Imperial College London

Department of Computing

Master Thesis

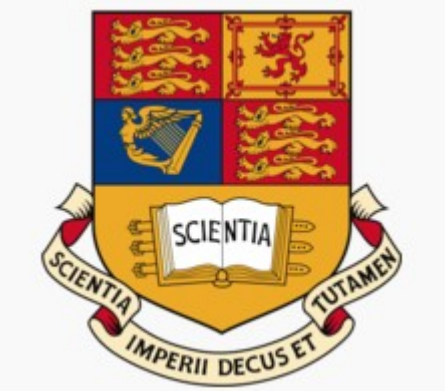
Smooth Optimisation and Computationally Robust Numerical Solutions in Hydraulic Modelling

Author:
Bogdan Cozmaciuc

Supervisor:
Dr. Panos Parpas

Second Supervisor:
Dr. Marc Deisenroth

June, 2014



Abstract

Water distribution networks represent a vital part of our infrastructure, providing the basis of good wealth and well-being. In recent years, we have witnessed large technological enhancements in the water sector and, as a result, there is now a great research interest in achieving more advanced solutions that will address issues such as real-time failure diagnosis and control of large scale water distribution networks.

At a lower level, we are interested in providing robust numerical solutions to the hydraulic equations, task that represents the control of water distribution networks. By exploiting the sparse nature of the problem and conducting a performance analysis of linear solvers, we will show that we obtain better computational results compared to the state-of-the-art hydraulic solvers.

Also, we are interested in providing a rigorous mathematical optimisation framework that will help us use optimisation algorithms such as Newton-Raphson in a *correct* manner and formally prove that we can achieve better computational complexity than the standard methods in use. The key idea is to transform the current optimisation problem into a smooth optimisation one by finding a μ-smooth approximation of the original formulation.

Finally, we will deliver a reliable framework that researchers can use in order to develop and validate new theories and ideas in the field of hydraulics.

Acknowledgements

I would like to thank:

- My first supervisor, Dr. Panos Parpas, for his constant guidance and support throughout the project. His advice has been of great value in the development of the project and his enthusiasm was an important motivation for me.
- My second supervisor, Dr. Marc Deisenroth, for his useful suggestions on the project. His prompt and detailed feedback has been of great help with writing this report.
- Dr. Ivan Stoianov, for kindly offering me the chance of working alongside his research group during the development of my thesis. His continuous support and research ideas have made a substantial impact in the outcome of this project.
- Dr. Edo Abraham, for the discussions and advice received on several aspects of my project including linear solvers, implementation details and hydraulic modelling.
- Robert Wright, for his insight in hydraulic modelling.

Last, but not least, I would like to thank my family, especially my parents, Dragos and Constantina, and my sister, Ana, for their unconditional support, trust and encouragement that they have shown throughout my studies.

Contents

List of Figures

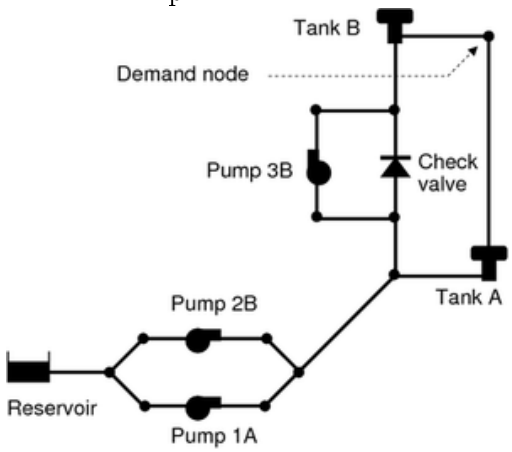
List of Tables

1 Introduction

1.1 Project Explanation

The project we are working on concerns near real-time failure diagnosis and control of large scale water distribution networks, tasks that are achieved through computationally efficient algorithms and optimisation models. A water distribution network can be perceived as a graph whose topological structure consists of links, which are pipes, pumps and valves, and nodes, which are tanks, reservoirs and other junctions of the aforementioned links. An example is given in Figure 1.1.

Figure 1.1: A simple Water Distribution Network



Thus, we are interested in the modelling and high performance flow analysis of water networks or, more precisely, determining both the optimal water flow and pressure at the links and nodes of the network, respectively. In order to find these values, we require the use of optimisation algorithms and in the past 30 years, Newton-Raphson has been the main algorithm used [1].

We will explain why the current problem formulation cannot be *correctly* solved by Newton-Raphson and prove that a new optimisation framework will adhere to the conditions of optimisation algorithms and increase even further the overall performance of the hydraulic analysis. The idea is that the current problem formulation uses non-differentiable functions as part of the objective function and we will modify this, whilst keeping in mind the accuracy of the solution. Moreover, we will conduct a rigorous numerical analysis to establish a methodology that optimally solves our problem.

1.2 Motivation

At a high level view, water supply networks represent a critical part of our infrastructure, providing the basis of good health and well-being. In recent years there has been a large expansion in technology in the water sector and, as a result, there is now a strong interest in achieving a more advanced form of pipe analysis and failure detection and control that will integrate with the notion of 'Smart Cities'.

An integral part of achieving a smart water system is the modelling and control of such a system. Water utilities across the world are facing certain difficulties due to the ageing technology. Among other issues, water loss is a major reason of concern that can be addressed through the implementation of smart water systems, which can perform tasks as leak detection and pressure monitoring, and obtain a more efficient use of water supplies.

Moreover, it is a project which has a paramount social impact, especially in less developed environments where water represents a crucial problem. The development of such a smart water network will enhance the overall life quality and will make a momentous impact for a large number of people.

1.3 Project aims

The aims of the project are to provide performance enhancements on the hydraulic analysis problem, which can be achieved in two main areas.

Firstly, we investigate and establish a methodology which is optimally efficient for solving the flow analysis problem. This is done by analysing and determining the various linear solvers, matrix storage schemes and implementation mechanisms best suited for the problem.

