Incompressible fluid simulation using the finite element method

Bachelor Semester Project S5 (Academic Year 2024/25), University of Luxembourg

Benjamin Vogel
benjamin.vogel.001@student.uni.lu
University of Luxembourg
Belval, Esch-sur-Alzette
Luxembourg

Dr. Ezhilmathi Krishnasamy
ezhilmathi.krishnasamy@uni.lu
Belval, Esch-sur-Alzette
Luxembourg

Prof. Pascal Bouvry
Belval, Esch-sur-Alzette
Luxembourg

ABSTRACT

Computational Fluid Dynamics is a vast field of research intersection many domains such as fluid mechanics, numerical methods, and high performance computing. Hence, the goal of this report is to introduce how CFD problems can be defined, present a strategy for solving such a problem and other ideas linked to CFD. Specifically, this report presents how to define fluid flow problems, how the finite element method works, as well as some important considerations on the finite element method such as the choice of finite elements and the trade-off between iterative and direct methods.

1 INTRODUCTION

The domain of computational fluid dynamics (CFD) concerns itself with the analysis of systems that incorporate the flow of fluids, transfer of heat, or chemical reactions through computer simulation. These computer simulations are powerful tools for the testing, experimentation, and analysis of such systems. Notably, such computational testing and analysis is often more cost-effective than physical testing and provides a faster way to test new ideas. Some areas of application of CFD would be simulations of systems such as the aerodynamics of planes or cars, the flow of liquids or gas through pipes, the evolution of temperature of electronic devices, and even the weather.

This bachelor semester project delves into precisely this domain of CFD and Simulation Engineering and tackles the problem of how to model the properties and movement of a fluid and how to solve the time-dependent system of equations governing the fluids behaviour. In other words, this project describes two stages frequently encountered in computational simulations: Modelling and solving the problem. To elaborate, the project report first covers the topics on problem definition in the context of CFD, the governing equations for a fluid, and the modelling of the boundary conditions. Furthermore, the report gives an explanation on the use of the finite element method for solving the system of partial differential equations (PDE) and how to extract a solution from the results. Finally, the report briefly shows an example of a simulation along with visuals for easier analysis and understanding of the results.

2 FLUID FLOW

This section is going to provide the necessary knowledge on fluid mechanics to build a simulation. This section is going to talk about the navier stokes equation, 3 types of flow classification, reynolds numbers and their interpretability, and how to define inlet, outlet and wall boundary conditions for fluid flow.

2.1 Governing equations for Incompressible Fluid flow

This section covers the momentum and continuity equations for incompressible Newtonian fluid flow[7, page = 53, page = 59]:

$$\begin{cases} \dfrac{\partial u}{\partial t} + (u \cdot \nabla)u = -\dfrac{1}{\rho}\nabla p + \dfrac{\mu}{\rho}\nabla^2 u & \text{(1a)} \\ \nabla \cdot u = 0 & \text{(1b)} \end{cases}$$

where u is the velocity vector field, ρ is the fluid density, p is the pressure scalar field, and mu is the dynamic viscosity.

2.1.1 Continuity Equation. The continuity equation for incompressible fluid flow (1b) states that the mass within a closed system is constant. In fact, if the divergence of the velocity field were to be positive, this would imply that fluid is being added to the system. Whereas, a negative divergence of the velocity field would imply that fluid is leaving.

2.1.2 Momentum Equation. The momentum equation (1a) is derivated from Newtons Second Law. To elaborate, the left hand side represents the material derivative of the velocity of the fluid element. In short, the material derivative $\frac{\partial u}{\partial t} + (u \cdot \nabla)u$ accounts for both the time rate of change of velocity and the convective acceleration, which arises from the fluid's motion. This term represents how the velocity of a fluid element changes both due to time and due to the movement of the fluid itself.

The right hand side represents the forces acting on the fluid. The term $-\nabla p$ represents the pressure gradient force where the pressure difference accelerates fluids from regions of high pressure to regions of low pressure. The term $\mu\nabla^2 u$ is the stress tensor representing viscous forces acting between the fluid particles. This term accounts for the internal friction between fluid layers, resulting from normal and shear stresses acting between fluid particles. These stresses resist the relative motion of adjacent fluid layers and are proportional to the rate of deformation of the fluid.

2.2 Flow Classification and Reynolds Numbers

Fluid flow is generally classified into 3 categories:

- Laminar Flow
- Transitional Flow
- Turbulent Flow

These 3 classifications describe the velocity pattern of the flow. Laminar flow is characterized by a smooth undisturbed motion of the fluid. Hence, the velocity of laminar flow is time independent. On the other hand, turbulent flow is characterized by erratic and seemingly random movement of the fluid. Consequently, the velocity of turbulent flow is time dependent. Finally, transitional flow is the state of fluid flow transitioning from laminar flow to turbulent flow. A dimensionless parameter has been created in order to better

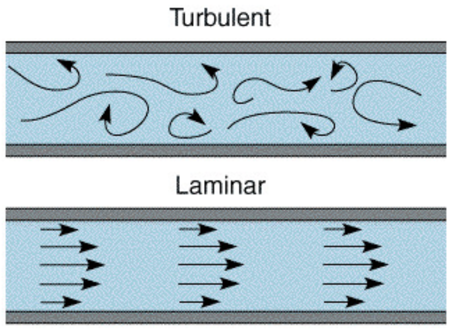


Figure 1: Visual on how to interpret the difference between laminar and turbulent flow

classify flow called the Reynolds number. This parameter takes the viscosity, density and velocity of the fluid as well as the space the flow is confined to in order to determine the flow behaviour:

$$Re = \frac{\rho u D}{\mu} \tag{2}$$

with D being the diameter of the flow region. While the interpretation of the Reynolds Number changes depending on the setting of the fluid flow, generally speaking, low Reynolds numbers indicate laminar flow and high Reynolds numbers indicate turbulent flow. Flow with high Reynolds Number are harder to simulate as the behaviour is very chaotic and more computational power is needed to guarantee accurate simulations.

2.3 Boundary Conditions for Fluid flow problems

There are 3 types of boundary conditions needed for fluid flow simulations: conditions for inflow, outflow and walls. This section explains how to define them.

2.3.1 Inlets. Defining the boundary from which fluid flows into the domain can be done by defining either the pressure at the inlet, the velocity or mass-flow of the fluid at the inlet.

$$\phi(x) = C \quad \forall x \in \Omega_{Inlet} \tag{3}$$

where ϕ is an arbitrary function and Ω_{Inlet} is the region of the inlet.

2.3.2 Outlets. Similarly for outlets, it is possible to use the same measurements to define the outlet boundary. however, it is important to make sure that the outlet conditions make sense when compared to the inlet conditions. For example, if the boundaries are defined using pressure, it should be enforced that the outlet has lower pressure than the inlet.

$$\phi(x) = C \quad \forall x \in \Omega_{Outlet} \tag{4}$$

where ϕ is an arbitrary function and Ω_{Outlet} is the region of the outlet.

2.4 Walls

Defining how the fluid behaves at walls has a bit more nuance. This section quickly presents the no-slip and free-slip conditions for solid walls. The no-slip condition states that the velocity of fluid at the wall is equal to 0. The free-slip condition states that only the velocity component tangential to the wall is 0.

$$u(x) = 0 \quad \forall x \in \Omega_{Wall} \quad \text{(No-Slip Condition)} \tag{5}$$

$$u \cdot n = 0 \qquad \text{(Free-Slip Condition)} \tag{6}$$

where u is the velocity field and n are vectors tangent to the walls.

3 FINITE ELEMENT METHOD

The Finite Element Method is a numerical method used to solve differential equations. The general idea of the finite element method is the division of a problem domain into smaller subdomains, called finite elements, which often are basic shapes such as triangles or rectangles. The solution to the differential equation is approximated for each element resulting in local set of equations. These local equations can be assembled to construct a global system of equations representing the entire problem domain. Solving the global system provides an approximation to the solution of the original differential equation.

The following sections are going to explain how the finite element method approximates the solution to the differential equation, how the local equations are derived for each element and how to assemble the global system, making sure that all constraints of the problem domain are satisfied.

3.1 Differential equations in the context of the FEM

A differential equation is an equation that equates one or more functions along with their derivatives. In the context of this paper, a differential equation defined on a problem domain Ω is of the form:

$$L(u(\mathbf{X})) = f(\mathbf{X}) \quad \forall \mathbf{X} \in \Omega \tag{7}$$

where L is a linear operator, u is the unknown function, f is an arbitrary known function, and X is the vector of all independent variables of functions f and u.

