right\} \psi_{\eta,p(p+1)}^{(0)}(\xi,\eta)
\end{array}\right] \tag{25}$$

$\psi^{(2)}(\xi,\eta)$ is stored in the same manner where the basis functions are constructed with a tensor product of edge functions $(e(\xi)e(\eta))$.

Next, the determinant of the Jacobian and its components are given in the code as 2-dimensional array with the following axis representation:

Axis 0 represents the increment in η-direction.

Axis 1 represents the increment in ξ-direction.

Thus, each value of e.g. `det_F` is the determinant of $\mathbf{F}$ evaluated at the point (ξ_n, η_m):

$$\det(\mathbf{F})(\xi,\eta) = \begin{bmatrix} \det(\mathbf{F})(\xi_0,\eta_0) & \det(\mathbf{F})(\xi_1,\eta_0) & \dots & \det(\mathbf{F})(\xi_p,\eta_0) \\ \det(\mathbf{F})(\xi_0,\eta_1) & \det(\mathbf{F})(\xi_1,\eta_1) & \dots & \det(\mathbf{F})(\xi_p,\eta_1) \\ \vdots & \vdots & \vdots & \vdots \\ \det(\mathbf{F})(\xi_0,\eta_p) & \det(\mathbf{F})(\xi_1,\eta_p) & \dots & \det(\mathbf{F})(\xi_p,\eta_p) \end{bmatrix} \tag{26}$$

Lastly, the Gauss-Lobatto integration weights are provided as two 1D arrays corresponding to the weights in the ξ and η direction.

Task 3: Use the definitions provided above to implement the three mass matrices in "massMatrices.py".

5 Stokes Flow

For the case of the mixed formulation of the Stokes flow, before enforcing boundary conditions, the weak form can be written as:

Find $\underline{\omega} \in H(\text{curl}, \Omega)$, $\underline{u} \in H(\text{div}, \Omega)$ and $\hat{p} \in L^2(\Omega)$ such that for all $\underline{\zeta} \in H(\text{curl}, \Omega)$, $\underline{v} \in H(\text{div}, \Omega)$ and $\hat{q} \in L^2(\Omega)$ it holds that:

$$\int_\Omega \underline{\zeta} \cdot \underline{\omega} \mathrm{d}\Omega + \int_\Omega \underline{u} \cdot (\nabla \times \underline{\zeta}) \mathrm{d}\Omega = \int_{\partial\Omega} \underline{u} \cdot (\underline{n} \times \underline{\zeta}) \mathrm{d}\Gamma = \int_{\partial\Omega} \underline{u}_t \cdot (\underline{n} \times \underline{\zeta}_t) \mathrm{d}\Gamma, \tag{27}$$

$$\int_\Omega \underline{v} \cdot (\nabla \times \underline{\omega}) \,\mathrm{d}\Omega + \int_\Omega p \nabla \cdot \underline{v} \mathrm{d}\Omega = \int_{\partial\Omega} \underline{n} \cdot \underline{v} p \mathrm{d}\Gamma - \int_\Omega \underline{v} \cdot \underline{f} \mathrm{d}\Omega = \int_{\partial\Omega} v_n p \mathrm{d}\Gamma - \int_\Omega \underline{v} \cdot \underline{f} \mathrm{d}\Omega, \tag{28}$$

$$\int_\Omega q \nabla \cdot \underline{u} \mathrm{d}\Omega = 0. \tag{29}$$

In this formulation $\underline{\zeta}_t$ and $\underline{u}_t$ denote the tangential components of the test function $\underline{\zeta}$ and the solution $\underline{u}$, and $v_n = \underline{n} \cdot \underline{v}$ denotes the normal component of testfunction $\underline{v}$.

Task 4: Verify that the last relation of (27) holds, since the normal components do not contribute to the boundary integral.

Task 5: In your report, discuss how the normal and tangential velocities are prescribed on the boundary. Explain which component(s) are weakly prescribed and which are strongly prescribed, and how this will affect the discrete (*i.e.*, finite element) function spaces.

After discretization with basis functions, the resulting system of equations becomes:

$$\begin{bmatrix} \mathbb{M}^{(0)} & \mathbb{E}^{1,0^T}\mathbb{M}^{(1)} & \varnothing \\ \mathbb{M}^{(1)}\mathbb{E}^{1,0} & \varnothing & \mathbb{E}^{2,1^T}\mathbb{M}^{(2)} \\ \varnothing & \mathbb{M}^{(2)}\mathbb{E}^{2,1} & \varnothing \end{bmatrix} \begin{bmatrix} \bar{\underline{\omega}} \\ \bar{\underline{u}} \\ \bar{p} \end{bmatrix} = \begin{bmatrix} \mathcal{B}_t(h) \\ \underline{f} \\ \mathbf{0} \end{bmatrix}, \tag{30}$$

which is the system that is implemented in the provided code.