Secondly, we propose a new optimisation framework that mathematically guarantees a better complexity on finding the optimal solution than the standard algorithm and also guarantees that Newton-Raphson can be used as an optimisation algorithm.

We will conclude by delivering a robust and flexible C++ framework that can easily be used by researchers in order to develop and validate new ideas and theories in the field.

1.4 Contributions

We propose a methodology to address the aims of the project and meet the objectives which have been established based on the following contributions:

1. Proposing the use of a new optimisation framework and guaranteeing an improved complexity of the number of iterations required for the convergence of the optimisation algorithm through the use of smoothing functions.
2. Offering a robust mathematical framework for hydraulic analysis which can safely be used in conjunction with the Newton-Raphson algorithm or other optimisation algorithms. This

is a main contribution as the problem formulation which has been used is not suitable for this use.

3. Allowing any possible extensions to the hydraulic model to be implemented. For instance, we may impose restrictions on the water flow or water pressure that would resemble better the physical properties of the water network.
4. A rigorous numerical analysis of different implementations which focus on the overall time required to obtain optimal solutions. The analysis puts emphasis on the matrix operations (multiplications, decompositions) that are performed and on the linear solver we choose.
5. A robust and reliable framework that suits perfectly the research study in the field. This includes developing and validating theories regarding the flow analysis problem.
6. A full C++ implementation of a hydraulic solver that solves for the optimal values of water flow and pressure in the network.

1.5 Report structure

The report will follow a simple structure and will detail on how we have achieved the main contributions, starting from the objectives we have set. We will list and briefly discuss the chapters that follow:

1. **Background**
   This chapter gives the required background on mathematical optimisation and hydraulic principles and models so that we can tackle the aims we have set.
2. **Numerical Analysis**
   In this chapter we talk about the process, which was undertaken, in order to achieve high computational performance and optimal accuracy. We see the momentous impact that matrix operations and linear solvers have on the overall cost of the simulation and elaborate on the methodology that brought us to outperform the state-of-the-art hydraulic solvers.
3. **Smooth Framework**
   We introduce a novel way to solve the system of hydraulic equations, which can be *correctly* solved by the Newton-Raphson algorithm. We also prove that in the optimisation framework we introduce, we achieve a better bound on the complexity of the number of iterations required for convergence than the standard algorithms currently used.
4. **Other Ideas**
   This chapter presents the main topics we have tackled: a robust formulation of our optimisation problem, as well as valve modelling and a complete implementation of a pressure-driven model.
5. **Implementation**
   In this chapter we cover the implementation details of the hydraulic solver and explain why it is a good framework for researchers to easily develop and validate their ideas and theories.
6. **Conclusion**
   Lastly, we summarise the main achievements of the project and discuss possible extensions

to the project, as well as offering recommendations for future work.

2 Background

2.1 Mathematical Optimisation

This section provides a brief overview of some optimisation techniques that will be considered when solving the problems for the water network.

2.1.1 Optimisation algorithms

A generic optimisation problem involves solving a problem of the form:

$$\min_x f(x) \tag{2.1}$$

$$s.t. : x \in \Omega, \tag{2.2}$$

where Ω represents the feasible set of the problem. To solve this problem for a convex function (i.e. a function with only one local minimum), there are a few different approaches that follow the same basic idea of generating a sequence of numbers x_1, x_2, .. such that $f(x_1) > f(x_2) > f(x_3) >$ .. so it intuitively yields:

$$\lim_{k\to\infty} x_k = x^*,$$

where x^* denotes the local minimum of the function.

We will now consider some types of optimisation problems that will arise including Linear Programming, Unconstrained Problems and Constrained Problems, where the last two will consist of the Nonlinear Programming paradigm. We will skip the former as it is not required by the project and we refer to [16] for details.

2.1.2 Unconstrained Problems

With unconstrained problems, we face a simpler version of the problem described by equations (2.1)–(2.2), where the objective function can be non-linear and we face no other optimality constraints on x. We are therefore interested in solving problems of the type:

$$\min_{x\in\mathbb{R}^n} f(x),$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$.

Before discussing the optimality conditions of these problems, we will take a look at constrained optimisation which is our next section.

2.1.3 Constrained Optimisation

The discussion so far has been restricted to unconstrained optimisation problems where we had to minimize a certain objective function without putting any constraints on the optimisation variables. This framework for solving optimisation problems is useful, but sometimes we need a more general one in order to solve certain problems.

There are two generic types of problems that fall into the category of constrained optimisation. First of all, we have equality constraints and the problems we encounter are of the type:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{2.3}$$

$$s.t. : h(x) = 0, \tag{2.4}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$ and $m \leq n$.

A second type of constrained optimisation problems that arises is when we have inequality constraints as well. Simply put, the problem is formulated as:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{2.5}$$

$$s.t. : h(x) = 0 \tag{2.6}$$

$$g(x) \leq 0, \tag{2.7}$$

where f and h are defined as above and $g : \mathbb{R}^n \to \mathbb{R}^p$ and the feasible region could be defined as $\Omega = \{x \in \mathbb{R}^n \mid h(x) = 0, g(x) \leq 0\}$.

2.1.4 Optimality Conditions For Equality Constraints

We will now establish necessary and sufficient conditions for a constrained optimisation problem that only has equality constraints. These conditions are required to be satisfied by a solution point and thus, are important in developing algorithms to determine the points satisfying them.

Before we resume the discussion, we shall introduce the term of regular points, which will help us define the necessary and sufficient conditions. Thus, a point x^* satisfying the constraints of the optimisation problem (i.e. $h(x^*) = 0$) is called regular if the gradient vectors $\{ \nabla h_1(x*)$, $\nabla h_2(x*)$, ..., $\nabla h_m(x*) \}$ are linearly independent.