Furthermore, initial conditions and/or boundary conditions are required to force a unique solution on a differential equation. This is also often referred to 'closing the system'.

From an engineering point of view, initial conditions define the value(s) taken by a function at time $t = 0$. For example:

$$\begin{cases} \frac{d}{dt}u = f & \text{(differential equation)} \\ u(0) = 2 & \text{(initial condition)} \end{cases} \tag{8}$$

On the other hand, boundary conditions define the value(s) taken by a function at specific points in space. The type of boundary condition we are going to care about in this paper are the Dirichlet boundary condition as well as the Neumann boundary condition. To elaborate, the Dirichlet boundary condition force the solution to the differential equation to admit specific values at specific regions of the domain. Similarly, the Neumann boundary conditions force the derivative of the solution to admit specific values at specific regions of the domai. For example: [9]

$$\begin{cases} \frac{d}{dx}u &= f \quad \text{for} \quad x \neq 0 \quad \text{(differential equation)} \\ u &= 0 \quad \text{if} \quad x = 0 \quad \text{(Dirichlet b.c.)} \end{cases} \tag{9}$$

$$\begin{cases} \frac{d}{dx}u &= f \quad \text{for} \quad x \neq 0 \quad \text{(differential equation)} \\ \frac{d}{dx}u(x) &= 0 \quad \text{for} \quad x = 0 \quad \text{(Neumann b.c.)} \end{cases} \tag{10}$$

3.2 A toy problem

Let us consider a modified version of the heat transfer problem given in [8, page 30]. The heat transfer from a wall at $Tw = 200°C$ to a rod of length $L = 0.05$. The heat $u(x)$ across the rod is given by the following differential equation and boundary conditions:

$$k\frac{d^2u}{dx^2} = \frac{hP}{A_c}(u - T_\infty) \tag{11}$$

$$u(0) = \text{Tw} = 200°C \tag{12}$$

$$\left.\frac{du}{dx}\right|_{x=L} = 0 \tag{13}$$

where $k = 386$, $h = 20$, $P = 2 * 10^{-3}$, $A_c = \frac{\pi}{4} * 10^{-6}$. Using the notation from equation (7) and defining $m = \frac{hP}{A_c k}$:

$$L(u(x)) = \frac{d^2u}{dx^2} - mu \tag{14}$$

$$f(x) = -mT_\infty \tag{15}$$

$$\Omega = [0, L] \tag{16}$$

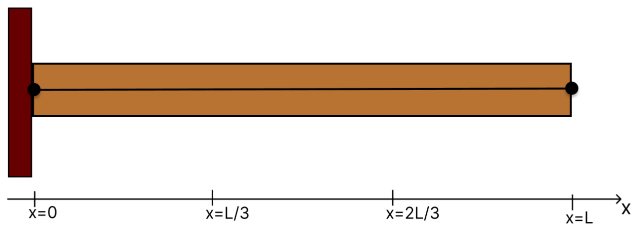


Figure 2: Copper rod attached to a wall at 200°C

Knowing the exact solution for this problem, we can measure and compare the effectiveness of the approximations that are computed using the FEM:

$$\text{Tinf} + \frac{\text{Tw} - \text{Tinf}}{cosh(mL))}cosh(m(L - x)) \tag{17}$$

3.3 Approximating the solution to a differential equation

One method to numerically solve a differential equation is to make assumptions on the shape of the unknown function u and try to determine an approximation $\tilde{u}$. The FEM usually assumes the shape of the function u to be an n-th order polynomial. As a result the approximation $\tilde{u}$ is of the form:

$$\tilde{u}(x) = c_0 + \sum_{i=1}^{n} c_i x^i \quad \forall k \in \{0, .., n\}, c_k \in \mathbb{R} \tag{18}$$

Because $\tilde{u}$ is an approximation it most likely does not exactly solve the differential equation. This arising error is called the residual R [8, page 18]:

$$R(x) = L(\tilde{u}(x)) - f(x) \tag{19}$$

Hence, the coefficients c_k in equation (18) are determined such that the approximation satisfies the boundary conditions and minimizes the residual. Additionally, it is important to observe the form of $L(u)$ when making assumptions on the solution of the differential equations. More precisely, it is necessary to determine the constraint $L(u)$ puts on the differentiability of the approximation function $\tilde{u}$. Consider:

$$L(u) = \frac{d^2u}{dx^2} - mu \tag{20}$$

Choosing a function $\tilde{u}$ that does not have a second order derivative might cause the approximation to not be able to capture an important aspect of the solution.

Finally, combining the problem domain and boundary conditions, the differentiability constraints and the assumptions made on the solution, it is possible to define a solution space for our differential equation.

3.4 Weighted Residual Statement

To determine the n coefficients c_i for the approximation $\tilde{u}$, n equations are required. Hence, the higher the polynomials degree, the more information is need to find all its coefficients. The simplest way of collecting these equations, is using the p points in the domain for which the values of u are known (boundary conditions), and create the remaining n-p equations by equating the residual R to zero at n-p arbitrarily chosen points. [A.1]

Unfortunately, this technique is far from perfect because it only aims to minimizes the Residual at certain points of the domain Ω. The end goal is to find an approximation that has the lowest possible residual over the entire domain Ω. Hence, instead of equating the residual to zero at certain points of the domain, we can formulate the desire of a zero residual using the weighted residual statement[8, page 24]:

$$\int_\Omega W_i(x)R(x)\, d\Omega = 0 \tag{21}$$

where $W_i(x)$ is an arbitrary weighing function and $R(x)$ the residual of our approximation. While the choice of $W_i(x)$ is free as long as it is integrable, non-zero, and satisfy the boundary conditions, the Galerkin Method, a special case of weighted residual statement, proposes to use parts of the approximation as weighing functions. [8, page 24] In fact if you rewrite the approximation in the following manner:

$$\tilde{u} = \phi(x) + \sum_{i=1}^{n} c_i N_i(x) \tag{22}$$

Then it is possible to take $W_i = N_i$. As seen in the above function, there exists an N_i for every undetermined c_i. As a result, we always end up with sufficient equations to solve for all coefficients.[A.2]If we also consider equation (19) we have:

$$\int_\Omega N_i(x)(L(\tilde{u}(x)) - f(x))\ d\Omega = 0 \tag{23}$$

3.5 Weak Formulation

As said in section 3.3 it is important to keep in mind the smoothness constraint L puts on $\tilde{u}$. Conveniently, the Weighted Residual Statement allows us to ease this constraint due to the problem formulation being written in an integral form. Consider the weighted residual statement:

$$\int_\Omega W_i R\ d\Omega = \int_\Omega W_i(L(\tilde{u}) - f)\ d\Omega = 0 \tag{24}$$

Assume $L(\tilde{u}) = \frac{d^2\tilde{u}}{dx^2}$ as an example:

$$\int_\Omega W_i \frac{d^2\tilde{u}}{dx^2}\ d\Omega = \int_\Omega W_i f\ d\Omega \tag{25}$$

The above expression is written in a form in which integration by parts can be performed which reduces the differentiation constraint on our approximation $\tilde{u}$, while increasing it for our weight function W_i:

$$\left[W_i \frac{d\tilde{u}}{dx}\right]_\Omega - \int_\Omega \frac{dW_i}{dx}\frac{d\tilde{u}}{dx}\ d\Omega = \int_\Omega W_i f\ d\Omega \tag{26}$$

Consequently, linear approximations have been made possible. The formulation (26) is called the Weak Form of the problem statement. The idea is to distribute the differentiations between the weighting function and the approximation. In practice, the smoothness constraint arising on W_i is already satisfied if we choose $W_i = N_i$ from (22). Moreover, the weak formulation conveniently allows the integration of Neumann boundary conditions into the equation through the boundary term that appears when integrating by parts.

3.6 Piecewise continuous trial functions

In practice, a single polynomial fails at describing the behavior of a function over the entire domain. It turns out that splitting the target function domain into subdomains and approximating the function piecewise results in a lower residual.