5.1 Mesh convergence study with method of manufactured solutions

If you correctly followed the previous steps, you should now have a working code to solve the Stokes equation. However, mistakes can and will sneak into all computer codes. If you are lucky, you get an error. If you are unlucky, the code runs despite the mistake and may even produce reasonable looking results (which are completely wrong). For this reason, **code verification** is very important when writing scientific codes.

A verification test often used for PDE solvers is a mesh convergence study to obtain the method's **order of convergence**. In a mesh convergence study, the same case is run on a set of increasingly refined meshes. Typically, the side length h of each cell is decreased by a factor two in each refinement step, this is illustrated in Figure 4. Note that refining the mesh by a factor two results in four times more cells in 2D (and eight times more in 3D). For each mesh, a global error norm is computed over the **difference** between the **numerical** and the **exact** solution in the cells.

Since different function spaces are considered, different norms can be calculated. In the tasks below, specific norms need to be discussed. These are defined as follows (note that for $L^2(\Omega)$ the function f can represent both a scalar and a vector):

$$\|f\|^2_{L^2(\Omega)} = \int_\Omega |f|^2 \mathrm{d}\Omega, \tag{31}$$

$$\|\underline{f}\|^2_{H(\mathrm{curl},\Omega)} = \|\underline{f}\|^2_{L^2(\Omega)} + \|\nabla \times \underline{f}\|^2_{L^2(\Omega)} = \int_\Omega \left[|\underline{f}|^2 + \left|\nabla \times \underline{f}\right|^2 \right] \mathrm{d}\Omega. \tag{32}$$

$$\|\underline{f}\|^2_{H(\mathrm{div},\Omega)} = \|\underline{f}\|^2_{L^2(\Omega)} + \|\nabla \cdot \underline{f}\|^2_{L^2(\Omega)} = \int_\Omega \left[|\underline{f}|^2 + \left(\nabla \cdot \underline{f}\right)^2 \right] \mathrm{d}\Omega, \tag{33}$$

The mesh convergence study is presented by plotting the global error norm against $1/h$ in a loglog-plot, the result will look something like Figure 5. The slope in this graph is an interesting measure: it indicates how much the error decreases when refining the mesh. On the left side of Figure 5 (coarse meshes), the slope varies. For fine meshes ($1/h > 10^1$) on the other hand, the slope is constant, we call this the *asymptotic region.* **Definition:** the **order of convergence** is the slope observed in the **asymptotic region** of a mesh convergence study plot. Thus, to find the order of convergence, you have to use a number of meshes to make sure you compute the order of convergence in the asymptotic region. In the code, the computation of the order of convergence is already implemented.

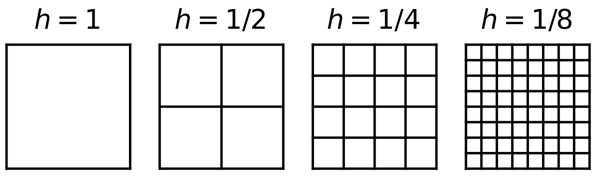

Figure 4: Illustration of mesh refinement, the mesh is refined by a factor two for each subsequent mesh.

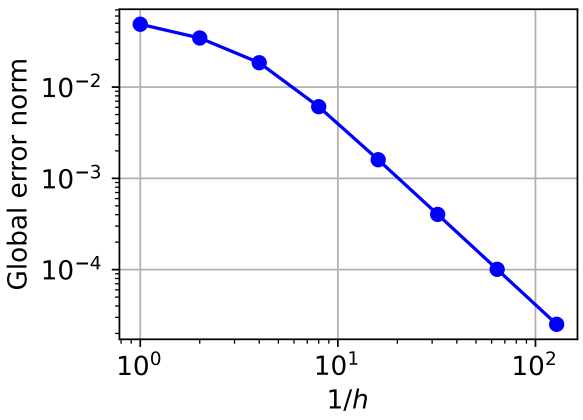

Figure 5: Example of mesh convergence study.