Next, we will take a look at some first order and second order conditions, which need to be satisfied by any solution point:

- First Order Necessary Conditions (FONC)
- Second Order Necessary Conditions (SONC)
- Second Order Sufficient Conditions (SOSC)

where first order refers to the fact that only the gradient is required and second order includes the hessian as well.

The FONC or Lagrange's theorem for this problem states:
Let x^* be a local minimizer (maximizer) of $f : \mathbb{R}^n \to \mathbb{R}$ subject to $h(x) = 0$, $h : \mathbb{R}^n \to \mathbb{R}^m$, $m \leq n$. Assume that x^* is a regular point. Then there is a $\lambda^* \in \mathbb{R}^m$ such that:

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* = 0. \tag{2.8}$$

We will now introduce some useful terminology regarding the FONC, which we have written above:

$$h(x^*) = 0$$
$$\nabla f(x^*) + \nabla h(x^*)\lambda^* = 0,$$

represents the Lagrange condition. The vector λ^* is the Lagrange multiplier vector. The Lagrangian function associated with the problem is:

$$L(x, \lambda) = f(x) + \lambda^T h(x).$$

We can now observe that Lagrange Condition above is nothing else but $\nabla L(x, \lambda) = 0$.

The SONC for this problem is as follows:
Let $\Omega = \{x \mid h(x) = 0\}$ and $f \in \mathcal{C}^2$, $x^* \in \Omega$ be a local minimizer of $f : \mathbb{R}^n \to \mathbb{R}$ over Ω and where $h : \mathbb{R}^n \to \mathbb{R}^m$ with $m \leq n$ is also in $\mathcal{C}^2$. Suppose x^* is regular. Then there exists a λ^* such that:

$$1. \nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla_x f(x^*) + \nabla_x h(x^*)\lambda^* = 0 \tag{2.9}$$
$$2. d^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) d \geq 0, \forall d \text{ such that } \nabla h(x^*)^T = 0, \tag{2.10}$$

where $\nabla_x \mathcal{L}(x, \lambda)$ denotes the Lagrangian associated with the problem.

Up until we have seen only necessary conditions. However, a point may satisfy the necessary conditions without actually being a solution. We will now look at SOSC:
Let $\Omega = \{x \mid h(x) = 0\}$ and $f \in \mathcal{C}^2$ where $h : \mathbb{R}^n \to \mathbb{R}^m$ with $m \leq n$ is also in $\mathcal{C}^2$. Suppose there is some $x* \in \mathbb{R}^n$ and $\lambda^* \in \mathbb{R}^m$ such that:

$$1. \nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla_x f(x^*) + \nabla_x h(x^*)\lambda^* = 0 \tag{2.11}$$
$$2. d^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) d > 0, \forall d \text{ such that } \nabla h(x^*)^T d = 0. \tag{2.12}$$

Then x^* is a strict local minimizer for f over Ω.

2.1.5 Optimality Conditions for Inequality Constraints

As we did with the equality constraints, in order to find a minimizer for the problem, we will define some necessary and sufficient conditions for a point to be a solution of the problem formed by equations (2.8)–(2.10). Before we begin, we will introduce the notions of active constraint and regular points, which will be needed by the optimality conditions.

- Active Constraint
  An inequality constraint $g_j(x) \leq 0$ is said to be active at the point x^* if $g_j(x^*) = 0$. It is inactive if $g_j(x^*) < 0$.

- Regular Constraints
  Let x^* satisfy $h(x^*) = 0$ and $g(x^*) \leq 0$ and let $J(x^*)$ be the index set of active inequality constraints:
  $$J(x^*) = \{j \mid g_j(x^*) = 0\}.$$
  Then we say that x^* is a regular point if the vectors $\nabla h_i(x^*)$, $\nabla g_j(x^*)$ with $1 \leq i \leq m$ and $j \in J(x^*)$ are linearly independent.

We can formulate the Karush-Kuhn-Tucker (KKT) Theorem, which established the FONC for the optimisation problem with inequality constraints:
Suppose that f, g and h are $\mathcal{C}^1$. Let x^* be a regular and local minimizer of problem found at equations (2.8)–(2.10). Then there exists some $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p$ such that:

a. $\mu^* \geq 0$

b. $\nabla_x f(x^*) + \nabla_x h(x^*)\lambda^* + \nabla_x g(x^*)\mu^* = 0$

c. $(\mu^*)^T g(x^*) = 0$

d. $h(x^*) = 0$

e. $g(x^*) \leq 0$,

Maximization problems, problems with *greater than* constraints can be solved in a similar manner after some basic mathematical manipulation.

Next, we will present the SONC for this case:
Suppose that f, g and h are $\mathcal{C}^2$. Let x^* be a regular and local minimizer of problem found at equations (2.8)–(2.10) over $\{ x \in \mathbb{R} \mid h(x) = 0,\ g(x) \leq 0 \}$. Then there exists some $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p$ such that:

a. $\mu^* \geq 0$

b. $\nabla_x f(x^*) + \nabla_x h(x^*)\lambda^* + \nabla_x g(x^*)\mu^* = 0$

c. $(\mu^*)^T g(x^*) = 0$

d. $h(x^*) = 0$

e. $g(x^*) \leq 0$

f. $d^T H(x^*, \lambda^*, \mu^*)d \geq 0$, $\forall d$ such that:

$$\nabla_x h(x^*)d = 0$$

$$\nabla_x g_j(x^*)d = 0 \text{ with } j \in J(x^*)$$

and where $H(x, \lambda, \mu)$ is the Hessian of the Lagrangian defined as:

$$H(x, \lambda, \mu) = \nabla_x^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla_{xx}^2 h_i(x) + \sum_{i=1}^{p} \mu_i \nabla_{xx}^2 g_i(x).$$

Finally, we can discuss sufficient conditions of optimality and we will present the SOSC for this case:
Suppose that f, g and h are $\mathcal{C}^2$ and that there exists points $x^* \in \mathbb{R}^n$, $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p$ such that:

a. $\mu^* \geq 0$

b. $\nabla_x f(x^*) + \nabla_x h(x^*)\lambda^* + \nabla_x g(x^*)\mu^* = 0$

c. $(\mu^*)^T g(x^*) = 0$

d. $h(x^*) = 0$

e. $g(x^*) \leq 0$

f. $d^T H(x^*, \lambda^*, \mu^*)d > 0$, $\forall d$ such that:

$$\nabla_x h(x^*)d = 0$$

$$\nabla_x g_j(x^*)d = 0 \text{ with } j \in J(x^*, \mu^*),$$

where $J(x^*, \mu^*) = \{i \mid g_i(x^*) = 0, \mu_i^* > 0\}$.