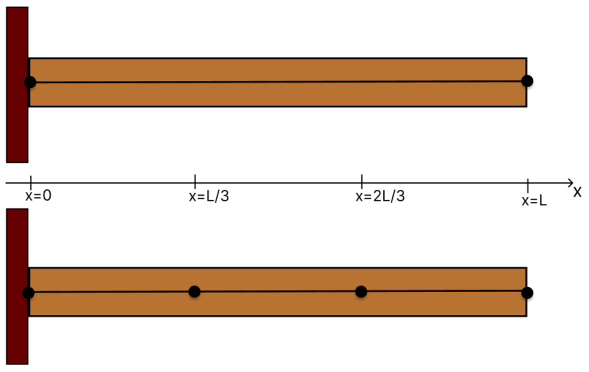


Figure 3: Splitting the problem domain into subsections. Using three 'connected' functions instead of a single function.

The issue with this strategy is that we need to know the values of n+1 points to interpolate for a nth order polynomial. This, however, is not possible as our function is unknown and we do not posses any data other than of the boundary conditions.

3.7 Finite Elements

Finite elements can be seen as mathematical objects that are used to discretize the problem domain. A finite element is defined by its nodes, its dimension of freedom, and its shape functions. The nodes represent the vertices or points of the finite element, while the degrees of freedom correspond to the number of independent parameters that define the element (calculated as the DOF per node multiplied by the number of nodes). The shape functions of a finite element describe how the nodal values are interpolated across the element. These shape functions are defined best on the elements geometry and express the approximate solution at any point within the element.

Defining a finite element requires specifying its geometry, which includes the positions of the nodes and the connectivity between them. Additionally, you need to define the polynomial order of the shape functions, which determines the type of the finite element. Each node has a coordinate pair (x,y) representing its position in space, and a corresponding field value, like displacement or temperature, that represents the result of the function at that node. These values are used to interpolate the field variable throughout the finite element. The finite elements approximation should be a similar form as equation (22):

$$\tilde{u} = \sum_i^n u_i * N_i(x) \tag{27}$$

where n is the number of nodes of the element, u_i are the nodal field value and N_i are the shape functions of the finite element.

Take the 1D bar finite element as an example:

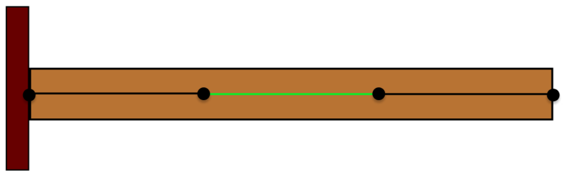


Figure 4: First order 1D finite element in green

3.7.1 *Derivation of the first order 1D finite element.*

The first order 1D finite element is composed of two nodes connected by an edge. Denote the two nodes $(0, u_1)$ and (l, u_2), where l is the length of the finite element. Knowing that

$$\tilde{u}(x) = c_0 + c_1 x \tag{28}$$

we can insert our two nodes into $\tilde{u}$ and write down:

$$\begin{cases} u_1 = c_0 \\ u_2 = c_0 + c_1 l \end{cases} \tag{29}$$

Solving the system gives us:

$$\begin{cases} c_0 = u_1 \\ c_1 = \frac{u_2 - u_1}{l} \end{cases} \tag{30}$$

And

$$\tilde{u}(x) = u_1 + \frac{u_2 - u_1}{l}x \tag{31}$$

Finally, we develop the expression to obtain an expression under the form of equation (27):

$$\tilde{u}(x) = u_1(1 - \frac{x}{l}) + u_2\frac{x}{l} \tag{32}$$

Thus, the shape functions of the first order 1D finite element are:

$$N_1(x) = 1 - \frac{x}{l}$$
$$N_2(x) = \frac{x}{l}$$

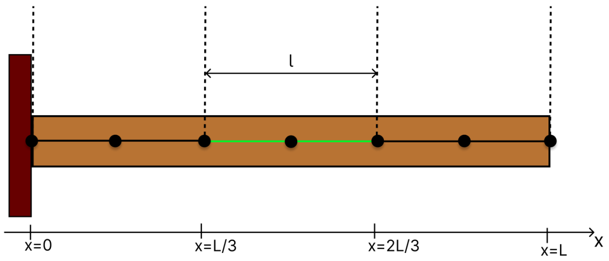


Figure 5: 2nd order 1D finite element in green

3.7.2 Derivation of the second order 1D finite element. The second order 1D finite element is composed of three nodes connected by two edge. Denote the three nodes $(0, u_1)$, $(\frac{l}{2}, u_2)$, and (l, u_3), where l is the length of the finite element. This time the approximation is given by:

$$\tilde{u}(x) = c_0 + c_1x + c_2x^2 \tag{33}$$

Because there are 3 coefficients to determine, we need 3 equations. As a result, our finite element is compose of 3 nodes $(0, u_1)$, $(l/2, u_2)$, and (l, u_3). Insert the three nodes into $\tilde{u}$ gives us:

$$\begin{cases} u_1 = c_0 \\ u_2 = c_0 + c_1\frac{l}{2} + c_2\frac{l^2}{4} \\ u_3 = c_0 + c_1l + c_2l^2 \end{cases} \tag{34}$$

Solving the system gives us:

$$\begin{cases} c_0 = u_1 \\ c_1 = \frac{1}{l}(-3u_1 + 4u_2 - u_3) \\ c_2 = \frac{1}{l^2}(2u_1 - 4u_2 + 2u_3) \end{cases} \tag{35}$$

And

$$\tilde{u}(x) = u_1 + \frac{1}{l}(-3u_1 + 4u_2 - u_3)x + \frac{1}{l^2}(2u_1 - 4u_2 + 2u_3)x^2 \tag{36}$$

Again, we develop the expression to obtain an expression under the form of equation (27):

$$\tilde{u} = u_1(1 - \frac{3}{l}x + \frac{2}{l^2}x^2) + u_2(-\frac{1}{l}x + \frac{2}{l^2}x^2) + u_3(\frac{4}{l}x + \frac{4}{l^2}x^2) \tag{37}$$

Thus, the shape functions of the second order 1D finite element are:

$$N_1(x) = 1 - \frac{3}{l}x + \frac{2}{l^2}x^2 \tag{38}$$

$$N_2(x) = -\frac{1}{l}x + \frac{2}{l^2}x^2 \tag{39}$$

$$N_3(x) = \frac{4}{l}x + \frac{4}{l^2}x^2 \tag{40}$$

3.7.3 Further dimension/order finite elements. The number of coefficients needed for a finite element is given by the dimension of the problem space as well as the order of the approximation. In general, to determine an elements shape functions, the nodes position and its field values have to be defined. Afterwards, constructing a Vandermonde matrix with the nodal positions and adding the nodal filed values to create a system of equations that can be solved for the coefficients of the approximation. Otherwise, there are many other strategies to interpolate a function such as lagrange interpolation[8, page 158]. Further derivations of finite elements can be found in the appendix of this report [A.3]. Finally, once the finite elements have been defined and the problem domain has been partitioned, the problem can be numerically solved.

3.8 Assembly

After obtaining the finite element, one might ask how they can be used to numerically solve a differential equation. The idea is to use the approximation of the function over the finite element as well as its shape functions in the weak formulation of our problem to create a system of equations that solves for all the nodal values of our mesh.

From now on we are going to express the system of equations with matrix notation as it is easier to see what is happening.

Each finite element provides a set of equations in terms of its nodal values and its shape functions resulting in a so called local matrix. Combining the local matrices of all finite elements gives a global matrix in terms of all nodal values.