One thing we have not addressed yet, is how to obtain an exact solution. For typical problems, the exact solution is not available (if it were, you would not need to write a numerical solver). However, you can also take the inverse approach: start from a manufactured (chosen) solution and find which problem produces this solution, this is known as the **method of manufactured solutions**. For the chosen solution, you only need to make sure that it satisfies the boundary conditions. To make sure that your chosen solution satisfies the problem formulated in the equation, you can add source terms to the right-hand-side (we show an example in the next paragraph). For the solution that we will choose later on, it already satisfies the first (vorticity) and third (continuity) equation such that only a force term needs to be added to the second (momentum) equation.

In order to better understand the method of manufactured solutions, consider the example of the following simple ODE: $\frac{\partial^2 u}{\partial x^2} = f(x)$ on the domain $[0, 1]$ and with $u(0) = 0$ and $u(1) = 1$. One solution which satisfies these boundary conditions is $u(x) = \sin(\frac{\pi}{2}x)$. Plugging this into the ODE gives $f(x) = -\frac{\pi^2}{4}\sin(\frac{\pi}{2}x)$. Thus, if you have programmed a solver for this ODE, you can run a mesh convergence study by setting $f(x) = -\frac{\pi^2}{4}\sin(\frac{\pi}{2}x)$ such that you know the exact solution is $u(x) = \sin(\frac{\pi}{2}x)$.

Now let's apply the method of manufactured solution to the problem at hand (Stokes flow). We already give you the manufactured solution:

$$u_{x,exact}(x,y) = y + \cos(2\pi x)\sin(2\pi y), \tag{34}$$
$$u_{y,exact}(x,y) = \sin(2\pi x)(1 - \cos(2\pi y)), \tag{35}$$
$$p_{exact}(x,y) = \sin(2\pi x)\cos(2\pi y). \tag{36}$$

Note: this manufactured solution is **only valid for the square domain** (i.e. $h(x) = 0$ in Figure 1). Hence, for the convergence study you should use the square domain, this is already set up in the relevant file (`convergenceTest.py`), please don't change it. Also, while the domain is square, we deformed the elements inside such that you can properly verify the mappings in your mass matrices, the mesh is also shown in Figure 6.

Task 6: Show that the manufactured solution in (eqs. (34) and (36)) satisfies the previously provided boundary conditions in the case setup (i.e., $\underline{u} = \underline{0}$ at $y = 0$; $\underline{u} = [1, 0]^T$ at $y = 1$).

Task 7: Show that the manufactured solution in (eqs. (34) and (36)) satisfies the continuity equation ($\nabla \cdot \underline{u} = 0$).

Task 8: Derive the force term in (9) associated with the manufactured solution in (eqs. (34) and (36)).

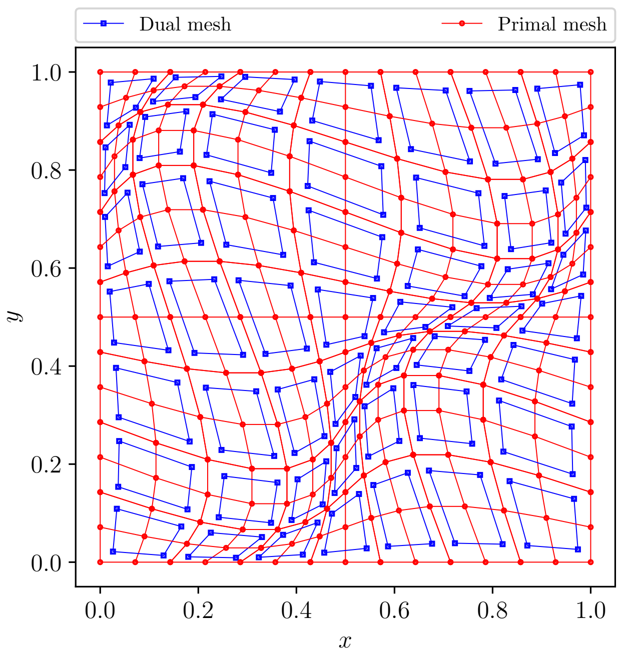

Figure 6: Example of the mesh on the square domain used in the convergence study.

Now you can run your own mesh convergence study and determine the order of convergence. Most steps to perform the mesh convergence study are already pre-programmed. This study is carried out in the Python file titled `convergenceTest.py` attached with the assignment. We ask you to take the following steps:

Task 9: Implement the forcing term for Stokes you derived in Task 8. You do this by editing the following lines in `convergenceTest.py`

```
## ========================================================================= ##
## +++++++++++++++++++ Fill in Source term of PDE ++++++++++++++ ##
## +++++++++++ corresponding to the manufactured solution +++++++ ##
## ++++++++++++++++++++++ in the lines below +++++++++++++++++++ ##
## ========================================================================= ##
fx = lambda x, y: # Add x-component of the forcing term as a function of x and y
fy = lambda x, y: # Add y-component of the forcing term as a function of x and y
## ========================================================================= ##
## ========================================================================= ##
```

Task 10: Run the Python file `convergenceTest.py`, this should run the Stokes flow case with the supplied forcing term and output 5 convergence plots. Present the convergence plots outputted by the code in you report.