Then x^* is a local minimizer of f over $\{x \in \mathbb{R}^n \mid h(x) = 0, g(x) \leq 0\}$.

2.1.6 Descent algorithms

These algorithms are called descent algorithms as they decrease the value of the objective function with each iteration. The generic scheme for a descent algorithm adheres to the following structure:

1. Given a point x_k.
2. Derive a descent direction $d_k \in \mathbb{R}^n$, i.e. $\nabla f(x_k)^T d_k < 0$
   where
   $$\frac{\partial f}{\partial d}(x) = \nabla f(x_k)^T d_k$$
   is the directional derivative of f at point x along direction d.
3. Decide on a step-size α_k.
4. Compute the next point $x_{k+1} = x_k + \alpha_k d_k$.

Since the directional derivative is negative, the function is decreasing along the direction of d. Since it is only locally decreasing, then the step-size is required, but must be chosen carefully since if it is too large we might not get to a decreasing point and if it is too small it will add complexity to the algorithm on converging.

There are a few ways in which the aforementioned step-size can be chosen:

1. Exact Line Search
   where α_k is chosen as: $\alpha_k \in arg\min_{\alpha_k} f(x_k + \alpha_k d_k)$
2. Inexact Line Search
   - Percentage Test
     Similar to exact line search, but stops within a fixed accuracy (e.g. $|\alpha_k - \alpha^*| < c\alpha^*$, with $0 < c < 1$, $c = 0.1$ a typical value).
   - Curve fitting
     Fit a function to $f(x_k + \alpha_k d_k)$ and minimize that (e.g. Newton's method with a quadratic fit)

- Armijo's Rule
  A rule to ensure that α is not too large and not too small.
- Backtracking
  A simple variation of Armijo's rule

2.1.7 Gradient Descent

A concrete example of descent algorithm is the gradient descent algorithm that always assumes the decent direction to be $-\nabla f(x_k)$, thus transforming the scheme outlined above into:

1. Given a point x_k.
2. Compute the gradient at x_k and let d_k be the descent direction:
$$d_k = \nabla f(x_k)^T.$$
3. Decide on a step-size α_k,
$$\alpha_k \in arg\min_{\alpha_k} f(x_k + \alpha_k d_k).$$
4. Transition to the next point $x_{k+1} = x_k + \alpha_k d_k$.

The Gradient Descent Algorithm is widely used as it offers a number of advantages such as ease of implementation, the fact that it only requires first order information (gradient) and, more importantly, it does converge to a minimum provided an appropriate step-size strategy.

2.1.8 Newton-Raphson

The Newton-Raphson method gives a more efficient way of minimizing an objective function by taking into account both the first and second order derivatives of the function we try to minimize. It does so at the expense of added complexity and facing issues such as being sensitive to initial conditions (it might not always converge), it might cycle or it might even fail to find a descent direction. There are certain methods that overcome these shortcomings as we will discuss further.

To begin with, minimizing a general non-linear function $f(x)$ is a difficult problem to solve and instead, the Newton method uses a quadratic approximation of the initial function obtained from the Taylor series as such:

$$q(x) \approx f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)\nabla^2 f(x_k)(x - x_k)$$

and the problem can now be translated into minimizing the approximation function, $q(x)$. Carrying on, we can apply the First Order Necessary Conditions (FONC) in order to find the potential minimizers of the functions. These state that if x^* is a local minimizer of f, continuously differentiable function, over a subset $\Omega \in \mathbb{R}^n$, then for any feasible direction d at x^*, the following statement holds:

$$d^T \nabla f(x^*) \geq 0,$$

where d satisfies: $\exists \alpha_0$ such that $\forall \alpha \in [0, \alpha_0]$:

$$x^* + \alpha d \in \Omega.$$

Applying the oultined FONC, we obtain the following equation by differentiation:

$$0 = \nabla q(x) = \nabla f(x_k) + \nabla^2 f(x_k)(x - x_k).$$

By setting $x = x_{k+1}$ the next point in finding the minimum, rearranging and assuming the hessian is positive definite, we get:

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k).$$

Note that we can make the assumption of the hessian being positive semidefinite as the Second Order Necessary Conditions for x^* to be a minimizer state exactly that. Adding a step-size strategy, we obtain that Newton method is in fact a type of descent algorithm:

$$x_{k+1} = x_k - \alpha_k \nabla^2 f(x_k)^{-1} \nabla f(x_k),$$

where, again, α_k is chose as: $\alpha_k = arg\min_{\alpha_k} f(x_k - \alpha_k \nabla^2 f(x_k)^{-1} \nabla f(x_k))$. Further on, the assumption of the hessian being positive definite makes the Newton direction a descent direction since if we multiply by the gradient we obtain:

$$\nabla f(x_k)^T d_k = -\nabla f(x_k) \nabla^2 f(x_k)^{-1} \nabla f(x_k) < 0.$$

2.2 Hydraulic Principles

In fluid dynamics, Bernoulli's Principle [30] states that for an inviscid flow (the flow of an ideal fluid that is assumed to have no viscosity) an increase in the speed of the fluid occurs simultaneously with a decrease in pressure or a decrease in the fluid's potential energy. This result follows directly from the principle of conservation of energy that states that the total energy (kinetic and potential) in an isolated system cannot change.

$$E_{kinetic} = \frac{1}{2} m v^2$$
$$E_{potential} = mgh,$$

where m is the mass of the object, v its velocity, g the standard gravity and h the height of the object for which the potential energy is computed. Thus, an increase in speed of the fluid results in increasing the kinetic energy as is stated by the formula that, in turn, determines a decrease in potential energy.