The derivation of the local matrix is presented using our toy problem on a first order 1D finite element [3.2]. Take the weak formulation of the governing differential equation:

$$\left[W_i\frac{d\tilde{u}}{dx}\right]_\Omega - \int_\Omega \frac{dW_i}{dx}\frac{d\tilde{u}}{dx}\,d\Omega - k'\int_\Omega W_i\tilde{u}\,d\Omega = -k'T_\infty\int_\Omega W_i\,d\Omega \tag{41}$$

We take $\tilde{u}$ and W_i corresponding to the second order finite element and define Ω as the space of the finite element:

$$\tilde{u}(x) = \begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{42}$$

$$W_i = N_i(x) \tag{43}$$

It is advantageous to compute the derivatives of the Weight and the approximate function in advance:

$$\frac{d}{dx}\tilde{u}(x) = \begin{bmatrix} \frac{d}{dx}N_1(x) & \frac{d}{dx}N_2(x) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{44}$$

$$\frac{d}{dx}W_i = \frac{d}{dx}N_i(x) \tag{45}$$

A more compact notation gives:

$$\tilde{u} = [N][u] \tag{46}$$

$$\frac{d}{dx}\tilde{u} = \left[\frac{dN}{dx}\right][u] \tag{47}$$

This allows us to rewrite the weak formulation (41) as:

$$\int_\Omega \left[\frac{dN}{dx}\right]^T \left[\frac{dN}{dx}\right][u]\ d\Omega + k' \int_\Omega [N]^T[N][u]\ d\Omega = k'T_\infty \int_\Omega [N]\ d\Omega \tag{48}$$

which can be simplified to:

$$\int_\Omega \left[\frac{dN}{dx}\right]^T \left[\frac{dN}{dx}\right] + [N]^T[N]\ d\Omega[u] = k'T_\infty \int_\Omega [N]\ d\Omega + Q \tag{49}$$

Knowing from (38 - 40) that:

$$[N] = \begin{bmatrix} 1 - \frac{x}{l} \\ \frac{x}{l} \end{bmatrix} \tag{50}$$

$$\left[\frac{dN}{dx}\right] = \begin{bmatrix} -\frac{1}{l} \\ \frac{1}{l} \end{bmatrix} \tag{51}$$

We can compute $\int_\Omega [\frac{dN}{dx}]^T[\frac{dN}{dx}]\ d\Omega$, $\int_\Omega [N]^T[N]\ d\Omega$ and $\int_\Omega [N]\ d\Omega$:

$$\int_\Omega \left[\frac{dN}{dx}\right]^T \left[\frac{dN}{dx}\right] d\Omega = \begin{bmatrix} \frac{1}{l} & -\frac{1}{l} \\ -\frac{1}{l} & \frac{1}{l} \end{bmatrix}$$

$$\int_\Omega [N]^T[N]\ d\Omega = \begin{bmatrix} \frac{l}{3} & \frac{l}{6} \\ \frac{l}{6} & \frac{l}{3} \end{bmatrix}$$

$$\int_\Omega [N]\ d\Omega = \begin{bmatrix} \frac{l}{6} \\ \frac{l}{6} \\ 14\frac{l}{3} \end{bmatrix}$$

The coefficient matrix A ends up being:

$$A = \begin{bmatrix} kp * \frac{l}{3} + \frac{1}{l} & kp * \frac{l}{6} - \frac{1}{l} \\ kp * \frac{l}{6} - \frac{1}{l} & kp * \frac{l}{3} + \frac{1}{l} \end{bmatrix} \tag{52}$$

The variable vector ends up being the nodal values $[u]$ and the constant vector b equals:

$$b = k'T_\infty \begin{bmatrix} \frac{l}{2} \\ \frac{l}{2} \end{bmatrix} \tag{53}$$

The local matrix then is written as:

$$\begin{bmatrix} kp * \frac{l}{3} + \frac{1}{l} & kp * \frac{l}{6} - \frac{1}{l} \\ kp * \frac{l}{6} - \frac{1}{l} & kp * \frac{l}{3} + \frac{1}{l} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = k'T_\infty \begin{bmatrix} \frac{l}{2} \\ \frac{l}{2} \end{bmatrix} \tag{54}$$

The same result is obtained for every element in our problem domain, the only thing differing being the unknown vector representing the nodal values.

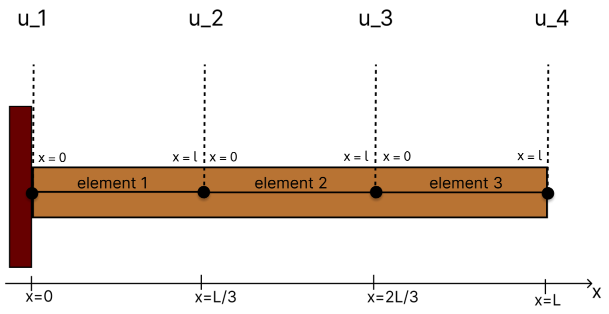


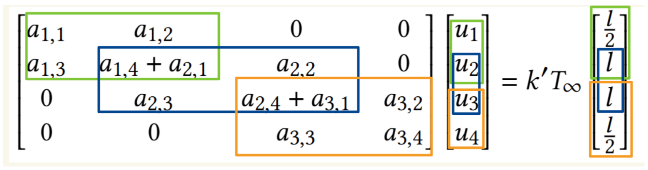
Figure 6: Notice how some nodes are part of different finite elements

Assembling the local matrices into a global matrix is quite simple:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 \\ a_{1,3} & a_{1,4} + a_{2,1} & a_{2,2} & 0 \\ 0 & a_{2,3} & a_{2,4} + a_{3,1} & a_{3,2} \\ 0 & 0 & a_{3,3} & a_{3,4} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = k'T_\infty \begin{bmatrix} \frac{l}{2} \\ l \\ l \\ \frac{l}{2} \end{bmatrix} \tag{55}$$

$a_{i,j}$ such that i is the coefficient matrix of the i-th finite element and j is the j-th element of the coefficient matrix.

Simply put, we added all the contributions of the local coefficient matrices and constant vectors for the unknown nodal values to the global matrix.



$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 \\ a_{1,3} & a_{1,4} + a_{2,1} & a_{2,2} & 0 \\ 0 & a_{2,3} & a_{2,4} + a_{3,1} & a_{3,2} \\ 0 & 0 & a_{3,3} & a_{3,4} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = k'T_\infty \begin{bmatrix} \frac{l}{2} \\ l \\ l \\ \frac{l}{2} \end{bmatrix}$$

Figure 7: Contribution of every finite element colored in a different color

After applying our boundary conditions, we can obtain the nodal values that fit best our problem with the given approximation.

3.9 Boundary Conditions

The neumann boundary conditions are enforced through the boundary terms arising from integration by parts. The Dirichlet boundary conditions are enforced by setting an unknown to its value and subtracting the value times its corresponding coefficient in each row from the constant vector. For example, if we know that $\tilde{u}(x = 0) = C$, we set the nodes at $x = 0$ to C and update the other formulas:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a_{1,4} + a_{2,1} & a_{2,2} & 0 \\ 0 & a_{2,3} & a_{2,4} + a_{3,1} & a_{3,2} \\ 0 & 0 & a_{3,3} & a_{3,4} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = k'T_\infty \begin{bmatrix} C \\ l - a_{1,3}C \\ l \\ \frac{l}{2} \end{bmatrix} \tag{56}$$

which then allows us to solve a smaller system of equations:

$$\begin{bmatrix} a_{1,4}+a_{2,1} & a_{2,2} & 0 \\ a_{2,3} & a_{2,4}+a_{3,1} & a_{3,2} \\ 0 & a_{3,3} & a_{3,4} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = k'T_\infty \begin{bmatrix} l - a_{1,3}C \\ l \\ \frac{l}{2} \end{bmatrix} \quad (57)$$

Solving this system gives us an approximation for the Steady State solution of our problem. When analyzing figure (8) shows us that subdividing the problem domain into three 1st order finite elements is not appropriate to approximate the heat transfer : Analysing the

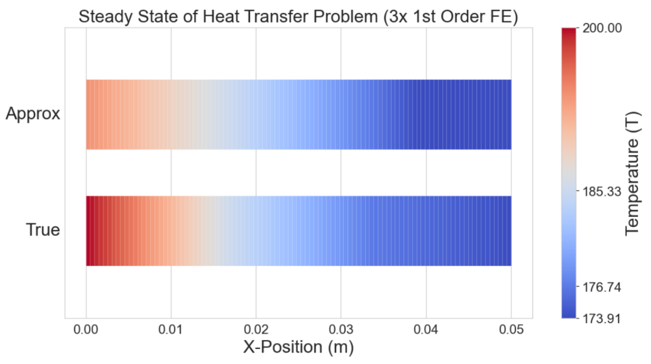


Figure 8: Comparison between the approximate and true steady state of the toy problem with three 1st order finite elements

absolute error between the true and approximated steady state shows how the method fails to provide accurate results, especially at the edges of the domain:

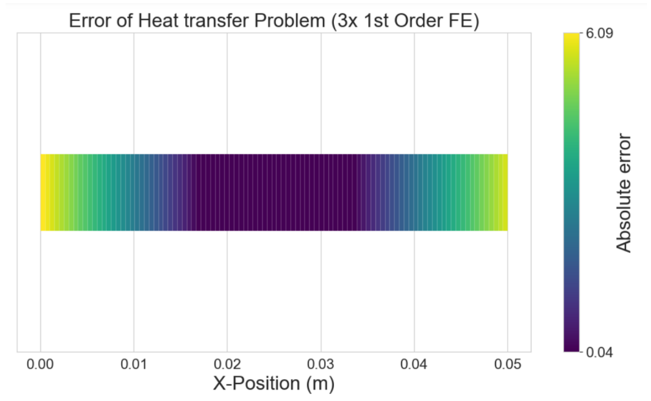


Figure 9: Error distribution for the approximation with three 1st order finite elements

There are two possible ways to improve our approximation:

- increase the number of elements
- increase the order of the elements

Trying to approximate the function using 6 finite elements instead of 3 turns out to improve the accuracy near the wall but worsens the accuracy further away (B.17) (B.18).
It is not sufficient to just increase the number of elements to improve the accuracy. In fact, increasing the order of the element has a much larger impact on the accuracy than the number of elements. With just one second order element the total error decreases by a lot(B.??)(B.??).