Task 11: Consider the error curves for the velocity in L^2. Describe the effect of increasing the polynomial degree p. Can you explain your observations?

Task 12: Compare the convergence rate of the velocity in the $H(\text{div}, \Omega)$ norm and the $L^2(\Omega)$ norm. Explain your observations.

Task 13: Compare the convergence rate of the vorticity in the $H(\text{curl},\Omega)$ norm and the $L^2(\Omega)$ norm. Explain your observations.

Task 14: Consider the convergence of the pressure in the $L^2(\Omega)$ norm. Do you notice something strange here? Can you explain this behaviour?

Task 15: (BONUS) If you look inside the code, you can see that the number of quadrature points for the integration of the mass matrices is not the same as the integration to obtain the error norms. Can you explain why this was implemented like this?

5.2 Contour plots

Now that you have verified your implementation, we ask you to run `stokesFlow2D.py` and report the solution you obtain. Note: for this case you should **use the curved bottom wall** (the exact shape depends on your group number). the correct shape is automatically used in `stokesFlow2D.py`, please don't change this. For the mesh use $p = 4$, $N = 16$, $M = 8$ and complete the following tasks:

Task 16: Show the contour plot of the vorticity corresponding to the isolines [0., 0.2, 0.4, 0.6, 0.8, 1., 1.2, 1.4, 1.6, 1.8, 2., 2.2, 2.4, 2.6, 2.8, 3., 3.2, 3.4, 3.6, 3.8, 4.] (this is already programmed).

Task 17: Show the contour plot of the streamlines and a plot of the pointwise divergence (you can use functions like plt.pcolor()).

6 Navier-Stokes

With Navier-Stokes, the momentum equation is altered and becomes time-dependent. We treat the pressure term in this equation implicitly, while the other terms are treated with a midpoint rule (*i.e.,* a Crank-Nicolson scheme). The weak formulation therefore becomes:

$$\begin{aligned}\int_\Omega \underline{v}\cdot\frac{\underline{u}^{n+1}-\underline{u}^n}{\Delta t}\mathrm{d}\Omega - \frac{1}{Re}\int_\Omega \underline{v}\cdot\left(\nabla\times\underline{\omega}^{n+\frac{1}{2}}\right)\mathrm{d}\Omega - \int_\Omega p^{n+1}\nabla\cdot\underline{v}\mathrm{d}\Omega \\ = \int_\Omega \underline{v}\cdot\underline{f}^{n+\frac{1}{2}}\mathrm{d}\Omega + \int_\Omega \underline{v}\cdot\underline{\omega}^{n+\frac{1}{2}}\times\underline{u}^{n+\frac{1}{2}}\mathrm{d}\Omega.\end{aligned} \tag{37}$$

The first and third line of the weak form (*i.e.,* the vorticity relation and the mass balance) are both enforced implicitly. Due to the nonlinear convective term, a nonlinear system of equations is obtained. To enable a direct solution, the nonlinear term is approximated using a predictor-corrector approach in the code, such that the convective term can be fully incorporated in the right-hand-side:

$$\underline{\omega}^{n+\frac{1}{2}}\times\underline{u}^{n+\frac{1}{2}} \approx \frac{1}{4}\left(\widetilde{\underline{\omega}^{n+1}}+\underline{\omega}^n\right)\times\left(\widetilde{\underline{u}^{n+1}}+\underline{u}^n\right). \tag{38}$$