Mathematically, the Bernoulli principle can be written through Bernoulli's equation and is formulated as:

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = const,$$

where v is the velocity of the fluid, g is the standard gravity, z is the elevation or height of the point considered above a reference plane, p is the pressure at a certain point and ρ the density of the fluid. By multiplying by ρ we obtain the following equation:

$$\frac{\rho v^2}{2} + \rho g z + p = const,$$

or arrange it in a convenient way that will become clear later:

$$q + \rho g h = p_0 + \rho g z = const,$$

where q is the dynamic pressure, h is the piezometric head or hydraulic head and p_0 is the sum of static pressure and dynamic pressure and they are given by the following:

$$q = \frac{1}{2}\rho v^2$$

$$h = z + \frac{p}{\rho g}$$

$$p_0 = p + q.$$

Now, the constant terms we have expressed above can be normalized and the most common way to achieve this is true the total head or energy head:

$$H = h + \frac{v^2}{2g},$$

which would represent one of the unknowns of our problem, the unknown head at the nodes of the water network.

2.3 Mathematical Models in Hydraulics

Thus far, we have introduced some background that would be useful for a better understanding of the problem we are going to solve, which is computing the total energy at the nodes of the network (H), as well as the flowrates from the pipes of the network (Q). We will now take a look at the mathematical model that we use in order to solve the hydraulic problem.

2.3.1 Introduction

A water distribution network comprises a number of distinct elements that add up to form the topological structure of it. Thus, it contains links, which are denoted by the pipes we have in the network, the pumps and the valves. Also, there are nodes, which are denoted by reservoirs, tanks (components whose total head H_0 we know) and junctions, places at which we are interested in finding the energy head H and the flowrate Q.

At a higher level of design, such a water grid has two different stages:

- network flow analysis
- network definition and optimisation

The first part of the process implies determining the sizes of the linking elements that minimize a certain objective function, under the assumption of given physical properties and constraints, as well as the nodal demands and the costs involved in each link. The physical properties and constraints involved are the diameters of the pipes in use, the minimum head each node has to have, maximum velocity that can be accommodated in each pipe. Nevertheless, this is rather an 'a priori' stage in our problem as we will focus our attention on the second issue, of network flow

analysis as the input of our problem will consist of a given topological structure of the network.

The second stage, network flow analysis, is the main problem we are focusing on as it gives a measure of reliability and consistency of a particular water grid. Hence, we are trying to compute the flow rates and piezometric head at all the nodes of the network. This enables us to do more in-depth analysis and decide, for instance, if service requirements are met throughout the network. We have discussed a few methods for solving the network flow analysis problem including Local Gradient, Newton-Raphson, Linearization.

2.3.2 The model

The proposed method by Todini and Pilati [1] starts by proving the existence and uniqueness of the partly linear, partly non-linear system, which is to be solved, and then, the Newton-Raphson technique is applied to the problem of which we know the existence and uniqueness of the solution. The problem is then transformed to the problem of finding the recursive solution of a linear system. In addition, the resulting matrix has some advantageous properties being symmetrical, positive definite, which leaves space for implementing efficient schemes to solve the problem. We will now explain this text through the mathematical encoding of the water network we are dealing with. The problem we need to solve can be formulated as:

$$A_{12}H + F(Q) = -A_{10}H_0$$

$$A_{21}Q = q,$$

where

- $A_{12} = A_{21}^T$ (np, nn) incidence matrix representing the unknown head nodes and is defined as: $A_{12}(i,j) = \begin{cases} 1, & \text{if flow of pipe i enters node j} \\ 0, & \text{if pipe i and node j are not connected} \\ -1, & \text{if flow of pipe i leaves node j} \end{cases}$
- $A_{10} = A_{01}^T$ (np, no) Fixed head nodes incidence matrix
- Q (1, np) flowrates in each pipe
- H (1, nn) unknown nodal heads
- H_0 (1, no) fixed nodal heads
- $F(Q)$ (1, np) laws expressing head losses in pipes

with

- nn the number of nodes with unknown head
- no the number of nodes with fixed head
- np the number of pipes with unknown flow

The head losses function describes the water pressure that is lost in the pipes due to the friction. For our model, we will make use of the Hazen-Williams head losses function and write for each pipe i:

$$F_i(Q_i) = R_i|Q_i|^{n_i-1}Q_i,$$

with R_i, n_i constants.

2.3.3 Existence and uniqueness of solution

We are now faced with a set of non-linear equations due to the way our head loss function is defined. Before applying Newton-Raphson to solve the system, we need to establish the existence and uniqueness of the solution. We will achieve this by considering a new model we know is convex and, hence, has a unique solution. Further on, we will choose the model such that its optimality conditions coincides with the system that we require to solve. Unsurprisingly, we will integrate f_i to obtain the Content Model:

$$\min\ C(Q) = \frac{\sum_{i=1}^{np} R_i |Q_i|^{n_i+1}}{n_i+1} + \sum_{j=1}^{no} H_{0j} \sum_{i=1}^{np} A_{01}(j,i) Q_i$$

$$\text{s.t. } \sum_{i=1}^{np} A_{21}(j,i) Q_i - q_j = 0 \text{ , j = 1, nn.}$$

This is a constrained optimisation problem that can be translated into an unconstrained one by the means of Lagrange multipliers. Applying the optimality conditions will yield the following problem:

$$\min\ L(Q,\lambda) = \frac{\sum_{i=1}^{np} R_i |Q_i|^{n_i+1}}{n_i+1} + \sum_{j=1}^{no} H_{0j} \sum_{i=1}^{np} A_{01}(j,i) Q_i + \sum_{j=1}^{nn} \lambda_j \sum_{i=1}^{np} A_{21}(j,i) Q_i - q_j.$$