3.10 Workflow

Working with the finite element method can be generalized into a workflow. First, write down the problem definition. This include the governing differential equation as well as boundary conditions. Then, write the Weighted Residual Statement and set up the weak form of the problem while making sure to satisfy the Neumann boundary conditions.

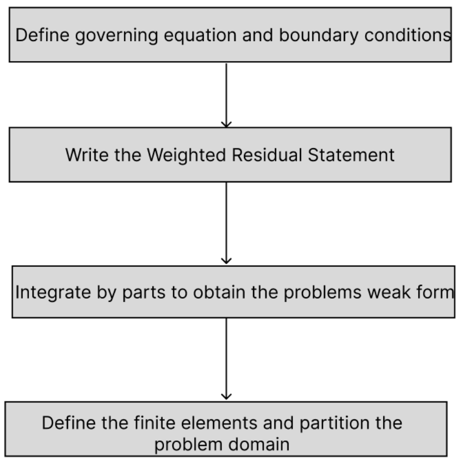

Define governing equation and boundary conditions
Write the Weighted Residual Statement
Integrate by parts to obtain the problems weak form
Define the finite elements and partition the problem domain

Figure 10: Workflow for the problem definition

Afterwards, it is necessary to define the finite element and compute the shape functions. Then, the local systems are derived for each finite element and assembled into a global system. Finally the Dirichlet boundary conditions are integrated into the system and the problem solved.

Finally, you want to visualize the problem to allow for easier interpretability of the simulation results. Hence, the general work flow requires the definition of the problem in the context of the FEM, the construction and solving of the system and the visualization of the result.

3.11 Accuracy comparison

This section compares the function approximation strategies seen in this report using the toy problem defined in section (3.2). The L^2 norm between the exact and approximate solution are computed to measure how close the approximation is to the true function. Concretely, we are going to compare the following methods:

- Satisfying boundary conditions and setting the residual at some points to 0 (S)
- Satisfying boundary conditions and using Weighted Residual Statement (WR)

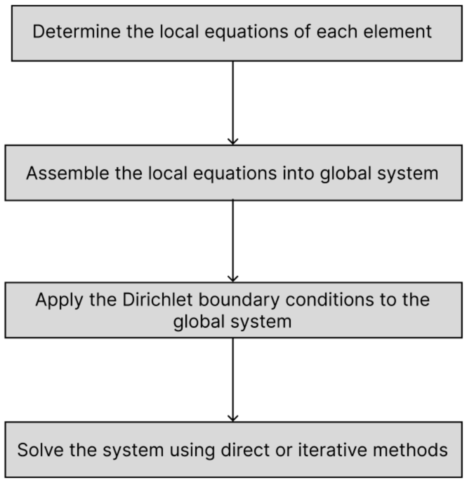

Determine the local equations of each element
Assemble the local equations into global system
Apply the Dirichlet boundary conditions to the global system
Solve the system using direct or iterative methods

Figure 11: Workflow for system assembly and solving

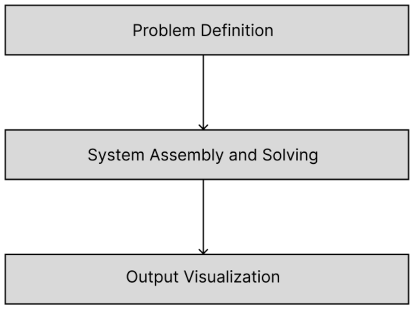


Figure 12: Workflow for system assembly and solving

- Basic Interpolation using sample points from the true function (IP)
- three first order finite elements over the domain (1D_FEM)
- one second order finite element over the domain (2D_FEM)

Each method is tested on the same problem for various lengths of the bar. It turns out that the exact solution for this problem is not easily approximated with polynomials. As seen previously, the higher-order finite element method performs best in this problem.

Bar length	S	WR	IP	1D_FEM	2D_FEM
0.050	0.117	0.020	0.011	1.258	0.011
0.100	1.391	0.315	0.179	5.065	0.170
0.200	7.495	2.944	1.763	11.683	1.588
0.400	21.688	14.173	9.303	12.068	7.899
0.800	46.789	37.885	28.570	18.332	23.958
1.600	82.440	71.635	59.461	49.561	51.294
3.200	129.785	115.853	100.183	94.123	88.730

Table 1: L^2 distance between the true function and the approximation using various techniques

However, the precision drops as the bar length increases. This is due to the nature of the solution of the problem (17). The function can still be approximated to be a polynomial when the distance is kept short.

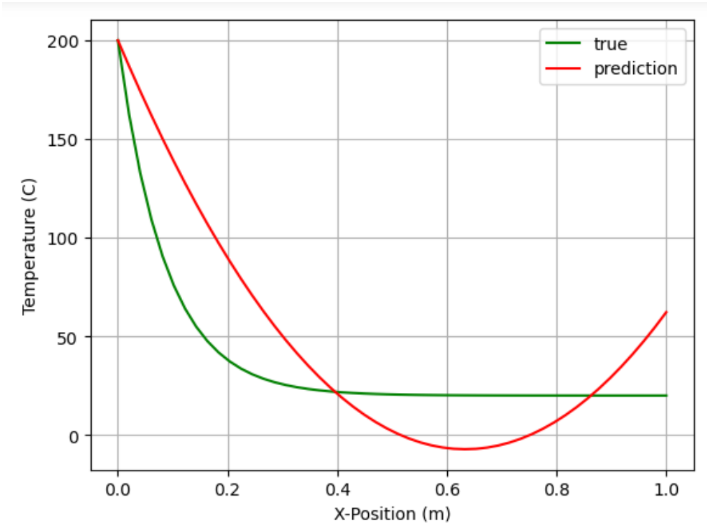


Figure 13: For a bar of 1 meter the second order finite element cannot accurately describe the solution

3.12 On direct and iterative solvers

The finite element methods most time consuming step is solving the assembled global system:

$$Ax = b \tag{58}$$

There are two types of methods for solving such systems: Direct and Iterative methods. Choosing the right type of method is essential to improve execution time and possibly improve memory usage. Both types of methods try to solve systems.

On the one hand, direct methods are deterministic methods that determine the exact solution for a system, assuming that there is a unique solution, similar to Gaussian elimination. Decomposition methods are frequently applied to matrices to simplify operations on these matrices.[3] For example, solving a system $Ax = b$ directly using Gaussian elimination takes O(n^3) complexity. However, performing a LU decomposition on A results in

$$A = LU \tag{59}$$

where L is a lower triangular matrix and U is an upper triangular matrix. The system $Ax = b$ can then be writen as $LUx = b$ and solved by first solving

$$Ly = b \tag{60}$$

and then

$$Ux = y \tag{61}$$

Determining the matrices L and U in (59) takes $O(n^3)$ time but solving (60) and (61) then only take $O(n^2)$ time, greatly improving computational speed.

