unibz
Fakultät für Ingenieurwesen
Facoltà di Ingegneria
Faculty of Engineering

Bachelor in Computer Science

Bachelor Thesis

A high-performance semi-implicit numerical method for the simulation of compressible flows in systems of compliant elastic pipes

Candidate: Luca Boscarato

Supervisor: Prof. Maurizio Tavelli
Co-supervisor: Prof. Bruno Carpentieri

2024

To my mother,
who has always been with me and always will be.

Abstract

The objective of this thesis is to implement a semi-implicit numerical method for the simulation of compressible flows in systems of compliant elastic pipes. Detecting and locating water leakages in pipes is nowadays one of the many challenges society has to face daily, hence the interest that led to the project becoming part of the PRIN program. The proposed solution is to use machine learning models based on neural networks to assess the leakages in the system of pipes. However, since such models require a dataset with a vast variety of samples to train the network, this thesis focuses on implementing an efficient method to generate the dataset. In particular, the user only has to specify the network parameters, as well as its structure and the location of the leakages, and run the provided code. The discretization of the partial differential equations governing the phenomenon leads to a nonlinear system of equations, which is solved by using the Nested Newton Method. The nonlinear system is decomposed into a sequence of linear systems, which are tackled by using the Conjugate Gradient Method given their Symmetric Positive Definite structure. To further accelerate the speed of convergence of the method, a custom preconditioner is introduced. Finally, the numerical method is implemented in Fortran, to exploit vectorialization and efficient memory management, and CUDA, to take advantage of parallelization on the GPU. The results show that the method can generate the dataset for a given set of networks and their parameters, both in the case of single pipes and more complex networks. Further testing on real-world scenarios can be simulated by modifying the parameters in the code accordingly. The performances of the method are analyzed, also in comparison with a previous version implemented in MATLAB, to highlight the improvements and optimizations this new version offers. The generation process is shown to be up to 90 times faster than the previous version. Moreover, performances increase further when parallel programming techniques are introduced. The results obtained are consistent with what the literature states. Also, since in the future the idea is to introduce systems operating in real-time, this work represents a relevant step towards the final goal.

Contents

Chapter 1

Introduction

In this thesis, we develop a semi-implicit numerical method for the generation of a dataset in the context of systems of compliant elastic pipes. This work is part of the PRIN project entitled *"Hybrid Transient-Machine Learning Approach for Anomaly Detection and Classification in Water Transmission Mains (TANDEM)"* (CUP: J53D23002110006). In particular, since the machine learning models need a dataset to be trained, we focus on developing a numerical method to generate them efficiently.

1.1 Structure of the thesis

- Chapter 1: General introduction to the problem and approaches used;
- Chapter 2: Background material on physical properties taken into consideration, as well as a background on the numerical and linear algebra concepts used;
- Chapter 3: Discussion on the numerical method and the techniques used to enhance performances;
- Chapter 4: Evaluation of the results both in terms of correctness and efficiency;
- Chapter 5: Conclusions and discussion on the future work.

1.2 Motivation

Currently, to locate fractures in systems of pipes, pressure waves are generated and fed into the pipe, propagating along the tube at high speeds. When they encounter a leak, part of the wave is reflected while the rest continues to propagate. By placing sensors along the pipe, operators analyse variations in the pressure wave signal. Additionally, using the arrival time of the reflected waves and the speed of wave propagation in the tube, the exact location of the leak could theoretically be guessed. Practically, however, this is feasible only for a single pipe or simple networks. In complex real-world systems, it is not trivial to interpret the signals, even with a high number of sensors, hence why machine learning models are used.

1.3 Approach

The numerical method is semi-implicit, meaning that some factors are taken as implicit while others are considered explicitly. In such a way, a tradeoff between the precision of the results and the performances of the method is reached. Furthermore, by taking a factor as implicit, a larger time-step size can be set. The goal is to simulate a real-world scenario of a network with the given parameters. To do this, we need to encode the majority of the physical properties governing the phenomenon and

make sure that all the equations are valid simultaneously, namely, we need to build a system of equations. The system is built by discretizing the tube, that is we subdivide the single pipe or system of pipes into N_s different chunks. Given the complexity of the equations, the system is nonlinear, and therefore computationally hard to solve. To solve the system we use a Nested-Newton method. In particular, at each iteration, we solve the linearized system. However, in this specific case, the matrix form of the system follows the definition of SPD (Symmetric and Positive Definite) matrix. As a consequence, efficient methods such as the Conjugate Gradient Method can be used to solve the linear system and retrieve the solutions. The formers represent the values of the parameters that make all the equations simultaneously valid, and therefore the values in a real-world scenario. For the simulation provided in this thesis, the parameters we analyse are the pressure inside the pipe, as well as the density, velocity, mass, and volume of the fluid and the hydraulic head, together with the properties of pipes and materials, such as equilibrium radius, elasticity coefficients, etc. The method is implemented in Fortran since the original idea was not only to produce a working numerical method but also an efficient one. For this reason, parallel-programming techniques are also used, as well as a custom preconditioner, specifically designed for this kind of discretization.

1.4 State of the art and contributions

The current state of the art involves pressure-based approaches. Moreover, these are limited to analysing cases of individual segments without fractures. In this work, however, the terms are formulated as a function of piezometric height. Furthermore, we are not limited to the case of the single segment, which is nevertheless analysed to confirm the results of the thesis, but networks of pipes are also introduced. Finally, there is the possibility of fractures within the system.

Chapter 2

Background

2.1 Phyisical properties

In this chapter, some fundamental hydraulics and fluid dynamics concepts are discussed. To retrieve further details on them, refer to [1], [2] and [3].

Compressible flows

The numerical method has been developed starting from the equations governing the phenomenon when the fluid is compressible. For a fluid to be considered compressible, it means that its density ρ can change significantly as a result of a change in pressure or temperature. In this case study, the fluid taken into consideration is water. Although not highly compressible, water can be considered a compressible fluid, especially when high pressures are met, therefore, equations for compressible fluids are used.

Piezometric Head

The piezometric head is a fundamental concept in hydraulics and fluid mechanics, used to describe the potential energy of a fluid in motion or a piping system. Starting from the formula of the pressure:

$$p = p_{ext} + \rho * g * (\eta - z), \tag{2.1}$$

it can be computed as follows

$$\eta = \frac{p - p_{ext}}{\rho g} + z, \tag{2.2}$$

in which, p represents the pressure exerted by the fluid, z is the geodetic height, namely the height compared to a fixed and predetermined point in space, g is the gravitational constant, ρ is the density of the fluid, and p_{ext} is the external pressure.

Elastic pipes

The pipes used for the simulation are elastic pipes, meaning that their volume and shape may vary over time as a result of the application of some forces. Notice that as soon as those forces are no longer applied, the pipe will return to its original shape and volume. Pipes must be able to withstand water pressure without bursting or deforming in a plastic way. Elastic materials can withstand pressure variations due to their ability to expand and contract. These variations will be reflected in the numerical method by computing the radius of the pipe at each iteration, even though no major variation is expected. However, since the deformation is critical if we want to see the correct wave propagation, the precision of the code has to be high.

Networks

The representation of the system of pipes, both for the case of single pipes or more complex networks, is covered with a staggered grid (first presented in [4]). In particular, the network is subdivided into a total of N_s segments and N_{Ep} nodes. The length of the segment Δx_j is considered either uniformly or nonlinearly, according to the needs of the user. Every internal node j separates two segments. We can assign an arbitrary direction and hence find a left ($l(j)$) and right ($r(j)$) segment, according to the normal vector. Throughout the work, those two segments will be referred to as *neighbors* of node j.

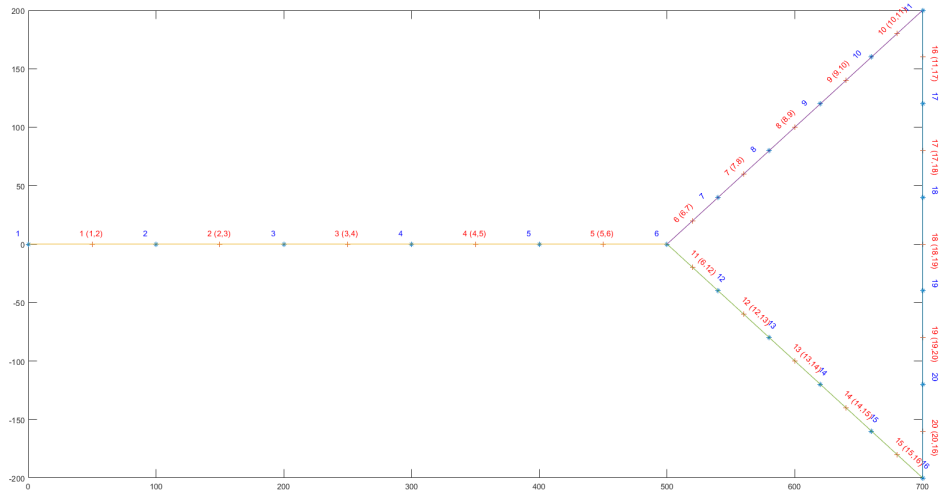

Figure 2.1: Example mesh of the Simple Network (4.1.2). See also the next figures.

At time t_n, the discrete velocity u_j^n is located at x_j, which represents the point at the center of the jth segment. As presented in the Approach chapter, the contributions from the *neighbors* will also be considered in the computation. Additionally, we have to set a direction for the vector u_j^n; we assume it goes from $l(j)$ to $r(j)$. The area of the pipe at x_j, correspondent to the constant pressure p_j^n, is computed as follows

$$A_j^n = 2\pi R(\eta_j^n)^2, \tag{2.3}$$

where R is the radius of the pipe, computed as a function of pressure and a rigidity coefficient β as shown in (3.8). As one could expect, the area A_j^n is always non-negative. The discrete pressure p_i^n is located at a segment endpoint and its respective spacial coordinate is identified by $x_i = 1, 2, .., N_{Ep}$.

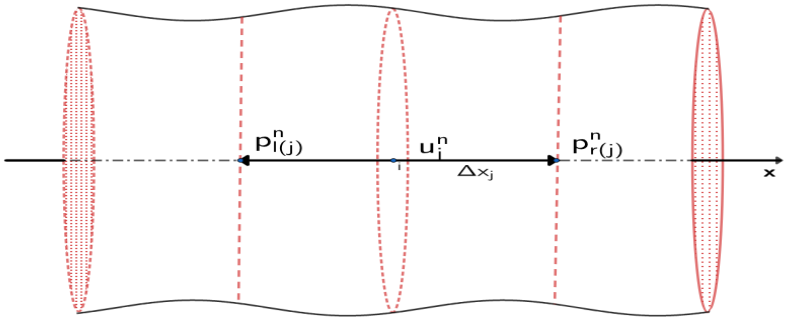

Figure 2.2: Representation of the jth segment in a pipe [4].