Now, the R_i and n_i terms are conveniently chosen to be positive so as to have a convex function L, which entails the solution of the Lagrangian exists and is unique and can be obtained by imposing the conditions necessary for an extreme (i.e. taking partial derivatives with respect to the variables we have):

$$\frac{\partial L}{\partial Q_i} = 0 \text{ with i = 1, np}$$

$$\frac{\partial L}{\partial \lambda_j} = 0 \text{ with j = 1, nn.}$$

The initial problem we need to solve can thus be represented by the following matrix equation:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & 0 \end{pmatrix} \begin{pmatrix} Q \\ \lambda \end{pmatrix} = \begin{pmatrix} -A_{10} H_0 \\ q \end{pmatrix},$$

where

$$A_{11} = \begin{pmatrix} R_1 |Q_1|^{n_1-1} & & & \\ & R_2 |Q_2|^{n_2-1} & & \\ & & \ddots & \\ & & & R_{np} |Q_{np}|^{n_{np}-1} \end{pmatrix}$$

is a (np, np) matrix. We can see that this system is very similar to the initial one and we can now deduce that the Lagrange multipliers λ we have are the unknown nodal heads of the junctions of our water network H. A simple substitution will yield the following system of non-linear equations:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & 0 \end{pmatrix} \begin{pmatrix} Q \\ H \end{pmatrix} = \begin{pmatrix} -A_{10}H_0 \\ q \end{pmatrix}.$$

2.3.4 Applying Newton-Raphson to solve the non-linear system

In order to solve the system of non-linear equations we can apply the Newton-Raphson technique outlined in section (2.4), assuming the A_{11} matrix does not become singular. Nevertheless, for practical reasons, we can impose a lower bound for the elements on the diagonal matrix. The iterative scheme can be obtained by differentiating the equations which we previously had with respect to Q and H, respectively:

$$\begin{pmatrix} NA_{11} & A_{12} \\ A_{21} & 0 \end{pmatrix} \begin{pmatrix} dQ \\ dH \end{pmatrix} = \begin{pmatrix} dE \\ dq \end{pmatrix},$$

where

$$N = \begin{pmatrix} n_1 & & & \\ & n_2 & & \\ & & \ddots & \\ & & & n_{np} \end{pmatrix}$$

and where

$$dE = A_{11}Q^k + A_{12}H^k + A_{10}H_0 \tag{2.13}$$

$$dq = A_{21}Q^k - q \tag{2.14}$$

representing the residuals to be iteratively reduced to 0 and Q_k and H_k the unknown flows and heads at iteration k. We will now aim at writing the iterative scheme we will need to solve. Consequently, we would like to get to a next state of H_{k+1} and Q_{k+1} as a function of H_k and Q_k.

To start off, we will denote:

$$NA11 = D^{-1}. \tag{2.15}$$

The inverse of the matrix system can be obtained analytically by a method called partitioning [10]:

$$\begin{pmatrix} D^{-1} & A_{12} \\ A_{21} & 0 \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

with:

$$B_{11} = D - DA_{12}(A_{21}DA_{12})^{-1}A_{21}D$$

$$B_{12} = DA12(A21DA12)^{-1}$$

$$B_{21} = (A_{21}DA_{12})^{-1}A_{21}D$$

$$B_{22} = -(A_{21}DA_{12})^{-1}.$$

The solution of the initial matrix system can be found bearing in mind that:

$$dQ = B_{11}dE + B_{12}dq$$

$$dH = B_{21}dE + B_{22}dq.$$

A substitution of the two sets of equations above, yields:

$$dH = (A_{21}DA_{12})^{-1}A_{21}D(A_{11}Q^k + A_{12}H^k + A_{10}H_0) - (A_{21}DA_{12})^{-1}(A_{21}Q^k - q)$$

$$dQ = (D - DA_{12}(A_{21}DA_{12})^{-1}A_{21}D)(A_{11}Q^k + A_{12}H^k + A_{10}H_0) + DA_{12}(A_{21}DA_{12})^{-1}(A_{21}Q^k - q),$$

which gives

$$dQ = D(A_{11}Q^k + A_{10}H_0) - DA_{12}(A_{21}DA_{12})^{-1}(A_{21}D(A_{11}Q^k + A_{10}H_0) + (q - A_{21}Q^k)).$$

Using the Newton-Raphson iterative scheme as:

$$dQ = Q^k - Q^{k+1}$$

$$dH = H^k - H^{k+1},$$

one finally obtains the recursion to be solved for optimal values of head loss and flowrate:

$$H^{k+1} = -(A_{21}N^{-1}A_{11}^{-1}A_{12})^{-1}(A_{21}N^{-1}(Q^k + A_{11}^{-1}A_{10}H_0) + (q - A_{21}Q^k))$$
$$Q^{k+1} = (I - N^{-1})Q^k - N^{-1}A_{11}^{-1}(A_{12}H^k + A_{10}H_0).$$

2.4 Smoothing and First Order Methods

In this section we will discuss a unifying framework that combines the smoothing approximation of non-differentiable functions with fast first order algorithms (that use only first derivative information) in order to solve non-smooth convex minimization problems. It can be proved that regardless of the convex non-smooth function involved and the first order method used, one can achieve an efficiency rate of $O(\epsilon^{-1})$ by solving the modified smooth problem, keeping a very good accuracy of solutions.

2.4.1 Introduction to smoothing

The motivation behind this technique is that it does not rely on schemes involving the subgradient as it redefines the problem in terms of continuously differentiable functions, which aim to approximate the original problem as well as possible. Further on, an enhancement in converge rate is achievable from $O(1/\epsilon^2)$ to $O(1/\epsilon)$ where ϵ denotes the error in solution and is particular to each problem although typical values may be around 10^{-6}.