On the other hand, iterative methods guess a solution x^0 to the system $Ax = b$ and then try to iteratively update the solution $x^{(k)}$ by minimizing its residual $b - Ax^{(k)}$, until a satisfactory solution has been found. Iterative methods use preconditioners to improve the stability and convergence of iterative methods. The goal of a preconditioner is to reduce the condition number κ of a matrix:

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \tag{62}$$

Where λ_{max} is the largest eigen-value of A and λ_{min} is the smallest eigen-value. In fact, the number of iterations that are needed to get a satisfactory solution $x^{(s)}$ to $Ax = b$ is directly linked to the condition number of the matrix A[1]. There are many methods that are constructed for specific types of systems. For example, the conjugate gradient method (CG) is intended to solve a system $Ax = b$, where A is symmetric-positive-definite, whereas the biconjugate gradient method (BiCS) works for systems $Ax = b$ where the matrix A is non-symmetric matrices. In general, CG is faster than BiCG because CG takes advantage of the additional assumptions that the coefficient matrix of the system is symmetric positive definite. As a result, it is important to understand the system we are trying to solve in order to make sure the best possible method is used.

Regarding convergence, the conjugate gradient method satisfies the following bound on its error[5]:

$$\frac{||x - x^{(k)}||_A}{||x - x^{(0)}||_A} \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \tag{63}$$

Figure (14) shows how computational cost increases with the conditioning number. Note that the bound error is a ratio between the initial guess and the kth guess. Hence, it is important to make a descent initial guess for the method to converge. There are many

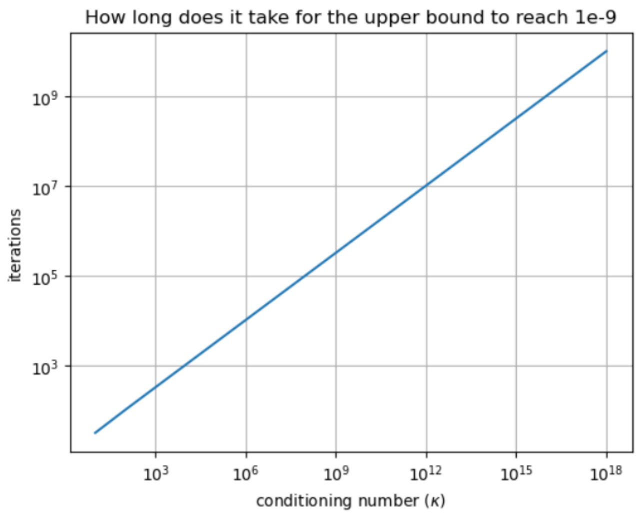


Figure 14: Number of iterations needed to reach upper bound of 1e-9 depending on conditioning number

preconditioners such as the Jacobi method, which premultiplies A by its diagonal matrixs inverse, or Incomplete LU Preconditioner, which premultiplies A with the inverse of a modified LU decomposition of A such that its sparsity is efficient.[1] In general, a preconditoner for the system $Ax = b$ is a matrix M, approximating A, such that $M^{-1}A$ has a better conditioning number than A. In fact, the preconditioned system becomes:

$$M^{-1}Ax = M^{-1}b \tag{64}$$

Similarly to iterative methods, knowing the nature of the system allows for a better choice of preconditioner: Jacobi preconditioning works well for matrices with dominant diagonals.[1]

Both direct and iterative methods have their advantages. Direct methods are advantageous when solving systems that are small and well conditioned. However, iterative methods using preconditioners outperform direct methods when the system is very large and ill-conditioned. It helps to consider two different problems:

- Flow through a curved pipe, 1600 node grid, 1000 time steps
- Flow through pipe with cylinder obstacle, 7584 nodes, 12800 time steps

	Curved Pipe	Cylinder Obstacle
Direct Method	6.7s	49m
Iterative Method	25.0s	15m

Table 2: Comparison between Iterative and Direct Methods depending on problem form

4 SOLVING THE NAVIER STOKES EQUATION USING THE FINITE ELEMENT METHOD

This following section covers topics related to the general strategy for numerically solving the Navier-Stokes equations using the finite element method. Specifically, the flow of an incompressible Newtonian fluid is presented as an example. To elaborate, this section touches on possible time discretizations of the Navier-Stokes equations, solution strategies such as the Chorins projection method, and an incremental pressure correction method. Finally, comparisons between iterative and direct methods for solving systems of equations are made, which is relevant due to the large systems that are assembled by the finite element method.

The Navier-Stokes equation can be expressed using the cauchy stress tensor[6, page = 395]:

$$\begin{cases} \frac{\partial u}{\partial t} + (u \cdot \nabla)u = \nabla \cdot \sigma(u, p) & (65a) \\ \nabla \cdot u = 0 & (65b) \end{cases}$$

with

$$\begin{cases} \sigma(u, p) = 2v\epsilon(u) - pI & (66a) \\ \epsilon(u) = \frac{1}{2}(\nabla u + \nabla u^T) & (66b) \end{cases}$$

where u is the unknown velocity vector, p is the pressure scalar and v is the kinematic velocity. Solving the Navier-Stokes equations leads to the determination of 4 unknowns: u_x, u_y, u_z and p. These unkowns have to be calculated numerically as there has yet to be an analytical solution.

4.1 Time discretization

The first step to numerically solve the Navier-Stokes equation is to decide how to discretize the time derivative. It is possible to use an explicit, implicit, or semi-implicit method to discretize the time derivative. There is a trade-off between stability and computational cost between the discretization techniques. In detail, explicit methods are computationally cheap but risk instability during the simulation, whereas implicit methods are unconditionally stable but very expensive because a system of equations must be solved. Therefore, many methods use a semi-implicit scheme, trying to maintain the stability while reducing the computational costs by treating some unkowns implicitly and some explicitly.[4] The choice of step size for explicit and semi-implicit methods can be taken from the courant-friedrichs-lewy (CFL) condition, which states[2]

$$\frac{u\Delta t}{\Delta x} \leq C_{max} \tag{67}$$

where u is the magnitude of the velocity of the system and Δt and Δx are the steps size done for numerically derivating with respect to time and space. C_max is a constant that depends on the method that is being used. Typically, for explicit methods you take $C_{max} = 1$.

4.2 Pressure correction schemes

The most intuitive method to solve the Navier-Stokes equation is called chorins projection method, also called a non-incremental pressure correction scheme.[6] The general idea behind this method is to compute a tentative velocity while ignoring the pressure (68), computing the pressure from the tentative velocity by solving a poisson equation (69) and then finally update the velocity using the tentative velocity as well as the computed pressure (70).

$$\frac{u^* - u^n}{\Delta t} = -(u^n \cdot \nabla)u^n + v\nabla^2 u^* \tag{68}$$

$$\nabla^2 p^{n+1} = \frac{\rho}{\Delta t}\nabla \cdot u^* \tag{69}$$

$$u^{n+1} = u^* - \frac{\Delta t}{\rho}\nabla p^{n+1} \tag{70}$$

One issue arising with this kind of pressure-correction method is that the resulting velocity will not perfectly satisfy the continuity equation, which can lead to inaccurate behaviour of the fluid. Chorins projection method is of first-order accuracy.[10] However, for turbulent fluid flow chorins method performs poorly because the time-step needed to maintain stability impractically small. Fortunately there are higher order projection methods such as the 'Incremental Pressure Correction Scheme' (IPCS). IPCS is a second order method that has the a similar strategy as chorins projection method but improves on it by using the pressure from the previous time step and an 'intermediate' velocity for the computation of the stress tensor:

$$\rho\frac{u^* - u^n}{\Delta t} = -\rho u^n \cdot \nabla u^n - \nabla \cdot \sigma(u^{n+\frac{1}{2}}, p^n) \tag{71}$$

$$\nabla^2 p^{n+1} = \nabla^2 p^n - \frac{\rho}{\Delta t}\nabla \cdot u^*. \tag{72}$$

$$\rho(u^{n+1} - u^*) = -\Delta t\nabla(p^{n+1} - p^n) \tag{73}$$

Plotting the accuracy with respect to the time-step size shows how IPCS performs better than chorins projection method.

4.3 Simulation evaluation

This section is going to finish off the main report by presenting some results from simple fluid flow simulations. The simulation has been created using the FEniCSx library in python.

The following simulation is fluid flow through a curved pipe at low Reynolds number: $Re = 100$. In other words, the fluid had a dynamic viscocity of $0.001Pa$ and a density of 1 $\frac{kg}{L}$. The pressure gradient from the inlet to the outlet was 40 pascal to 0 pascal and the walls enforece a no-slip boundary condition. The mesh consists of 6400 nodes of second order triangle finite elements for the velocity and first order triangle elements for the pressure. To solve the differential system, the IPCS scheme has been used.