We refer as S_i to the set of segments that share the ith pressure point. Generally, the size of the set is

not predefined. For a single pipe, the size of the set will always be 2, namely the two consecutive segments within the same branch that share the ith pressure point, but this is not the case if a junction is present. The only exceptions are the endpoints of the system (i.e. the ends of the tube), for which $|S_i| = 1$.

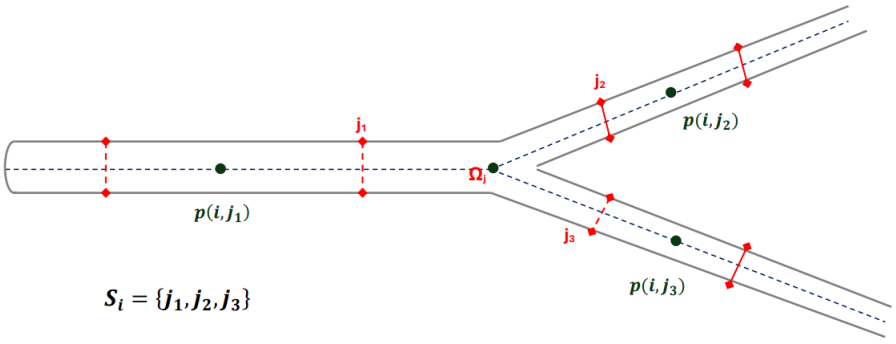

Figure 2.3: Sketch of a junction [4]. Check Evaluation chapter for plots.

Shallow-Water equations

The Shallow-Water equations (SWE) are a set of hyperbolic PDEs (partial differential equations) derived from the Navier-Stokes equations that describe fluid flow in contexts where the wavelength of fluid motion is much greater than the depth of the fluid itself. The 1D shallow water equations are:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} = 0, \tag{2.4}$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial \left(hu^2 + \frac{1}{2}gh^2\right)}{\partial x} = -hg\frac{\partial h}{\partial x}, \tag{2.5}$$

where:

- $h(x,t)$ represents the water depth;
- $u(x,t)$ represents the depth-averaged velocity;
- g represents the acceleration due to gravity.

The 2D shallow water equations are:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0, \tag{2.6}$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial (hu^2 + \frac{1}{2}gh^2)}{\partial x} + \frac{\partial (uv)}{\partial y} = -hg\frac{\partial h}{\partial x}, \tag{2.7}$$

$$\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial (hv^2 + \frac{1}{2}gh^2)}{\partial y} = -hg\frac{\partial h}{\partial y}, \tag{2.8}$$

where:

- $h(x,y,t)$ represents the water depth;
- $u(x,y,t)$ represents the depth-averaged velocity in the x-direction;
- $v(x,y,t)$ represents the depth-averaged velocity in the y-direction;
- g represents the acceleration due to gravity.

In both cases, continuity of mass and momentum is established [5].

2.2 Numerical Method

This section presents all the concepts needed to understand the choices made for the numerical method. Furthermore, discretization techniques will be introduced starting from a simpler problem than that of shallow-water equations. All the information were taken from [6].

Numerical discretization

To introduce some simple finite difference approximations, let us restrict ourselves to a simple linear advection problem, which still describes the transport of a scalar quantity through a fluid like the Shallow-Water Equations (2.1), but without accounting for factors like varying water height or gravitational forces. The problem is the following:

$$\begin{cases} \frac{\partial q}{\partial t} + a\frac{\partial q}{\partial x} = 0, x \in \mathbb{R}, t \in \mathbb{R}_0^+, \\ q(x,0) = k(x), \end{cases} \tag{2.9}$$

where $q = q(x,t)$ is the solution of the PDE, $k(x)$ represents the initial condition at $t = 0$ and $a \in \mathbb{R}$ is a constant coefficient. We now discretize the proposed PDE. First, we recall the definition of derivative:

$$\frac{\partial q}{\partial t} = \lim_{\Delta t \to 0} \frac{q(x, t+\Delta t) - q(x,t)}{\Delta t}, \tag{2.10}$$

$$\frac{\partial q}{\partial x} = \lim_{\Delta x \to 0} \frac{q(x+\Delta x, t) - q(x,t)}{\Delta x}. \tag{2.11}$$

Notice that we considered the derivate from the right, but one could also take the derivative from the left and compute $\frac{\partial q^-}{\partial t}$ and $\frac{\partial q^-}{\partial x}$. Nevertheless, in finite arithmetic, $\Delta x \to 0$ and $\Delta t \to 0$ do not exist; we can only rely on *finite difference* approximation. Therefore, the numerical derivative of (2.10) becomes the following

$$\frac{\partial q}{\partial t} \approx \frac{q_i^{n+1} - q_i^n}{\Delta t}, \tag{2.12}$$

with the definition $q_i^n = q(x_i, t^n)$. As for the space derivative (2.11), we have multiple schemes at disposal:

$$\frac{\partial q}{\partial x} \approx \frac{q_{i+1}^n - q_i^n}{\Delta x} \quad \text{forward finite difference,} \tag{2.13}$$

$$\frac{\partial q}{\partial x} \approx \frac{q_i^n - q_{i-1}^n}{\Delta x}, \quad \text{backward finite difference,} \tag{2.14}$$

$$\frac{\partial q}{\partial x} \approx \frac{q_{i+1}^n - q_{i-1}^n}{2\Delta x}, \quad \text{central finite difference.} \tag{2.15}$$

One has to choose which scheme to use based on the availability of data, as well as the accuracy, numerical stability, and computational efficiency required, and the type of differential equation that has to be solved. Since in this case we can only observe current or past values and we have to solve hyperbolic equations, the *upwind* discretizations provide the right choice. Otherwise, the scheme would be easily unstable, i.e. unphysical maximum and minimum would be generated, as demonstrated in the papers of Casulli [5][7]. The same scheme is used to discretize the PDEs presented in the next chapter (3.1). Back to the linear advection problem, a simple stable discretization of (2.9), assuming $a > 0$, reads

$$q_i^{n+1} = q_i^n - a\frac{\Delta t}{\Delta x}(q_i^n - q_{i-1}^n). \tag{2.16}$$

Let us now introduce some concepts and algorithms which will be useful in the next chapter.

Tridiagonal Matrices

A real matrix A is said to be tridiagonal if only non-zero entries appear on the diagonal or the lines above and below the former. Additionally, the matrix has to be square. Hence, this is how it should look like

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_n \end{bmatrix}$$

Symmetric and Positive Definite Matrices

A real matrix A is said to be positive definite if

$$u^\top Au > 0 \quad \text{for all} \quad u \neq 0, u \in \mathbb{R}^n. \tag{2.17}$$

Many properties can be derived as a consequence of a matrix being SPD, but the most relevant for us in this study is that the eigenvalues of an SPD matrix are all (real) positive [6]. Additionally, the following properties hold for a given matrix A which is SPD:

- A is nonsingular;
- Diagonal entries of A are always positive;
- Each k-th principal submatrix A_k is SPD;
- $Det(A_k) > 0$ for $k = 1, .., n$.

Conjugate Gradient Method

The Conjugate Gradient Method is an iterative method for solving a linear system of equations $Ax = b$, where A is an $n \times n$ Symmetric and Positive Definite matrix. Unlike the Descent Method, the CG is a semi-iterative method, since the number of iterations is limited by the dimension of the system. This is because the directions are mutually conjugate and after N iterations, they represent a basis of the space $\mathbb{R}^N$, hence the exact solution. The algorithm requires the user to make a first guess for the value of the solution. It starts by computing the residual of the system at the first iteration and continues until the residue is lower than some user-defined threshold (tol). Furthermore, a maximum number of iterations is set to limit them in more complex cases. At each iteration, we compute the step size α, update the solution estimate, compute the new residual and finally we check for convergence. If convergence has not been reached yet, meaning that the residual is too large compared to the given threshold, we compute the conjugate direction coefficient β and we update the search direction vector d [8].

Algorithm 1 Conjugate Gradient Method

procedure CGOP(A, b, x_1, tol, max_iter)
  $r_1 \leftarrow b - Ax_1$
  $d_1 \leftarrow -r_1$
  $i \leftarrow 1$
  while $i \leq$ max_iter and $||r_i|| >$ tol **do**
    $\alpha_i \leftarrow \frac{r_i^T r_i}{d_i^T A d_i}$
    $x_{i+1} \leftarrow x_i + \alpha_i d_i$
    $r_{i+1} \leftarrow b - Ax_{i+1}$
    if $||r_{i+1}|| \leq$ tol **then**
      break
    $\beta_i \leftarrow \frac{||r_{i+1}||^2}{||r_i||^2}$
    $d_{i+1} \leftarrow -r_{i+1} + \beta_i d_i$
    $i \leftarrow i + 1$
  return x_{i+1}

Newton-Raphson

Newton's Method, also known as the Newton-Raphson method, is an iterative numerical method used to approximate the roots (or zeroes) of a real-valued function [9]. The algorithm requires an initial guess x_0 for the root of the function. At each step, a check is made on whether the current value for x is accurate enough as root, according to a user-defined threshold. If not, the process is repeated as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.18}$$

Or equivalently, in matrix form

$$x_{n+1} = x_n - f'(x_n)^{-1} f(x_n). \tag{2.19}$$

For the case $f(x_n) = Ax_n - b$,

$$\begin{aligned} f'(x_n) &= A, \\ x_{n+1} &= x_n - A^{-1}(Ax_n - b), \\ x &= A^{-1}b. \end{aligned} \tag{2.20}$$

Other root-finding methods exist (such as the bisection method, secant method, etc.), but in this case we use the Newton method because it offers quadratic convergence. However, note that while in our scenario the derivative of the function is easy to compute and therefore we can choose this method, in other scenarios it could also not be the case. Additionally, we can ensure convergence in this case, which is not trivial for a general function f and every x_0.

Inexact Newton Method

The Inexact Newton Method, also known as the Newton-Krylov Method, is an enhancement of the classical Newton-Raphson method tailored to solve nonlinear systems of equations [10]. In our case, at each step of the method, a linear system will be solved using the CG method (2.2), since the matrix is SPD, and the solutions for the linear system will be used to compute the solution of the starting nonlinear system using a Nested Newton method.

Preconditioner

A preconditioner is a mathematical tool used to improve the efficiency and speed of convergence of iterative algorithms in solving linear systems of equations, particularly when the system is large and ill-conditioned. A generic matrix A is said to be ill-conditioned if the condition number is very large.