The problems we will consider here are general enough to form a basis for a wide range of convex optimisation problems:

$$\min_x F(x) + h_1(x) + h_2(x),$$

where F is smooth and h_1 and h_2 are non-smooth. The reason why we pick two non-smooth functions is to make the framework even more generic and explore the option of applying partial smoothing where we test to see if smoothing just one part of the objective function is in fact better that smoothing every function. Nonetheless, in our case there is only one non-smooth

function and so the decision is trivial so long as we can prove that smoothing is beneficial.

The problem can then be reformulated using the smooth approximations H_1 and H_2 for h_1 and h_2, respectively, as:

$$\min_x F(x) + H_1(x) + H_2(x).$$

Or, for completeness purposes, only apply partial smoothing to obtain the following generic problem:

$$\min_x F(x) + h(x),$$

where F denotes the differentiable function and h the non-smooth, non-differentiable part. This problem can be solved by fast first order algorithms in $O(1/k^2)$.

We will now take a look at smoothable functions, define what a fast first order optimisation algorithm is and obtain a bound on the rate of convergence of $O(1/\epsilon)$.

2.4.2 Smoothable functions

In order to progress with the explanation of this method, we will need to formally define the notion of smoothable functions. We will now define the concept of a smoothable function and the corresponding smooth approximation of a given non-differentiable convex function.

Definition 1: Let $g : E \to (-\infty, \infty]$ be a closed and proper convex function and let $X \subseteq dom\ g$ be a closed convex set. The function g is called (α, β, K)-smoothable over X if there exists β_1, β_2 satisfying $\beta_1 + \beta_2 = \beta > 0$ such that for every $\mu > 0$ there exists a continuously differentiable convex function $g_\mu : E \to (-\infty, \infty)$ such that the following hold:

1. $g(x) - \beta_1\mu \leq g_\mu(x) \leq g(x) + \beta_2\mu$
2. The function g_μ has a Lipschitz gradient over X with Lipschitz constant, which is less than or equal to $K + \alpha/\mu$. That is, there exists $K \geq 0$, $\alpha > 0$, such that:

$$\|\nabla g_\mu(x) - \nabla g_\mu(y)\| \leq \left(K + \frac{\alpha}{\mu}\right) \|x - y\|.$$

The function g_μ is called a μ-smooth approximation of g over X with parameters (α, β, K). If a function is smoothable over the entire vector space E, then it will just be called (α, β, K)-smoothable.

2.4.3 Problem formulation

We are interested in solving the convex problem given by:

$$H^* = \min\{H(x) = g(x) + f(x) + h(x) : x \in E\}, \tag{2.16}$$

where we consider the following functions:

- $h : E \to (-\infty, \infty]$ is an extended valued closed proper convex function that is sub-differentiable over its domain, which is denoted by $X = dom\ h$

- $f : X \to (-\infty, \infty)$ is a continuously differentiable function over X whose gradient is Lipschitz with constant L_f
- $g : X \to (-\infty, \infty]$ is a (α, β, K)-smoothable function over X

The problem defined in 2.19 is general enough to cover a large class of optimisation problems, however, for our purposes, the non-smooth function h will be 0 as we only have one function that is non-smooth and the objective is to determine if smoothing it will be advantageous. We will carry on with this formulation bearing in mind that h may be removed if we will.

Consequently, the smoothed problem will become:

$$H_\mu^* = \min\{H_\mu(x) = g_\mu(x) + f(x) + h(x) : x \in E\}, \tag{2.17}$$

where g_μ is indeed the μ-smooth approximation of function g.

Now, we should be able to use any algorithm for solving 2.20. For simplicity, we will rewrite the problem in a different form to represent the smooth and non-smooth parts:

$$D^* = min\{D(x) = F(x) + h(x) : x \in E\}, \tag{2.18}$$

where F is the smooth part and h the non-smooth.

This problem is called the input convex optimisation model and is characterized by the following data:

- h is an extended valued closed convex function that is sub-differentiable over its domain $dom\ h$
- F is a continuously differentiable convex function over $dom\ h$ whose gradient is Lipschitz with constant L_f.

The input convex optimisation model is therefore characterized by a triplet (F, h, L_f) satisfying the above premises.

2.4.4 Fast Iterative Methods

We will now take a look at another notion that classifies the first order optimisation algorithms and will help us establish the results that improve the theoretical rate of convergence and also help us make a link between the smoothing parameter of a function μ and the error we set for convergence ϵ.

Thus, we will define a fast iterative method as:

Definition 2: Let (F, h, L_f) be a given input convex optimisation model with an optimal solution x^* and let $x_0 \in E$ be any given initial point. An iterative method $\mathcal{M}$ for solving problem 2.21 is called a fast method with constant $0 < \Lambda < \infty$, which possibly depends on (x_0, x^*) if it generates a sequence $\{x_k\}_{k \geq 0}$ satisfying for all $k \geq 1$:

$$D(x_k) - D^* \leq \frac{L_f \Lambda}{k^2}.$$

What is interesting from such a setting is that we don't care about the specific method $\mathcal{M}$ we employ as we can show that independent of that, the problem in 2.21 with appropriately chosen smoothing parameter μ, can achieve an ϵ optimal solution is no more than $O(1/\epsilon)$ iterations, which clearly is an improvement over the usual sub-gradient methods that are bound by $O(1/\epsilon^2)$.

2.4.5 Link between μ and ϵ

This next result will help us formally establish the number of steps required for ϵ convergence of a smooth convex optimisation problem and also give us information on the values of μ.