The final system of equations then becomes[1]:

$$\begin{bmatrix} \mathbb{M}^{(0)} & \mathbb{E}^{1,0^T}\mathbb{M}^{(1)} & \varnothing \\ \frac{-1}{2Re}\mathbb{M}^{(1)}\mathbb{E}^{1,0} & \frac{1}{\Delta t}\mathbb{M}^{(1)} & -\mathbb{E}^{2,1^T}\mathbb{M}^{(2)} \\ \varnothing & \mathbb{M}^{(2)}\mathbb{E}^{2,1} & \varnothing \end{bmatrix} \begin{bmatrix} \underline{\bar{\omega}}^{n+1} \\ \underline{\bar{u}}^{n+1} \\ \bar{P}^{n+1} \end{bmatrix} = \begin{bmatrix} \mathcal{B}_t(h) \\ \underline{\bar{f}}^{n+\frac{1}{2}} + \frac{1}{\Delta t}\mathbb{M}^{(1)}\underline{\bar{u}}^n + \frac{1}{2Re}\mathbb{M}^{(1)}\mathbb{E}^{1,0}\underline{\bar{\omega}}^n + \underline{\mathcal{C}}(\underline{\bar{\omega}}^{n+\frac{1}{2}},\underline{\bar{u}}^{n+\frac{1}{2}}) \\ \mathbf{0} \end{bmatrix}, \tag{39}$$

with the predictor-corrector approach providing the term

$$\underline{\mathcal{C}}(\underline{\bar{\omega}}^{n+\frac{1}{2}},\underline{\bar{u}}^{n+\frac{1}{2}}) := \frac{1}{4}\int_\Omega \underline{v}\cdot((\widetilde{\underline{\bar{\omega}}^{n+1}}+\underline{\bar{\omega}}^n)\times(\widetilde{\underline{\bar{u}}^{n+1}}+\underline{\bar{u}}^n))\mathrm{d}\Omega. \tag{40}$$

6.1 Contour plots

We now ask you to run the Navier-Stokes code (`navierStokes2D.py`) and analyse the results. Note: for this case you should **use the curved bottom wall** (the exact shape depends on your group number). the correct shape is automatically used in `navierStokes2D.py`, please don't change this. For the mesh use $p = 4$, $N = 16$, $M = 8$ and complete the following tasks:

Task 18: Open the Python code `navierStokes2D.py` and to **set the Reynolds number to 40**. Run the Navier-Stokes case for 20,000 (`Ndt = 20001`) time steps with a time step of 1×10^{-3} with this Reynolds number and report the final value of `max((u^{n + 1} - u^{n})/dt)` outputted by the code.

[1]Note that the discrete system implemented in the code is slightly reformulated for the convenience of implementation. However, the systems are algebraically equivalent and does not affect the analysis you need to perform for the assignment.

Task 19: Show the contour plot of the vorticity at the final time level corresponding to the isolines between -30 and 30 with steps of 2 (this is already programmed). Contrast your observations with the solutions for Stokes flow.

Task 20: Show the contour plot of the streamlines and a plot of the pointwise divergence (you can use functions like plt.pcolor()) at the final time level. Contrast your observations with the solutions for Stokes flow.

Task 21: Now **set the Reynolds number to 400**. Run the Navier-Stokes case for 20,000 time steps (`Ndt = 20001`) with a time step of 1×10^{-3} with this Reynolds number and report the final value of `max((u^{n + 1} - u^{n})/dt)` outputted by the code, comment on the differences with the output of the $Re = 40$ case.

Task 22: Show the contour plot of the vorticity corresponding to the isolines between -30 and 30 with steps of 2 (this is already programmed). Contrast your observations with the solutions for the $Re = 40$ case. You can show the plots for only the final time level.

Task 23: Show the contour plot of the streamlines and a plot of the pointwise divergence (you can use functions like plt.pcolor()). Contrast your observations with the solutions for the $Re = 40$ case.

Task 24: For many applications, simulations with Reynolds number that are much higher are required. Do you think the current simulation setup can correctly handle those (you can try it out yourself)? What aspects of the implementation affect the range of Reynolds numbers that can be reached?

Task 25: In the investigation of the Stokes simulation, you have observed that different mesh sizes and polynomial orders result in different accuracies of the solution. Now perform the integral of the vorticity over the domain for simulations of Navier-Stokes with different mesh sizes. Can you explain your observations? The integral of the vorticity is computed by the following lines in the `navierStokes2D.py` file

```
# Compute integral of vorticity over the domain
integral_omega = msh.W_hiDOP@psi0_plot@omega[:, 1:];
```

`integral_omega` will be a 1D array containing the integrated value of the vorticity at each time step.

Task 26: Plot of the velocity profile of the x-component of the velocity as a function of y for $x = [0.5, 1, 1.5]$. Is the no-slip condition at the wall satisfied?

Write a report of **maximum 15 pages** in which you explain your implementation of the incidence and mass matrices and in which you explain your answers to the tasks. Write the report in a structured format (introduction, methodology, results+discussion, conclusion), integrate the answers to the tasks into this text rather than answering on a per-task basis.