In our scenario, the matrix A is an SPD matrix, hence invertible. The condition number is therefore computed as the ratio of the largest to the smallest eigenvalue of matrix A in terms of their norms:

$$k_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\max\limits_{i=1,\dots,n} |\lambda_i|}{\min\limits_{i=1,\dots,n} |\lambda_i|}. \tag{2.21}$$

In our case, we will use a preconditioner based on the Thomas algorithm since the matrix A in the linear system is not only SPD but also mostly tridiagonal.

Thomas algorithm

The Thomas algorithm is an efficient way to solve systems of tridiagonal matrices. It is based on the LU decomposition in which the system $Ax = b$ is rewritten as $LUx = b$, where L is the lower triangular matrix and U is the upper triangular matrix. By considering $Ux = y$, one can solve $Ly = b$ for y. Once y is found, we solve the system $Ux = y$ for x and retrieve the original solution to the system. When the matrix is tridiagonal, this method is preferred given its complexity of $\mathcal{O}(n)$ [11]. We use the Thomas Algorithm to solve the system $Ax = b$, in which:

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_n \end{bmatrix}, b = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Algorithm 2 Thomas Algorithm

1: **procedure** THOMAS(A, b)
2: $n \leftarrow \text{length}(b)$
3: $c'_1 \leftarrow \frac{c_1}{b_1}$
4: $d'_1 \leftarrow \frac{d_1}{b_1}$
5: **for** $i = 2$ to n **do**
6: **if** $i < n$ **then**
7: $c'_i \leftarrow \frac{c_i}{b_i - a_i c'_{i-1}}$
8: $d'_i \leftarrow \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}$
9: $x_n \leftarrow d'_n$
10: **for** $i = n - 1$ to 1 **do**
11: $x_i \leftarrow d'_i - c'_i x_{i+1}$
12: **return** x

After the initialization of the first elements, the forward elimination process is done. The algorithm iterates over all rows, modifying the coefficients c'_i and d'_i, effectively transforming the matrix into upper triangular form. Consequently, the second loop performs the backward substitution, effectively computing the solution vector x.

Chapter 3

Approach

3.1 Mathematical model

Computational fluid mechanics is a relevant field for a vast set of applications, that vary from aerospace and mechanical engineering to blood flow in the human cardiovascular system. However, despite the number of applications, the models governing each phenomenon are all derived from the same equations, namely the Navier-Stokes equations [12]. It is important to recall that depending on the context analysed, for instance, whether the considered fluid is incompressible or compressible, or other properties of the fluid, such as viscosity, the derived model may differ. In our case, the fluid taken into consideration is water, which is normally an incompressible fluid, but when high pressures are reached, it becomes compressible. Furthermore, the pipes through which water flows are elastic, hence why the volume of the pipe may vary. In vectorial form, the system is as follows:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \\ \frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho u \otimes u) + \nabla p = \nabla \cdot \sigma \end{cases} \tag{3.1}$$

The first equation is the continuity equation for mass conservation. The second one establishes the conservation of momentum. In particular, ∇ refers to the gradient operator, $\nabla\cdot$ denotes the divergence operator, and the Kronecker operator $\otimes$ is used to express the nonlinear convective terms in the momentum equation. In the system:

- ρ is the fluid density;
- u is the velocity vector;
- p is the pressure.

Applying the techniques introduced in the previous chapter, as well as some calculus, we discretize the first equation. First, we integrate over a control volume V_i around each point i and use the divergence theorem, to obtain

$$\int_{V_i} \frac{\partial \rho}{\partial t}\, dV - \int_{\partial V_i} (\rho u) \cdot \hat{n} dS = 0,\ \chi(x) = \begin{cases} 1 & \text{if } x \text{ is in the tube,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

in which $\hat{n}$ denotes the outward pointing unit normal vector on ∂V_i. We now approximate the integrals over the control volume V_i in discrete form as follows

$$\int_{V_i} \frac{\partial \rho}{\partial t}\, dV \approx \int_{V_i} \frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = \frac{M_i^{n+1} - M_i^n}{\Delta t} \text{ and } \int_{\partial V_i} (\rho u) \cdot \hat{n} dS \approx \sum_{j \in S_i} A_j \rho_j u_j, \tag{3.3}$$

where M_i^n is the mass defined as $\int_{V_i} \rho^n \chi(x) dx$. Combining these two expressions, we obtain

$$\rho_i^{n+1} \Delta V_i = \rho_i^n \Delta V_i - \Delta t \sum_{j \in S_i} A_j (\rho_j u_j) + \int_{\Omega_f} (\rho u) \cdot n_r dS, \tag{3.4}$$

where n_r is the normal in the radius direction and Ω_f is the leak size. It simplifies to

$$M_i^{n+1} = M_i^n + \Delta t \sum_{j \in S_i} A_j^n U_j^{n+1} - S_i^f. \tag{3.5}$$

By following a similar approach, the second equation can also be discretized. Using a finite difference approach, the time-discretized version of the previous system using the upwind scheme is the following

$$M_i^{n+1} = M_i^n + \Delta t \sum_{j \in S_i} A_j^n U_j^{n+1} - S_i^f, \tag{3.6}$$

$$U_j^{n+1} = F_U^n - \frac{g \Delta t}{\Delta x} \left(P h_{r(j)}^{n+1} - P h_{\ell(j)}^{n+1} \right) - F_f^n U_j^{n+1}. \tag{3.7}$$

In the system above:

- M_i^n: Non-linear mass at the ith segment at time step n;
- Δt: Time step representing the interval between time step n and $n+1$;
- $\sum_{j \in S_i}$: Sum over all segments j that are neighbors of node i;
- A_j^n: Cross-sectional area of the pipe of segment j at time n;
- U_j^{n+1}: Velocity or flow rate through segment j at time step $n+1$;
- S_i^f: Source term at node i. This term includes all external factors, but mainly the mass loss due to leaks;
- F_U^n: Non-linear convective term for segment j at time step n;
- Δx: spacial discretization representing the distance between $l(j)$ and $r(j)$;
- $P h_{r(j)}^{n+1}$ and $P h_{l(j)}^{n+1}$: The pressure head at the right and left ends of segment j at time step $n+1$. These terms are used to calculate the pressure gradient in segment j. Recall that $Ph = \rho \eta$, in which η is the piezometric head and ρ is the density;
- F_f^n: Fracture flow resistance factor at time step n.

The radii of the tube is computed as a function of pressure and a rigidity coefficient β as

$$R(\eta) = R_0 + \frac{\rho_0 * g}{\beta} (\eta - (\eta_0 + R_0)). \tag{3.8}$$

The density ρ is also computed in terms of pressure using a mixture formulation, see [13] for more details. Substituing the second equation (3.7) into the first one (3.6), we obtain the expression

$$M_i^{n+1} + \frac{g \Delta t^2}{\Delta x} \sum_{j \in S_i} \frac{1}{1 + F_{f,j}} A_j^n \left(P h_{r(j)}^{n+1} - P h_{\ell(j)}^{n+1} \right) = M_i^n + \Delta t \sum_{j \in S_i} \frac{1}{1 + F_{f,j}} A_j^n F_U^n - S_i^f. \tag{3.9}$$

It can be shown that, given a stable discretization of F_u, i.e. using a classical upwind approach or a semi-lagrangian scheme, this approach results unconditionally stable for any choice of Δt. In particular, (3.9) represents a mildly nonlinear system for the unknown Ph^{n+1}. The key concept here is that the only non-linear term is M^{n+1}, however, since it only appears on the diagonal and we know that it is a monotonically increasing function with respect to Ph, we can use a Nested Newton method to solve the system. Once Ph^{n+1} is known, the momentum can be computed using (3.7). Note that if M is convex, a newton is sufficient, otherwise, a decomposition of the concave-convex part is needed, together with a nested approach [7].

Numerical decomposition of the non-linear operator

The solution of the linear system is η and we now know also the mass M. Therefore, we can compute the volume starting from the mass equation

$$M = \rho V,\ \tilde{V} = \rho/\rho_0 V. \tag{3.10}$$

To lighten the notation, from now on we will refer to $\tilde{V}$ as V. To discretize the non-linear term volume we start with the following equation

$$V(\eta) + T(\eta) = b, \tag{3.11}$$

in which, T is an SPD matrix. Numerically,

$$V_i(\eta) = V_i^{(1)}(\eta) - V_i^{(2)}(\eta), \tag{3.12}$$

wherein

$$V_i^{(1)}(\eta) = \int_{-\infty}^{\eta} P_i(x)dx, \quad V_i^{(2)}(\eta) = \int_{-\infty}^{\eta} Q_i(x)dx. \tag{3.13}$$

In the equations above, $P(\eta)$ and $Q(\eta)$ are diagonal matrices; the diagonal entries are the terms $p_i(\eta_i)$ and $q_i(\eta_i)$ [13] [14].

3.1.1 Fracture

When a fracture occurs, it can have significant consequences on the water system. Regarding the pressure and velocity of the fluid, one expects both to decrease since part of the water volume flows through the leak and gets lost. Consequently, a mass loss is also likely. Here, the mass loss is written in terms of the pressure as an exponential function of factor $\alpha > 0$.

$$f(p) = cp^{\alpha} \qquad p = p_{ext} + \rho * g * (\eta - z). \tag{3.14}$$

Since it represents a mass loss (i.e. it would have a negative sign in the mass contribution) that can lead to instability, we perform a simple time linearization procedure centered in t^n to extract a linear contribution

$$f(p) \approx c(p^n)^{\alpha} + c\alpha(p^n)^{\alpha-1}(p^{n+1} - p^n). \tag{3.15}$$

Using the relation between the pressure and the hydraulic head, we obtain

$$f(\eta) \approx f^n + \rho g c\alpha \left(p^n\right)^{\alpha-1} \eta^{n+1}, \tag{3.16}$$

where $f^n = c(p^n)^{\alpha} - \rho g c\alpha \left(p^n\right)^{\alpha-1} \eta^n$. It is noted that since α, p, ρ and g are greater than zero, the contribution would enforce the diagonal of the resulting linear system.

3.1.2 Demand from the tanks

An arbitrary number of tanks can be placed in the system. Each tank can request part of the water volume from the system. However, water cannot be requested all at once. For this purpose, we make use of an error function. Given the input parameters:

- a (scaling parameter);
- atime (time constant for the error function);
- time (current time);
- t_0 (reference time).