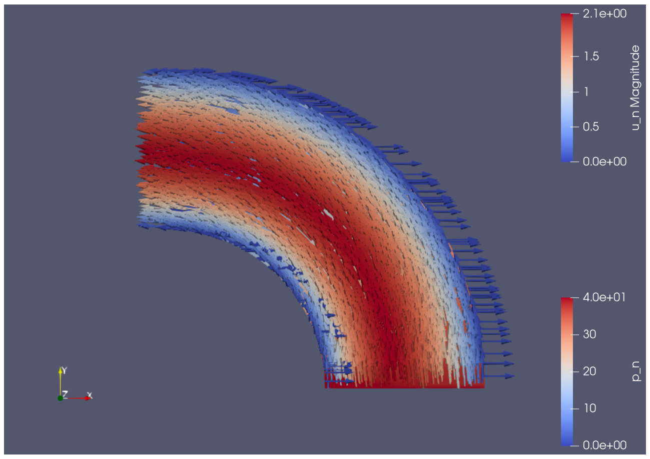


Figure 15: Flow through curved pipe

The velocity at the walls is slower than in the center of the pipe, due to the high viscosity of the simulated fluid.

A similar simulation for larger Reynolds numbers, fails to maintain stability. Possible reasons either lie in physically illogical boundary conditions or insufficient accuracy from the pressure correction schemes.

5 CONCLUSION

All in all, the finite element method is a powerful numerical tool that can be used to solve differential equations such as the navier stokes equations. The finite element method discretises a problem domain and approximates the target functions with piecewise polynomial functions. Moreover, for small systems, the global matrix is quite small and direct methods are sufficient to solve the system. However, finer meshes are needed for complex systems such as turbulent flow, which leads to larger global matrices and requires iterative methods to solve in an appropriate amount of time. Furthermore, preconditioners are used on the systems to improve convergence in iterative methods.

ACKNOWLEDGMENT

This endeavour would not have been possible without Dr Ezhilmathi Krishnasamy, who provided invaluable guidance and feedback during this project.

REFERENCES

[1] Robert Beauwens. 2004. Iterative solution methods. *Applied Numerical Mathematics* 51, 4 (2004), 437–450.
[2] Rajesh Bhaskaran and Lance Collins. 2002. Introduction to CFD basics. *Cornell University-Sibley School of Mechanical and Aerospace Engineering* (2002), 1–21.
[3] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. 2017. *Direct methods for sparse matrices.* Oxford University Press.
[4] Thomas Y Hou and Zuoqiang Shi. 2008. An efficient semi-implicit immersed boundary method for the Navier–Stokes equations. *J. Comput. Phys.* 227, 20 (2008), 8968–8991.
[5] Jörg Liesen and Zdenek Strakos. 2013. *Krylov subspace methods: principles and analysis.* Numerical Mathematics and Scie.
[6] Anders Logg, Kent-Andre Mardal, and Garth Wells. 2012. *Automated solution of differential equations by the finite element method: The FEniCS book.* Vol. 84. Springer Science & Business Media.
[7] Meinhard T Schobeiri. 2010. *Fluid mechanics for engineers: a graduate textbook.* Springer Science & Business Media.
[8] P Seshu. 2003. *Textbook of finite element analysis.* PHI Learning Pvt. Ltd.
[9] Amir Sharifahmadian. 2015. *Numerical models for submerged breakwaters: coastal hydrodynamics and morphodynamics.* Butterworth-Heinemann.
[10] E Weinan and Jian-Guo Liu. 1995. Projection method I: convergence and numerical boundary layers. *SIAM journal on numerical analysis* (1995), 1017–1057.

A APPENDIX: EXAMPLES

A.1 Heat transfer of a bar (simple)

Let us show an example using the scenario of a metal bar that transfers heat from a surface wall. Suppose the bar has an insulated tip:

$$\begin{cases} k\dfrac{d^2T}{dx^2} = \dfrac{Ph}{A_c}(T - T_\infty) & \text{(74a)} \\ T(0) = T_w & \text{(74b)} \\ \left.\dfrac{dT}{dx}\right|_L = 0 & \text{(74c)} \end{cases}$$

with

$$\begin{cases} k = 200W/m/^\circ C \\ P = 2mm \\ A_c = \frac{\pi}{4} * 10^{-6} \\ h = 20W/m^{2\circ}C \\ T_w = 300^\circ C \\ T_\infty = 30^\circ C \\ L = 50mm \end{cases} \quad (75)$$

First Notice that the governing equation (85a) puts a restraint on the smoothness of our trial function. We require the trial function to be twice differentiable:

$$\tilde{T}(x) = c_0 + c_1 x + c_2 x^2 \quad (76)$$

We now use the boundary conditions to determine the coefficients:

$$\tilde{T}(0) = c_0 = 200 \quad (77)$$

$$\left.\frac{d\tilde{T}}{dx}\right|_L = c_1 + 2c_2 L = 0 \quad (78)$$

Inserting (8) and (9) into (85a) we get:

$$\tilde{T}(x) = 200 - 2c_2 Lx + c_2 x^2 \quad (79)$$

If our approximation were correct we would have:

$$k\frac{d^2\tilde{T}}{dx^2} - \frac{Ph}{A_c}(\tilde{T} - T_\infty) = 0 \quad (80)$$

The residual for the trial solution would then be computed using this:

$$R(x) = k\frac{d^2\tilde{T}}{dx^2} - \frac{Ph}{A_c}(\tilde{T} - T_\infty) \quad (81)$$

Inserting (10) into (12)

$$R(x) = 2kc_2 - \frac{Ph}{A_c}(200 - 2c_2 Lx + c_2 x^2 - T_\infty) \quad (82)$$

A simple way of determining the remaining coefficientc_2 would be to take a point on the bar and try to put the residual at that point to 0. Let us do this for $x = L$:

$$R(L) = 2kc_2 - \frac{Ph}{A_c}(200 - 2c_2 L^2 + c_2 L^2 - T_\infty) = 0 \quad (83)$$

resulting in $c_2 = \frac{\frac{Ph}{A_c}(200 - T_\infty)}{2k + \frac{Ph}{A_c}L^2} \approx 10193.573$

The resulting trial function becomes:

$$\tilde{T}(x) = 200 - 1019.357Lx + 10193.573x^2 \quad (84)$$

A.2 Heat transfer of a bar (WR)

Take the same problem statement as in the previous example:

$$\begin{cases} k\dfrac{d^2T}{dx^2} = \dfrac{Ph}{A_c}(T - T_\infty) & \text{(85a)} \\ T(0) = T_w & \text{(85b)} \\ \left.\dfrac{dT}{dx}\right|_L = 0 & \text{(85c)} \end{cases}$$

Assume $\tilde{u}$ is a second order polynomial and applying the boundary conditions we get:

$$\tilde{u} = 200 - 2c_2 Lx + c_2 x^2 \quad (86)$$

This is the same notation as for equation (5). Still, we can rewrite this to correlate with equation(7)

$$\tilde{u} = 200 - c_2(-2Lx + x^2) \quad (87)$$

c_2 can then be determine by solving the weighted residual statement for c_2:

$$\int_0^L (-2Lx + x^2) * (2kc_2 - \frac{Ph}{A_c}(200 - 2c_2 L^2 + c_2 L^2 - T_\infty))dx = 0 \quad (88)$$

In fact, for every undetermined c_i, there is an N_i that we can use to create a equation such that there are enough equations to find all undetermined c_i.