The function $\alpha(\text{time})$ is defined as $\alpha(\text{time}) = f(\text{time}) + g(\text{time})$, where $f(\text{time})$ and $g(\text{time})$ are defined as:

$$f(\text{time}) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{\text{time}}{5} - 2\right)\right), \qquad g(\text{time}) = \frac{1}{2}(a_L + a_R) + \frac{1}{2}(a_R - a_L)\,\text{erf}\left(\frac{4\,(\text{time} - t_0)}{\text{atime}}\right), \tag{3.17}$$

in which,

$$a_L = 0, \qquad a_R = a - 1. \tag{3.18}$$

Thus, the final expression for $\alpha(\text{time})$ is:

$$\alpha(\text{time}) = \frac{1}{2}(a_L + a_R) + \frac{1}{2}(a_R - a_L)\,\text{erf}\left(\frac{4\,(\text{time} - t_0)}{\text{atime}}\right) + \frac{1}{2}\left(1 + \text{erf}\left(\frac{\text{time}}{5} - 2\right)\right).$$

The function looks as follows:

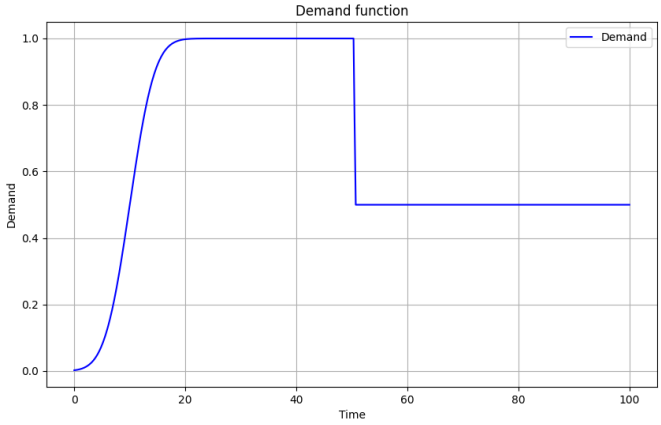

Figure 3.1: Demand function

Additionally, demand points can be set by the user through a time series as well.

3.2 Structure of the code

The code is structured in modules, as usual for Fortran projects, as follows:

File	Description
`start_simulation.f90`	This file can be considered the main program.
`PARAM.f90`	In this file, all the parameters regarding the simulation are set.
`linear_algebra.f90`	This file contains all algebra-related functions called in the other modules.
`physics.f90`	This file contains all physics-related functions, such as the computation of pressure, density, etc.
`Network.f90`	This file contains all the network-related functions, both to define and initialize a network of pipes.
`NumSolution.f90`	This file contains all the properties related to the numerical solution, such as the maximum number of iterations for the nested Newton, etc.

Table 3.1: Structure of the code

3.3 Parameters of the simulation

Before starting the simulation we have to initialize and define the properties. In particular, we have to define the network of pipes and the parameters of both the simulation and the numerical method.

Network of pipes

The network of pipes can be imaged as an undirected graph. Each node of the graph is a point on the pipe. When we discretize the pipe (or system of pipes), the number of segments into which the pipe is subdivided will be referred to as N_s, while the number of nodes will be referred to as N_{Ep}. For each node, we also have to define which are the "neighbor" nodes (i.e. the ones present in the set S_i of the i_{th} pipe segment). Additionally, parameters related to the system have to be set, such as the demand from the system, the degradation coefficient of the pipe and the external pressure. On the current version of the simulator, each segment can have a different value for the degradation, since pipes in a system can also be different, and a demand point can be placed anywhere along the network.

Parameters of the simulation

The parameters of the simulation are the following:

- Start/End time of the simulation;
- Start/End of recording time of sensors and their positions along the pipe;
- Delta time per iteration (arbitrary due to the unconditionally stable property);
- Initial fluid condition (velocity, pressure, and density);
- Gravitational constant g;
- Time and position of the fracture along the pipe;
- Position of tanks along the pipe and their demand, expressed in percentage;
- Other network-specific parameters that impact the simulation.

Parameters of the numerical method

The parameters of the numerical method are the following:

- Maximum number of iteration for the solver;
- Coefficient a_0/a_1 for the linear interpolation;
- Other network-specific parameters that have an impact on the numerical method.

3.4 Nested Newton

Once the parameters have been set correctly, the simulation can start. In particular, the time starts at 0 seconds and is incremented at each iteration by Δt. If the recording time has been reached, then the delta time specified in the parameters is used instead of the default one for the network, to measure more accurately the results in the specified interval and save them in the RES object, which contains all the results of the simulation. At each iteration, the radius, area and volume of the pipe is updated, as well as the density of the fluid and the hydraulic head. Based on the computation of these quantities, we now start to build the system of equations to solve. Recall that we want to solve a system of the form $Ax = b$. Hence, we need to specify both the matrix A and the vector of known terms b.

Construction of the right-hand side vector b

For the construction of the vector b, we also take into consideration the following factors:

- Viscous dissipation of the fluid (accumulator1);
- Impact of external forces or sources on the system (accumulator2);
- Pressure Gradient Term (accumulator3). Notice that this term is non-zero if the theta method ($Ph^{n+\theta} = \theta * Ph^{n+1} + (1-\theta)Ph^{n}$) is applied, otherwise, it is embedded in the matrix A;
- Boundary conditions or special terms that need to be added for specific kind of networks (accumulator4).

Depending on the provided system of pipes, some of these factors may be null. The complete algorithm is presented in the Appendix (A.1).

Definition of the matrix A

The matrix A (of size $N_{Ep} \times N_{Ep}$) is built in a sparse format. However, its structure is mostly tridiagonal, therefore, we avoid storing the whole matrix and instead only save two vectors:

1. CC - Central Contribution: Vector representing the elements on the diagonal of the matrix;
2. OC - Other Contribution: Matrix of size $N_{Ep} \times max(n_{\text{neighbors}})$ which contains the contribution from the neighbors of the $i-th$ element.

In the simplest case, namely the single-pipe, the matrix A (which is tridiagonal) is the following:

$$A = \begin{bmatrix} CC(1) & OC(1,2) & 0 & \cdots & 0 \\ OC(2,1) & CC(2) & OC(2,2) & \cdots & 0 \\ 0 & OC(3,1) & CC(3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & CC(N_{Ep}) \end{bmatrix} \tag{3.19}$$

Alternatively, in the GPU version, the Compressed Sparse Row (CSR) format for sparse matrices is used. In this form there are three vectors:

1. A_{row}: This vector contains the row index of the correspondent value in A_{val}
2. A_{col}: This vector contains the column index of the correspondent value in A_{val}
3. A_{val}: This vector contains the entries of the matrices

In particular, $A_{val}[i]$ is equal to $A[A_{row}[i], A_{col}[i]]$.

Conjugate Gradient Method

Since the vectors CC and OC are now defined, we can use those two instead of the sparse matrix A to simulate the matrix-vector and vector-matrix products involved in the CGOP. The only element which is still missing is the initial guess. As the initial value for the x, we set the value of the hydraulic head (at each point) at the previous iteration. Since no large variation between two successive iteration should occur, this guess should already be pretty close to the real solution, and therefore we expect the CGM to converge in a relatively small number of iterations.

Inner and Outer loop

Since we are trying to solve the nonlinear system using the Nested Newton method, we will have an inner and outer loop. The purpose of the outer loop is to iteratively refine the solution of the overall nonlinear system. On the other hand, the purpose of the inner loop is to build the matrix (in this case the vectors CC and OC) and solve the linear system, as well as solving the concave/convex part of the nonlinear function M. Notice that if one wants to apply the preconditioner it has to be applied on the linear system and it will speed up the convergence. As soon as a solution for the linear system is found, the outer loop takes care of computing the solutions for the nonlinear system, namely finding the values for the parameters that will be saved in the dataset as output of the simulation, for each time step.

3.5 Parallelization techniques

The numerical method presented is already efficient, but since a lot of operations are performed on vectors of size N_{Ep}, we can exploit multithreading to make each thread compute one value of the array. In such a way, computational time is saved, since we do not only rely on a single core. In our case, we use the Graphical Processing Unit (GPU) and CUDA, a proprietary parallel computing platform created by NVIDIA. All the experiments have been run on an NVIDIA GTX 1060 with a VRAM of 6GB.

3.5.1 Usage of the GPU

Whenever a variable is allocated, we have to decide whether to save it on the host (CPU memory) or the device (GPU memory). Our goal here is to find the best tradeoff between the transfer of data and how much this data is used. In particular, we want to save on the GPU-only data that is used several times throughout the computation. Additionally, since implementing a CUDA procedure running in multithread requires more time than implementing it serially, we want to implement only the functions in which the most CPU time is spent. For this purpose, we run the simulator and we analyse through the profiler which function required more CPU time. Results show that 94.44% of the CPU time is used in the Conjugate Gradient Method function. In particular, 75.00% of the time was spent on matrix-vector products, therefore we first focus on implementing the CGM on CUDA.

3.5.2 Multithreading management

Each GPU has a number of so known shading units, also known as CUDA cores. In the context of GPUs, we can refer as thread to a shader program executing. In particular, the GPU we use features 1280 shading units. Each unit will compute a value of the array, therefore, all the variables used in the CGM procedure (1) as temporary variables, as well as the final result x, will have to be saved on the device and since we only implemented the CGM in CUDA, we will have to transfer the data of the linear system at each iteration from the CPU to the GPU [15]. Potentially, one could implement the entire solver in CUDA and avoid this cost, which will further enhance the performances of the program.

3.6 Preconditioner

As mentioned previously, the system matrix A is SPD. However, the matrix is mostly tridiagonal with a few off-diagonal elements. The simpler the case, the more similar the matrix A is to a tridiagonal matrix. Therefore, we can consider the matrix A as $A = T + E$, in which, T is the tridiagonal part and E contains off-diagonal elements. The idea is to use the Thomas algorithm, specifically designed for tridiagonal matrices, to solve the system $Tx = b$. Our preconditioner will be $M = T$, and it will be used to transform the system $Ax = b$ to the system $M^{-1}Ax = M^{-1}b$. The preconditioned matrix is

$M^{-1}A$ and the preconditioned right-hand side is $M^{-1}b$. In such a way, the *CGOP* will convergence in a lower number of iterations, since $M^{-1}A$ is closer to being an identity matrix compared to A, which may be poor conditioned. On the other hand, $M^{-1}A$ will have a condition number close to the ideal one, namely 1. Hence, the directions computed at each iteration are more effective, and the residual error decreases faster. The CGOP algorithm with the preconditioner M becomes the following:

Algorithm 3 Conjugate Gradient Method - With preconditioner

procedure CGOP_PRECONDITIONED(A, b, x_1, tol, max_iter)
  T, E = Separate(A)
  $r_1 \leftarrow b - Ax_1$
  Solve $Tz_1 = r_1$ using Thomas
  $d_1 \leftarrow z_1$
  $i \leftarrow 1$
  while $i \leq$ max_iter and $||r_i|| >$ tol **do**
    $\alpha_i \leftarrow \frac{r_i^T z_i}{d_i^T A d_i}$
    $x_{i+1} \leftarrow x_i + \alpha_i d_i$
    $r_{i+1} \leftarrow r_i - \alpha_i A p_i$
    if $||r_{i+1}|| \leq$ tol **then**
      break
    Solve $Tz_{i+1} = r_{i+1}$ using Thomas
    $\beta_i \leftarrow \frac{r_{i+1}^T z_{i+1}}{r_i^T z_i}$
    $d_{i+1} \leftarrow z_{i+1} + \beta_i d_i$
    $i \leftarrow i + 1$
  return x_{i+1}

By using the Thomas algorithm as a preconditioner, we can significantly speed up the convergence of the CG. In particular, the more the matrix is similar to a tridiagonal one, i.e. the sparser the matrix E is, the faster the iterative solver will converge.

3.7 Flow Chart

Before moving on to the evaluation part, a flow chart is shown to clarify the order of application of all the components and avoid any doubts.

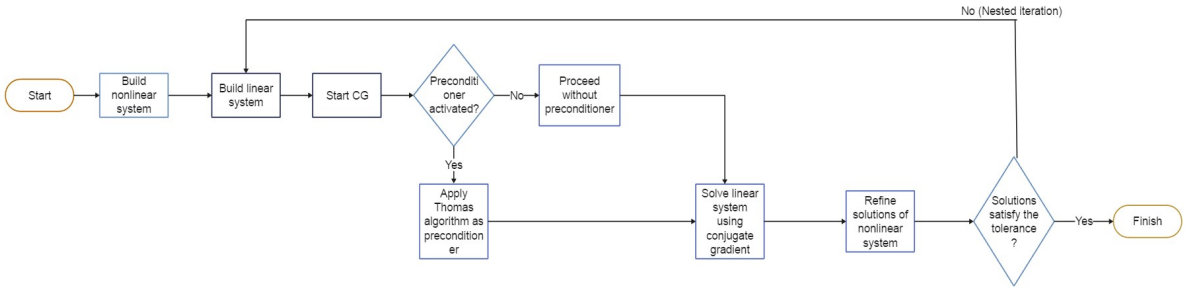

Figure 3.2: Flow Chart

Chapter 4

Evaluation

4.1 Correctness

In this section, we will address the correctness of the simulator, namely whether the data produced are coherent with the literature.

4.1.1 Single-Pipe

For the first simulation, we consider the simplest scenario possible, in which the system of pipes consists of a single pipe through which water flows. The parameters are the following:

Network Length	**Pipe Elasticity**	**Pipe Radius**	**Initial** (ρ, η, u)	**Station 1 Location**
1 km	1.7×10^{11} Pa	0.15 m	$(999.7, 50.0, 0.0)$	0.95 km

Table 4.1: Simulation Parameters - Single Pipe

Case 1 - No fracture and No Water Request

Firstly, we analyse the case in which there is no fracture along the pipe, as well as no demand. In this case, we expect no variation at all, since we are considering a stationary case.

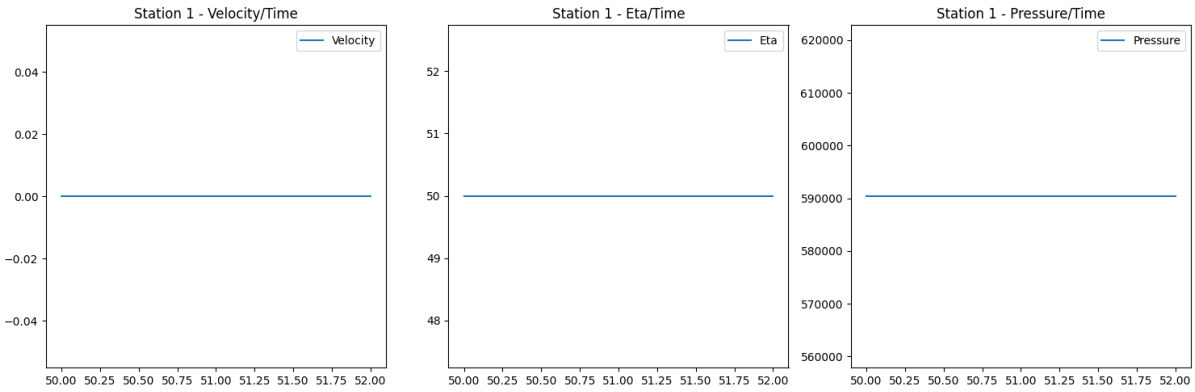

Figure 4.1: Time-Evolution of measurements on station1

The graphs display all constant functions. Since no water is requested from the pipe and no fractures are present, the velocity, density, and pressure of the fluid remain constant.

Case 2 - No fracture and Water Request

In this example there are still no fractures, but we want to analyse what happens if we add a demand. The water is extracted from a tank which is positioned at the end of the pipe. The demand is equal to 0.05 $\frac{m^3}{s}$ and water starts to get extracted at time 50.0s. At $t = 50.5s$, the demand is then reduced. First, we analyse the pressure and velocity over the pipe at $t = 50.0s$:

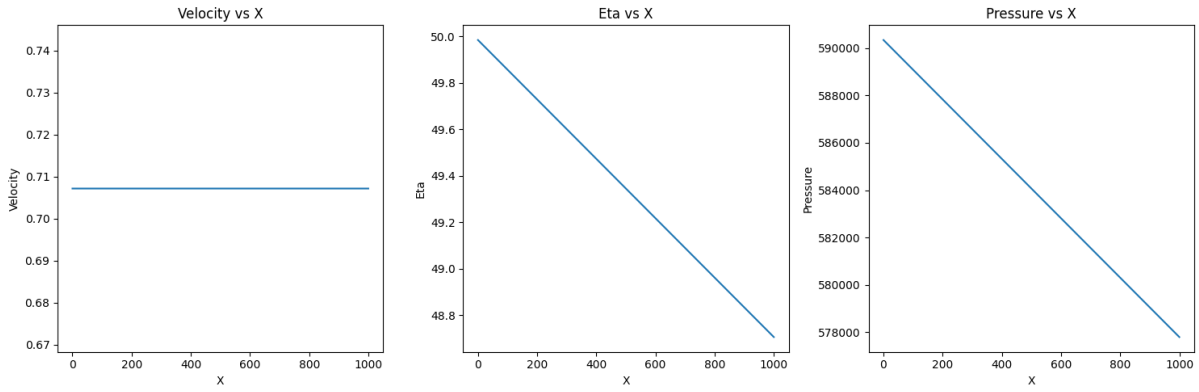

Figure 4.2: Space-Evolution of measurements over the entire pipe at time t=50s

The velocity is constant, since we have reached the steady state. As for the η and pressure, there is a linear drop due to friction. If friction was not considered in this simulation, there would have been no drop. Now that we have analysed the parameters in the space domain, let us do the same in the time domain:

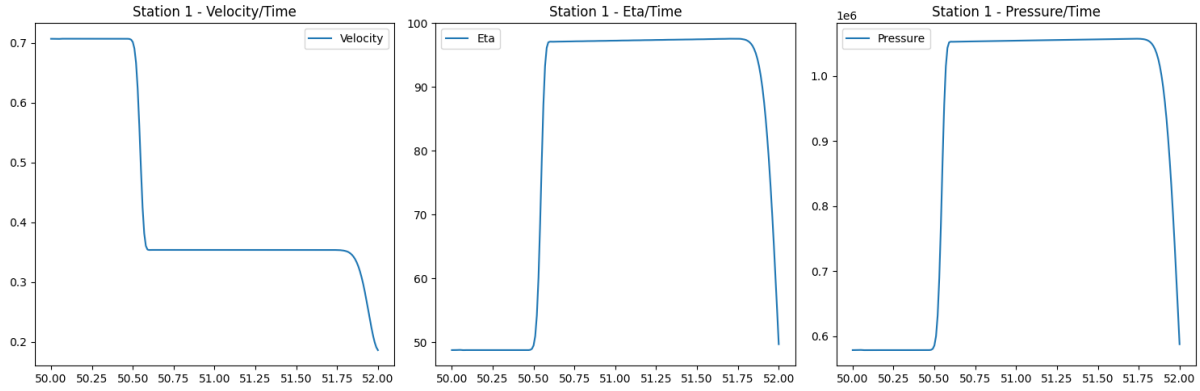

Figure 4.3: Time-Evolution of measurements on station1 with water request

The graphs are not constant anymore, but one can clearly see that the shape of the functions is smooth and no anomalies are present, and the reason behind this is that there is no fracture. Notice that the demand is reduced in a short amount of time following the rule defined in (3.1.2). The fast reduction of demand leads to the so-called *Water Hammer*, i.e. a fast pressure wave that travels backward on the pipe. As soon as the water starts to be extracted from the pipe, the pressure, as well as the hydraulic head, increases, just to go back to normal when the water has been extracted. On the other hand, the velocity of the fluid decreases, due to mass conservation.

Case 3 - Fracture and Water Request

Now we want to analyse the critical case, in which a fracture is present along the pipe. We expect to see some anomalies, that could be clearly detected also by a human since the network structure is very simple. The closer the station is to the fracture, the faster the anomaly should be detected and

reflected on the graphs. Hence, in this case we run the simulation twice, first placing a fracture at $x = 0.3$km and then at $x = 0.7$km.