Computing the integral and isolating for c_2 gives us:

$$c_2 = 5v\frac{-Tinf + Tw}{2(2L^2 v + 5)}) = 10490.620347275646 \quad (89)$$

with $v = \frac{Ph}{A_c k}$. We end up with the following approximation:

$$\tilde{T}(x) = 200 - 1049.062Lx + 10490.620x^2 \quad (90)$$

A.3 Derivation of the first order triangle finite element

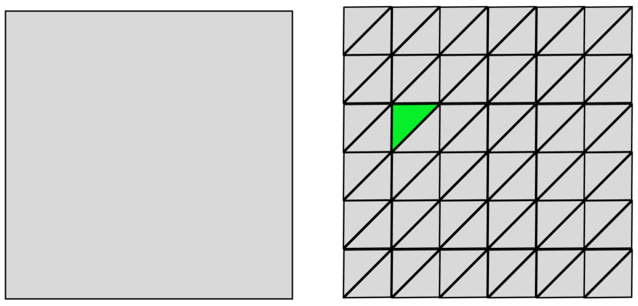


Figure 16: 2D Triangle finite element in green

The first order triangle finite element is composed of three nodes connected by three line in the shape of a triangle. Denote the three nodes (x_1, y_1, u_1), (x_2, y_2, u_2), and (x_3, y_3, u_3), where l is the height and base of the finite element. Knowing that

$$\tilde{u}(x, y) = c_0 + c_1 x + c_2 y \tag{91}$$

we can insert our three nodes into $\tilde{u}$ and write down:

$$\begin{cases} u_1 = c_0 + c_1 x_1 + c_2 y_1 \\ u_2 = c_0 + c_1 x_2 + c_2 y_2 \\ u_3 = c_0 + c_1 x_3 + c_2 y_3 \end{cases} \tag{92}$$

Solving the system is equivalent to computing:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{93}$$

which results in:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \frac{1}{d} \begin{bmatrix} p_1 + p_2 + p_3 \\ q_1 + q_2 + q_3 \\ r_1 + r_2 + r_3 \end{bmatrix} \tag{94}$$

with:

$$p_i = x_j y_k - x_k y_j \tag{95}$$

$$q_i = y_j - y_k \tag{96}$$

$$r_i = -x_j + x_k \tag{97}$$

$$d = x_1 y_2 - x_1 y_3 - x_2 y_1 + x_2 y_3 + x_3 y_1 - x_3 y_2 \tag{98}$$

where $i, j, k \in \{1, 2, 3\} \wedge i \neq j \neq k$. Inserting the coefficients into equation (91) and developing the expression to obtain an expression under the form of equation results in(27):

$$\tilde{u} = u_1 N_1(x) + u_2 N_2(x) + u_3 N_3(x) \tag{99}$$

with:

$$N_i(x) = \frac{1}{2d}(p_i + q_i x + r_i y) \tag{100}$$

A.4 Heat transfer of a bar using second order 1D finite element

Take the weak formulation of the governing differential equation:

$$\left[W_i \frac{d\tilde{u}}{dx} \right]_\Omega - \int_\Omega \frac{dW_i}{dx} \frac{d\tilde{u}}{dx} \, d\Omega - k' \int_\Omega W_i \tilde{u} \, d\Omega = -k' T_\infty \int_\Omega W_i \, d\Omega \tag{101}$$

We take $\tilde{u}$ and W_i corresponding to the second order finite element and define Ω as the space of the finite element:

$$\tilde{u}(x) = \begin{bmatrix} N_1(x) & N_2(x) & N_3(x) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{102}$$

$$W_i = N_i(x) \tag{103}$$

It is advantageous to compute the derivatives of the Weight and the approximate function in advance:

$$\frac{d}{dx}\tilde{u}(x) = \begin{bmatrix} \frac{d}{dx}N_1(x) & \frac{d}{dx}N_2(x) & \frac{d}{dx}N_3(x) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{104}$$

$$\frac{d}{dx}W_i = \frac{d}{dx}N_i(x) \tag{105}$$

A more compact notation gives:

$$\tilde{u} = [N][u] \tag{106}$$

$$\frac{d}{dx}\tilde{u} = \left[\frac{dN}{dx} \right][u] \tag{107}$$

This allows us to rewrite the weak formulation (101) as:

$$\int_\Omega \left[\frac{dN}{dx} \right]^T \left[\frac{dN}{dx} \right][u] \, d\Omega + k' \int_\Omega [N]^T [N][u] \, d\Omega = k' T_\infty \int_\Omega [N] \, d\Omega \tag{108}$$

which can be simplified to:

$$\int_\Omega \left[\frac{dN}{dx} \right]^T \left[\frac{dN}{dx} \right] + [N]^T [N] \, d\Omega [u] = k' T_\infty \int_\Omega [N] \, d\Omega + Q \tag{109}$$

Knowing from (38 - 40) that:

$$[N] = \begin{bmatrix} 1 - \frac{3}{l}x + \frac{2}{l^2}x^2 \\ -\frac{1}{l}x + \frac{2}{l^2}x^2 \\ \frac{4}{l}x + \frac{4}{l^2}x^2 \end{bmatrix} \tag{110}$$

$$\left[\frac{dN}{dx} \right] = \begin{bmatrix} -\frac{3}{l} + \frac{4}{l^2}x \\ -\frac{1}{l} + \frac{4}{l^2}x \\ \frac{4}{l} + \frac{8}{l^2}x \end{bmatrix} \tag{111}$$

We can compute $\int_\Omega [\frac{dN}{dx}]^T [\frac{dN}{dx}] \, d\Omega$, $\int_\Omega [N]^T [N] \, d\Omega$ and $\int_\Omega [N] \, d\Omega$:

$$\int_\Omega [\frac{dN}{dx}]^T [\frac{dN}{dx}] \, d\Omega = \begin{bmatrix} 7/(3l) & 1/(3l) & -20/(3l) \\ 1/(3l) & 7/(3l) & 52/(3l) \\ -20/(3l) & 52/(3l) & 496/(3l) \end{bmatrix}$$

$$\int_\Omega [N]^T [N] \, d\Omega = \begin{bmatrix} 2l/15 & -l/30 & -2l/15 \\ -l/30 & 2l/15 & 28l/15 \\ -2l/15 & 28l/15 & 512l/15 \end{bmatrix}$$

$$\int_\Omega [N] \, d\Omega = \begin{bmatrix} l/6 \\ l/6 \\ 14l/3 \end{bmatrix}$$

The coefficient matrix A ends up being:

$$\begin{bmatrix} (2k'l^2+35)/(15l) & (-k'l^2+10)/(30l) & 2(-k'l^2-50)/(15l) \\ (-k'l^2+10)/(30l) & (2k'l^2+35)/(15l) & 4(7k'l^2+65)/(15l) \\ 2(-k'l^2-50)/(15l) & 4(7k'l^2+65)/(15l) & 16(32k'l^2+155)/(15l) \end{bmatrix} \tag{112}$$

The variable vector ends up being the nodal values $[u]$ and the constant vector b equals:

$$b = k'T_\infty \begin{bmatrix} l/6 \\ l/6 \\ 14l/3 \end{bmatrix} \tag{113}$$

The local matrix then is written as:

$$A[u] = b \tag{114}$$

B APPENDIX: DIAGRAMMS

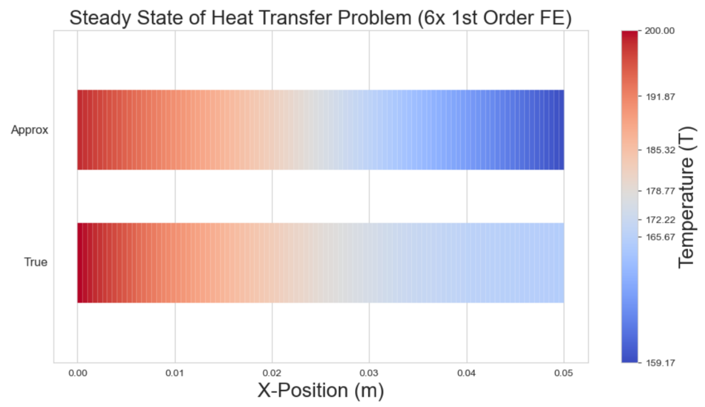


Figure 17: Comparison between the approximate and true steady state of the toy problem with six 1st order finite elements

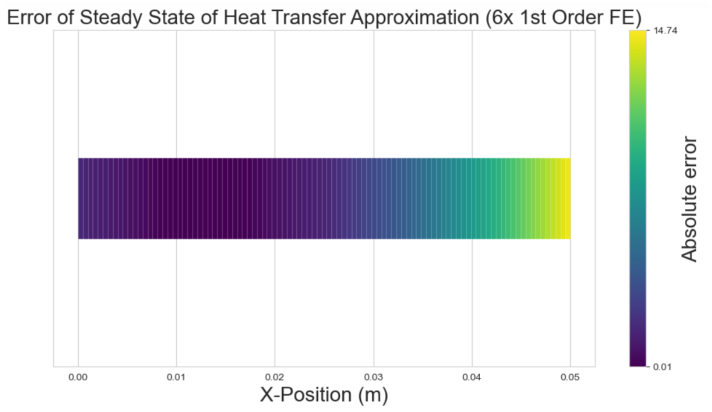


Figure 18: Error distribution for the approximation with six 1st order finite elements