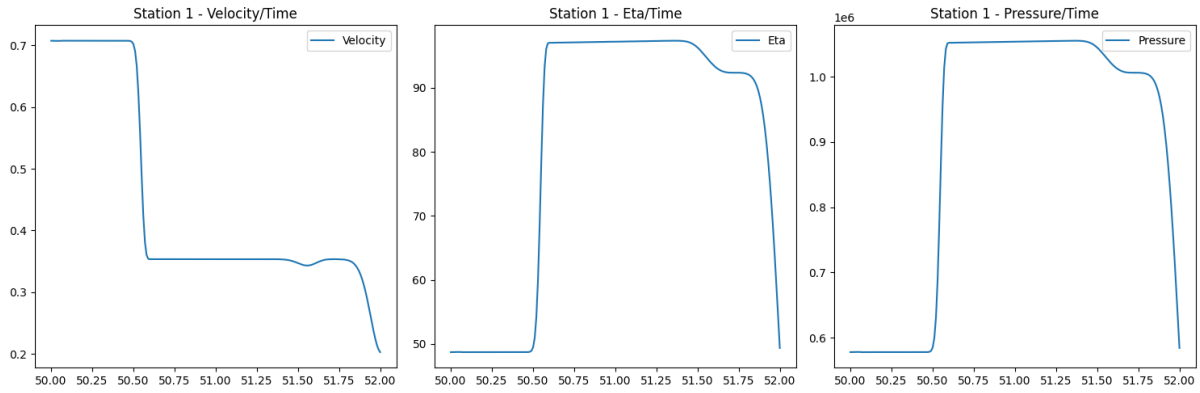

Figure 4.4: Station1 with water request and a fracture located at $x = 0.3$km

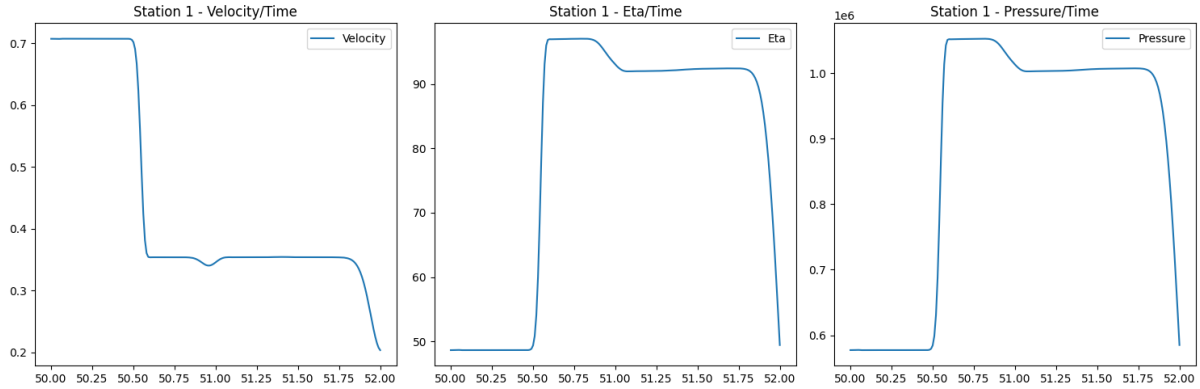

Figure 4.5: Station1 with water request and a fracture located at $x = 0.7$km

In both cases, one can see that there is a pressure loss due to the presence of the leak that is recorded by the station. In the second case, the loss was detected sooner, approximately at 51.00s compared to the 51.50s of the first simulation. This is correct since the station is closer to the second fracture than it is to the first one. Recall that when the wave hits the fracture, part of it is reflected, hence detected by the sensor. The greater the distance the wave has to travel, the later the leak will be detected.

4.1.2 Simple Network

In the previous case, we were able to both detect and locate the fractures just by looking at the graph and the results were consistent with the theory, however, we now want to analyse a more complex scenario. These are the parameters of the network:

Total Pipes	**Pipe Length**	**Pipe Elasticity**	**Pipe Radius**	**Initial** (ρ, η, u)	**Station 1 Location**
3	0.5 km	1.7×10^{11} Pa	0.15 m	$(999.7, 10.0, 0.0)$	0.95 km

Table 4.2: System Parameters - Simple Network

The network can be visualised as follows:

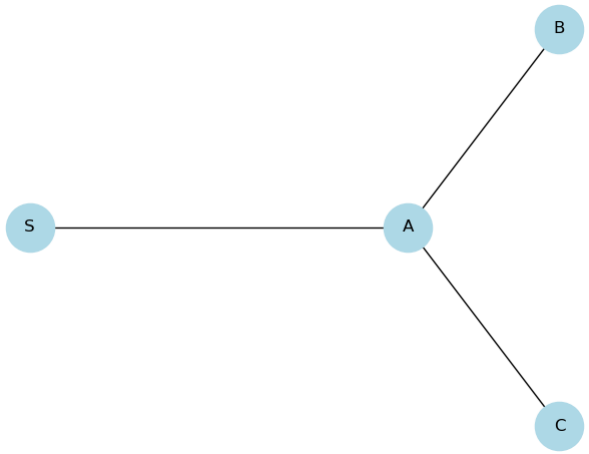

Figure 4.6: Simple Network

We have a total of 4 nodes and 3 edges. In a real application, we would encounter some problems in the interpretation of the graphs. For instance, if we place the sensor on the segment $\overline{SA}$ and the fracture is on the segment $\overline{AB}$ or $\overline{AC}$, we will be able to measure the distance from the sensor to the fracture, but not the correct pipe. To solve the issue, additional sensors should be placed. Instead, when using the numerical method, we distinguish the segment $\overline{AB}$ and $\overline{AC}$ by assigning them different y coordinates (or other dimensions if appropriate). In this case, the tank is placed where the junction is, namely at node A.

Case 1 - No fracture and no Water Request

First, we analyse the case with no fracture and no request from the tank.

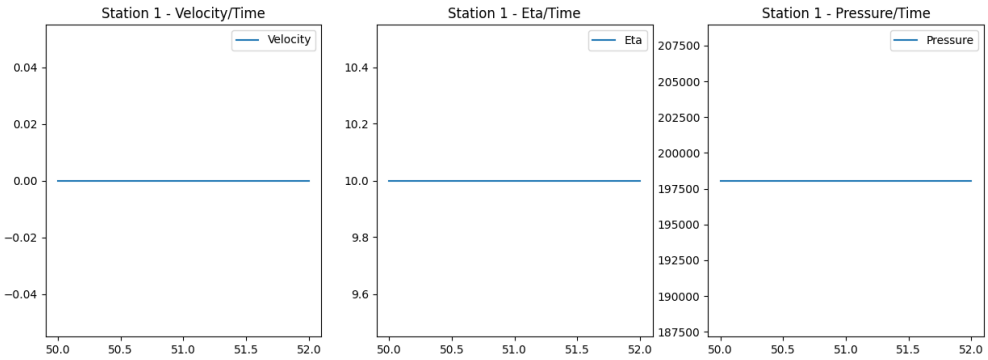

Figure 4.7: Station1 with water request

As one could expect, constant values are observed, just like in the case of a single pipe.

Case 2 - No fracture and Water Request

Let us now request some water from the tank. In particular, the demand is $0.2\frac{m^3}{s}$, starting from $t = 50.0s$, and is then reduced by a factor 2.

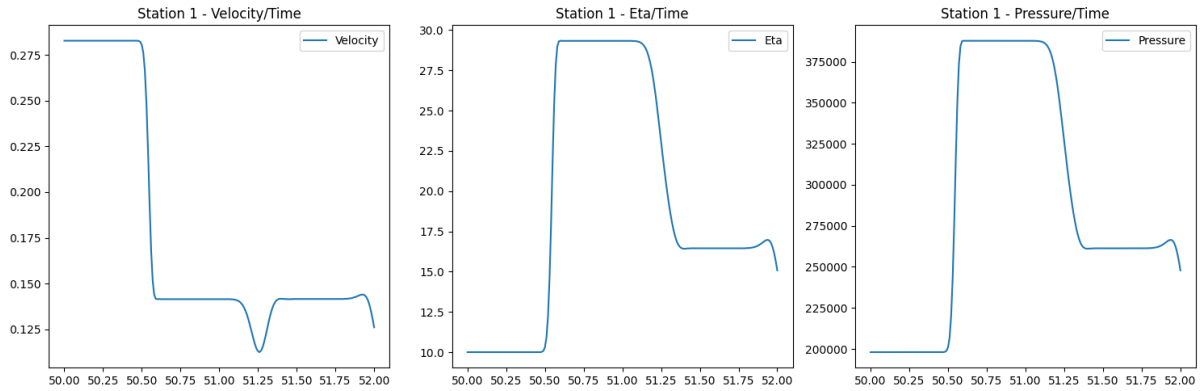

Figure 4.8: Station1 with water request

One can already see that the shape of the curve is more complex than the one for the single pipe.

Case 3 - Fracture and Water Request

A fracture on the $\overline{SA}$ segment is now placed.

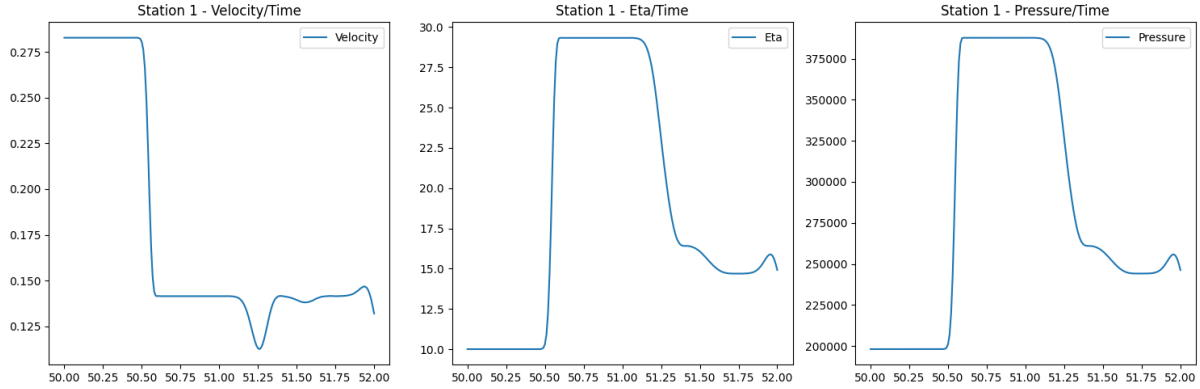

Figure 4.9: Station1 with water request and a fracture located at $x = 0.3km$

By comparing the graphs with and without the fracture, the difference is still noticeable. However, the point is that the more complex the network is, the more complex it will be to interpret these graphs. Therefore, a possible solution is to rely on Machine Learning approaches.

4.1.3 Riemann Problem 1 - Dam Break

The network consists of a single pipe of length 1 mt. We can modify the mass function in order to reproduce a classical open channel case by setting $\rho = \rho_0$ and $V = max(0, Ph)$. The parameters are:

Parameter	Value
Network Length	1 m
Pipe Elasticity	1.7×10^{11} Pa
Pipe Radius	0.15 m
Initial Condition (ρ, η, u) **for** $x < 0$	$(1.0, 1.0, 0.0)$ [Pa, m, $\frac{m}{s}$]
Initial Condition (ρ, η, u) **for** $x \geq 0$	$(1.0, 0.1, 0.0)$ [Pa, m, $\frac{m}{s}$]
Gravitational Constant g	$1\ m/s^2$

Table 4.3: System Parameters - RP1

No demand point is set. The exact same network was first analysed in [16]. We perform the same

analysis to verify the method implemented. The value set for Δx is 0.001m and the x-coordinates of the ends are -0.5m and 0.5m. Reference values were taken at time $t = 0.25s$.

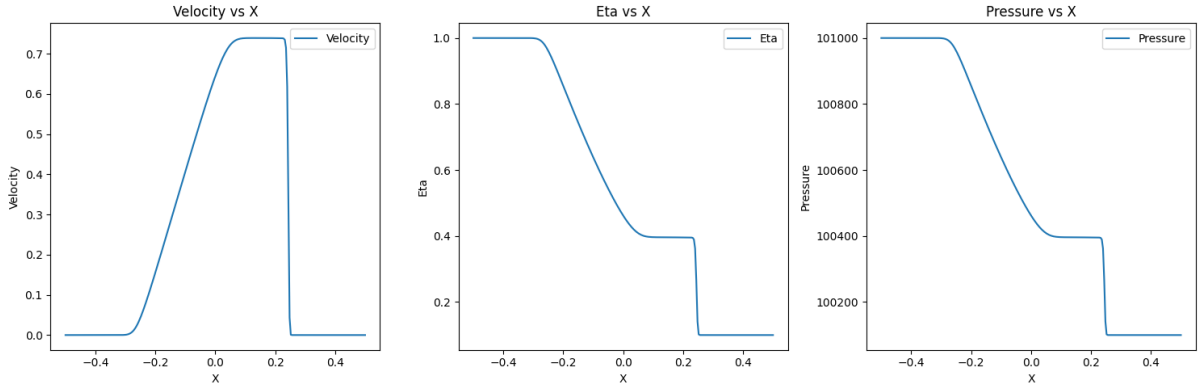

Figure 4.10: Space-Evolution of measurements of network RP1 at time $t = 0.25$s

The graphs are perfectly equivalent to the ones presented in the paper and therefore we can conclude that the simulation was successful. One question that may arise is why even though there is no fracture or demand the graphs are not constant. This is because the initial value for the hydraulic head was not constant for all segments, instead, the first half was set at 1m and the last half was set at 0.1m.

4.1.4 Riemann Problem 2

This network is very similar to the previous one, but the elastic behavior of the pipe and its properties change, see [13] as a reference (note that this problem is RP1 in the paper). The parameters are:

Parameter	Value
Network Length	1 m
Pipe Elasticity	1.0×10^{30} Pa
Pipe Radius	2.0×10^{-3} m
Initial Condition (ρ, η, u) **for** $x < 0$	$(10^6, 91.9, 0.0)$ [Pa, m, $\frac{m}{s}$]
Initial Condition (ρ, η, u) **for** $x \geq 0$	$(10^5, 0.0, 0.0)$ [Pa, m, $\frac{m}{s}$]
Gravitational Constant g	9.81 m/s^2

Table 4.4: System Parameters - RP2

As before, the parameters are analysed in the space domain, which in this case goes from -1m to 1m, and Δx is set to 0.001m.

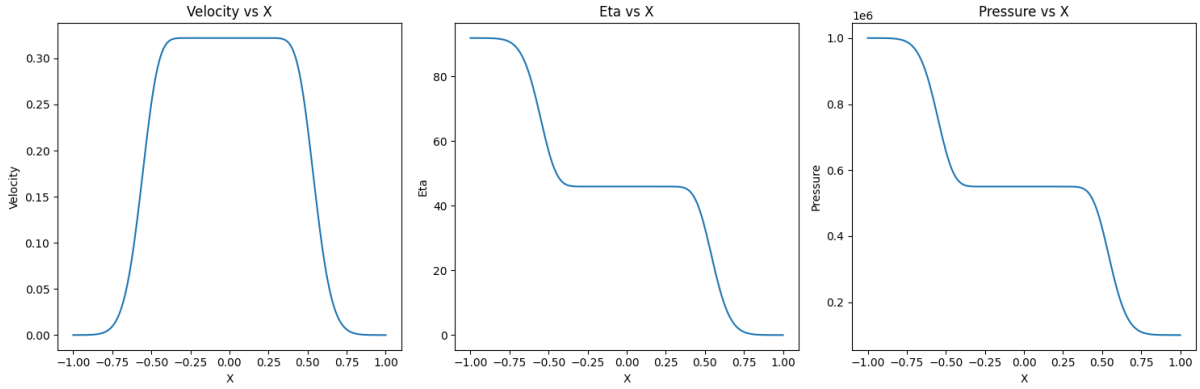

Figure 4.11: Space-Evolution of measurements of network RP2 at end time $t = 4 \times 10^{-4}$s

Again, the graphs obtained are equivalent to those in the paper mentioned before, and therefore we can also validate this test case.

4.1.5 Modena Network

Lastly, we run the simulator on a real-world network, situated in Modena. The network consists of 317 pipes, 272 nodes, and 283 tanks, for a total length of 71.651km. The network looks as follows:

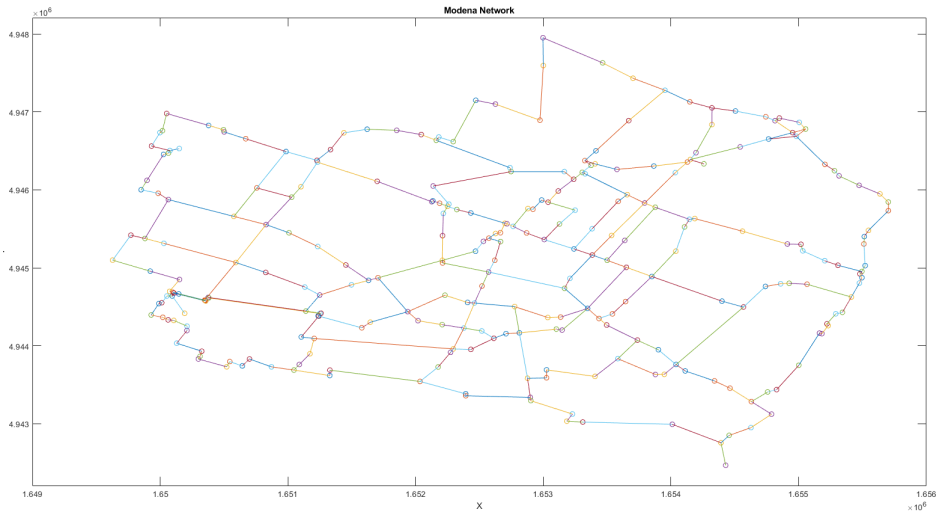

Figure 4.12: Modena Network (data taken from [17])

Almost all nodes have a demand, which varies from 0.01 $\frac{m^3}{s}$ to a maximum of 8.28 $\frac{m^3}{s}$. The initial η is 74.5m and we consider $\Delta x = 0.05$m. In this case, we do not set a lower resolution, as already with this one a system with 1.5 million unknowns must be solved. The end time of the simulation is $t = 502000s$ and $\Delta t = 1800$s. For the simulation, we do not place any fracture in the system and we make a space-domain analysis.

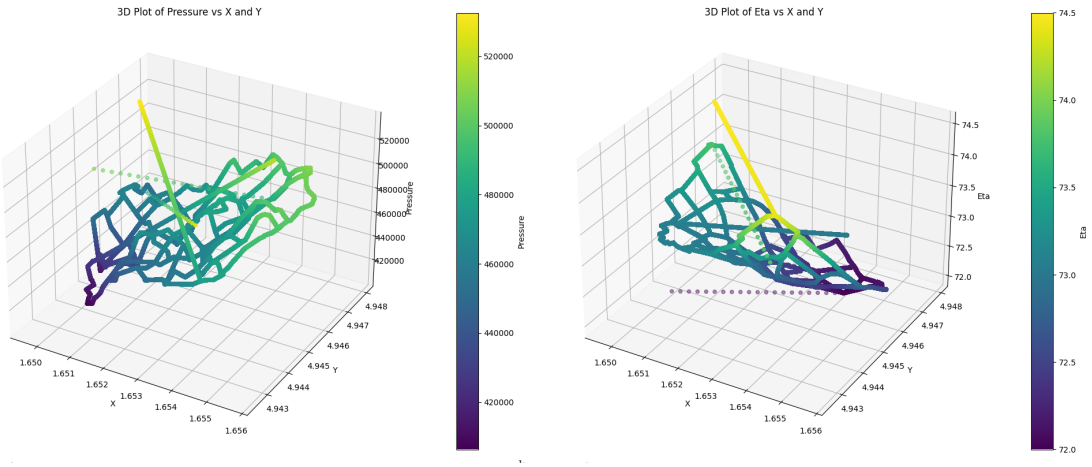

Figure 4.13: 3D Plot Modena - No Fracture at time t=502000s

The maximum value for both η and pressure is registered in correspondence with the source node of the network. The different elevation of each node and its demand are the reason why different pressure and η values are measured in the network.

4.2 Performances

In this section, we now assess the performances of the numerical method.

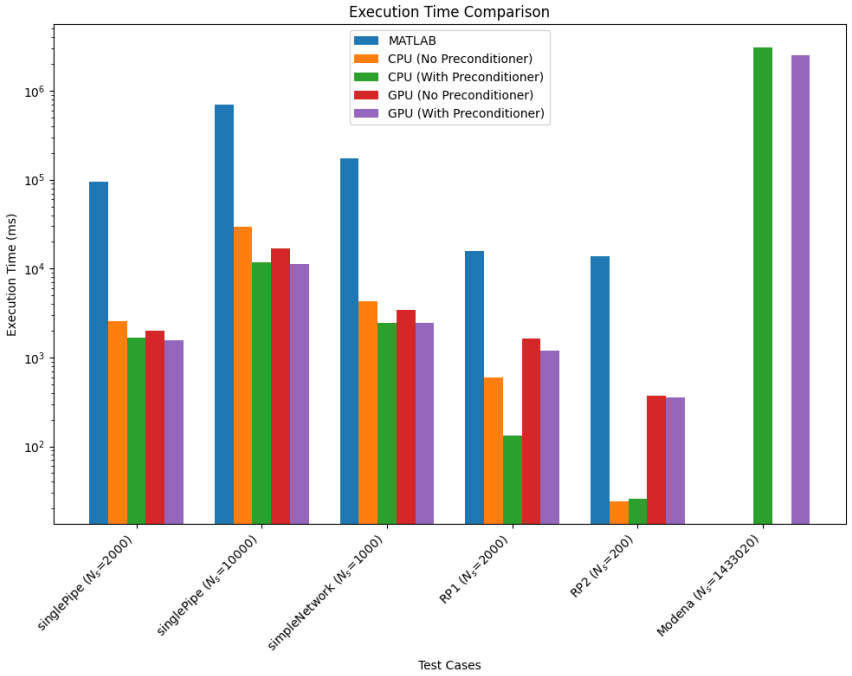

Figure 4.14: Performance analysis (in logarithmic scale)

Test Case	MATLAB	CPU (No Preconditioner)	CPU (With Preconditioner)	GPU (No Preconditioner)	GPU (With Preconditioner)
singlePipe (N_s=**2000**)	95920.92 ms	2535.16 ms	1655.28 ms	1995.79 ms	1578.40 ms
singlePipe (N_s=**10000**)	699553.35 ms	29491.42 ms	11708.00 ms	16731.81 ms	11204.94 ms
simpleNetwork (N_s=**1000**)	173002.02 ms	4338.00 ms	2431.12 ms	3394.10 ms	2440.00 ms
RP1 (N_s=**2000**)	15804.62 ms	602.23 ms	132.18 ms	1624.52 ms	1188.60 ms
RP2 (N_s=**200**)	13802.90 ms	24.23 ms	25.72 ms	367.83 ms	352.94 ms
Modena (N_s=**1433020**)	N/A	N/A	3104890.63 ms	N/A	2536750.36 ms

Table 4.5: Execution Times (in milliseconds) for Various Test Cases and Methods

From the table, one can clearly see that the improvements just by switching from MATLAB to Fortran are already great. Moreover, the preconditioner reduces the total time by speeding up the convergence of the CG as expected. In particular, the number of iterations for the simple cases tested goes from 300/400 to 1/2. That is because the matrix is basically tridiagonal. Therefore, the simpler the network, the more effective the preconditioner will be. This property is demonstrated by the fact that the CPU version using the preconditioner solves the problem faster than the GPU version without the preconditioner. However, the difference between the CPU version and the GPU version with the preconditioner activated is minimal. This behavior can be explained by looking at the networks. The networks considered for the test cases are simple, and therefore the systems involved will contain at most a few nonzero entries outside of the tridiagonal band. Consequently, by just applying the preconditioner we already have a good approximation of the solution of the system and only a few additional iterations will be needed. Since the preconditioner uses the Thomas algorithm, which is not parallelized since it requires ordered operations, most of the time is spent on the CPU. Ergo, the advantage of using parallel techniques is minimal, and in some cases, it could also affect negatively the total time. As regards the Modena network, which is a real scenario, the versions without the preconditioner were not able to solve the system in a maximum of 10000 iterations, and therefore no time was recorded.

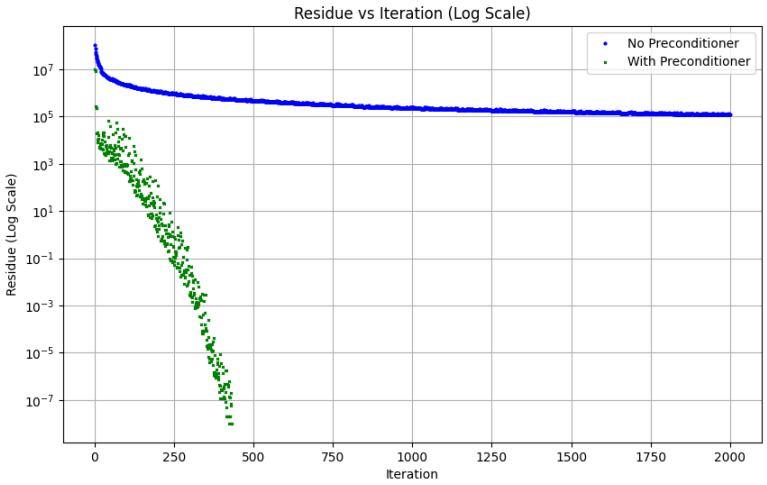

Figure 4.15: Convergence Plot for Modena Case - t=0 (Log Scale)

From the picture above, one can see that without the preconditioner, we will not be able to solve the linear problem. The versions using the preconditioner were able to solve the system with an average of 200 iterations each but required almost an hour to finish. In particular, the former converges but the residue curve is not monotonically decreasing. That is, because preconditioning generally accelerates convergences by improving the conditioning of the system, but does not guarantee the residue to decrease monotonically. In this case, we accept this behavior since it convergences in a reasonable amount of iterations and the norm of the error is small. In particular, it is of the order 10^{-4}. The GPU version is confirmed to be the fastest one as expected, even though not all operations were parallelized. Therefore, the simulator is open to further improvements.

Chapter 5

Conclusions and Future work

In this work, a model was first defined that could simulate the behavior of compressible fluids moving within a pipe system. Afterward, it was shown how equations could be used to implement a numerical method to simulate a real-world scenario. Finally, the most computationally critical portions of the code were analysed and solutions were sought to improve the efficiency of the code.

The numerical method implemented is consistent with the literature and confirms results already presented by other works. In addition, the performance analysis demonstrates the large gain from the use of a preconditioner and GPU. In a future study, this aspect could be further investigated and the whole method could be implemented in CUDA to enhance the performance. Concerning the final objective of the project, i.e. the implementation of a real-time system, the efficiency of the implemented method will allow the training of the networks to be carried out more quickly, considerably reducing the generation time of the dataset, especially for the real-world cases, in which the execution time is very large.

Furthermore, the mathematical model could be adapted to more complex scenarios that include more factors than those currently predicted. The literature on this subject is extensive and much work has already been done to demonstrate the applicability of many factors, so it would only need to be included in the mathematical model.

Acknowledgements

First and foremost, I would to thank my supervisor, Prof. Maurizio Tavelli, who constantly supported me throughout this research and provided me with all the necessary material to deeply understand the problem and possible solutions. Thanks to his assistance, I was able to understand complex physical properties that govern real-world phenomena that I had never encountered before in my academic career. I would also like to thank my co-supervisor, Prof. Bruno Carpentieri, who first introduced me to the project and gave me some interesting cues to further expand my knowledge in linear algebra and apply it on this work.

My thanks also go to all the people who have helped me during these three years. Not only my family, whom I thank for always allowing me to follow my dreams but also my friends, who have put up with me throughout this period. Thank you very much.

Lastly, I would like to thank everyone who has helped me over the years and whom I have not mentioned before, because the list would become too long, but I am truly grateful to all of you.

Bibliography

[1] P. D. Smith. *Basic Hydraulics*. Google Books, 2013.

[2] L. Hamil. *Understanding Hydraulics*. Springer, 2001.

[3] Eleuterio F. Toro. *Riemann solvers and numerical methods for fluid dynamics a practical introduction*. Springer, 2010.

[4] Vincenzo Casulli and Guus S. Stelling. A semi-implicit numerical model for urban drainage systems. *International Journal for Numerical Methods in Fluids*, 73(6):600–614, 2013.

[5] V. Casulli. *Semi-implicit finite difference methods for the two–dimensional shallow water equations*. Journal of Computational Physics, 1990.

[6] G. Strang. *Linear Algebra and its applications*. Cengage Learning, 2006.

[7] V. Casulli, P. Zanolli. *A Nested Newton-Type Algorithm for Finite Volume Methods Solving Richards' Equation in Mixed Form*. SIAM Journal on Scientific Computing, 2010.

[8] J. Nocedal, S. J. Wright. *Conjugate Gradient Methods*. Springer, 2006.

[9] Saba Akram and Quarrat Ul Ann. Newton raphson method. *International Journal of Scientific & Engineering Research*, 6(7):1748–1752, 2015.

[10] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug. Inexact newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[11] W. T. Lee. *Tridiagonal Matrices: Thomas Algorithm*. MS6021, Scientific Computation, University of Limerick, 2011.

[12] Maurizio Tavelli and Michael Dumbser. A pressure-based semi-implicit space–time discontinuous galerkin method on staggered unstructured meshes for the solution of the compressible navier–stokes equations at all mach numbers. *Journal of Computational Physics*, 341:341–376, 2017.

[13] Michael Dumbser, Uwe Iben, and Matteo Ioriatti. An efficient semi-implicit finite volume method for axially symmetric compressible flows in compliant tubes. *Applied Numerical Mathematics*, 89:24–44, 2015.

[14] Vincenzo Casulli and Paola Zanolli. Iterative solutions of mildly nonlinear systems. *Journal of Computational and Applied Mathematics*, 236(16):3937–3947, 2012.

[15] V. Kindratenko. *Numerical Computations with GPUs*. Springer, 2014.

[16] Maurizio Tavelli, Walter Boscheri, Giulia Stradiotti, Giuseppe Roberto Pisaturo, and Maurizio Righetti. A mass-conservative semi-implicit volume of fluid method for the navier–stokes equations with high order semi-lagrangian advection scheme. *Computers Fluids*, 240:105443, 2022.

[17] C. Bragalli, C. D'Ambrosio, J. Lee, A. Lodi, P. Tot. *On the Optimal Design of Water Distribution Networks: A Practical MINLP Approach*. Springer, 2008.

Appendix A

Appendix

A.1 Computation of RHS

To compute the Right-Hand Side of the system

$$Ax = b, \tag{A.1}$$

we need to take into consideration all the factors presented.

Algorithm 4 Computation of Right Hand Side Vector b

function COMPUTEB($M, Ph, rhou, Frhou, A, R, dt, IsActive, Network, Numsol$)
  $b \leftarrow$ zeros array of size $Network.Nep$
  for $i \leftarrow 2$ to $Network.Nep$ **do**
    Initialize accumulator1, accumulator2, accumulator3, accumulator4 $\leftarrow 0$
    for j in GETNEIGHBORS($i, Network$) **do**
      weight $\leftarrow$ COMPUTEWEIGHT($i, j, Network$)
      inv_momentum $\leftarrow$ COMPUTEINVMOMENTUM(.., $j, Network$)
      accumulator1 $\leftarrow$ accumulator1 + weight $\cdot\, A(j) \cdot$ rhou(j)
      accumulator2 $\leftarrow$ accumulator2 + weight $\cdot\, A(j) \cdot$ Frhou(j) $\cdot$ inv_momentum
      accumulator3 $\leftarrow$ accumulator3 + $\frac{\text{g}\cdot\text{weight}\cdot\text{A}(j)\cdot(\text{Ph(Network.right}(j))-\text{Ph(Network.left}(j)))}{\text{Network.dxj}(j)}$ $\cdot$ inv_momentum
      active_index $\leftarrow$ GETACTIVEINDEX($i, j, Network$)
      if IsActive(active_index) == 1 **then**
        inv_momentum $\leftarrow$ COMPUTEINVMOMENTUM(.., $j, Network$)
    $b(i) \leftarrow M(i) - dt \cdot (1-\theta) \cdot$ accumulator1
    $b(i) \leftarrow b(i) - dt \cdot \theta \cdot$ accumulator2
    $b(i) \leftarrow b(i) + dt^2 \cdot \theta \cdot (1-\theta) \cdot$ accumulator3
    $b(i) \leftarrow b(i) +$ accumulator4
    $b(i) \leftarrow b(i) - dt \cdot$ Network.Demand(i)
    fracture_term $\leftarrow$ Network.C(i) $\cdot$ p(i)$^{\text{Network.alpha}(i)}$ + Network.alpha(i) $\cdot$ Network.C(i) $\cdot$ p(i)$^{\text{Network.alpha}(i)-1} \cdot \text{g} \cdot \rho_0 \cdot$ Ph(i)
    $b(i) \leftarrow b(i) - dt \cdot$ fracture_term
  return b