We will now state the theorem that gives us the result:

Theorem 1: Let $\{x_k\}$ be the sequence generated by a fast iterative method $\mathcal{M}$ when applied to problem 2.20, that is, to the input optimisation problem $(f + g_\mu, h, L_{f+g_\mu})$. Suppose that the smoothing parameter is chosen as:

$$\mu = \sqrt{\frac{\alpha}{\beta}} \frac{\epsilon}{\sqrt{\alpha\beta} + \sqrt{\alpha\beta + (L_f + K)\epsilon}}. \tag{2.19}$$

Then for

$$k \geq 2\sqrt{\alpha\beta}\Lambda \frac{1}{\epsilon} + \sqrt{(L_f + K)\Lambda} \frac{1}{\sqrt{\epsilon}} \tag{2.20}$$

it holds that

$$H(x_k) - H^* \leq \epsilon, \tag{2.21}$$

where H^* is, as usual, the optimal solution of 2.20 and $H(x_k)$ the value of the objective H at the k-th iteration. Please see Appendix (10.1) for a complete proof of the theorem.

3 | Numerical Analysis

In this chapter we will discuss about the methodology used in order to solve the hydraulic problem. Moreover, we are going to go into specific performance details that greatly enhanced the execution time of the problem, as well as talking how well specific methods actually behave for our case.

3.1 Introduction

As we commence the discussion of the problem, it is important to discuss about the matrix operations we have and deduce how much time is spent of various matrix operations in order to achieve better performance.

Among the computations we have to perform, we are concerned with profiling the following matrix operations, which occur significantly:

- Addition, Subtraction
- Multiplication
- Computing the Transpose
- Computing the Inverse

These are some of the most frequent operations we need to perform in order to solve the optimisation problem. In addition to these, there can be more specific operations and matrix manipulation techniques that we need to apply, depending on the type of solver we use.

We will take a look at some optimal ways in which we can perform some of the following operations:

- Cholesky Factorization
- Incomplete Cholesky Factorization
- LU Decomposition
- LDLT Decomposition

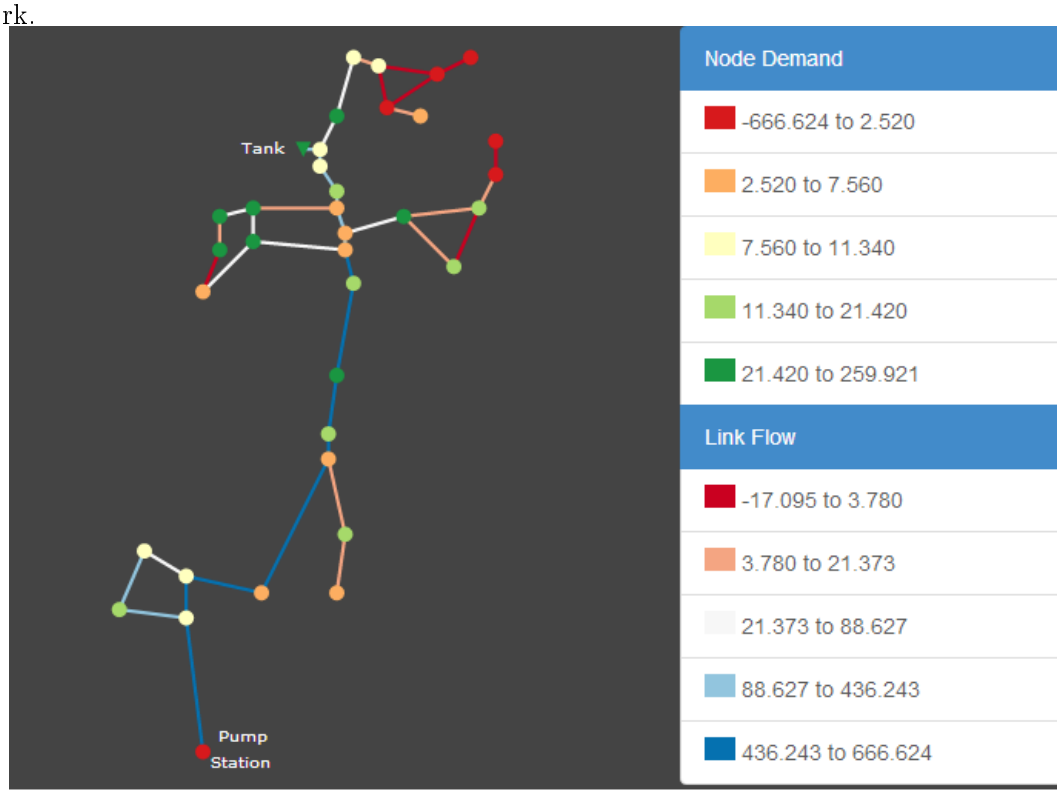
Of course, there quite a few libraries and schemes that offer these kind of operations and there is no one library of which we can say is the absolute best. Rather, it is important to analyse a variety of such techniques and determine which one is best for the type of problems you require to solve.

3.2 Operation Analysis

We will now refer to the most common operations in turn and determine the CPU time required to perform them.

Also, an important aspect to point out is that the vast majority of operations performed are carried on on square matrices whose dimensions can range from 10 by 10 to 4000 by 4000. In general, we will not handle the case where matrices are larger than 5000 by 5000 elements.

Figure 3.1: A simple network that is representative of the sparse connectivity of the nodes in the network.



Another important aspect, which is illustrated in Figure 3.1, is that the problem that we formulate consists of sparse matrices since the connectivity of the water distribution network is usually not very high. For instance, a node in the network will almost always be connected to some relatively small number of neighbours (with respect to the distance between the nodes) and not have links all across the network.

3.2.1 Addition and Subtraction

Addition and subtraction are the easiest operation we need to consider as the time spent doing matrix additions or subtractions is negligible when compared to multiplication or decompositions. Thus, even for large matrices we need not worry too much about these operations as they are executed quite fast, in $O(n^2)$.

Moreover, you can even overlook the sparsity nature of the matrices since even for a 2000 by

2000 matrix, the operations are done extremely fast. Nonetheless, we aim at reducing the overall computational cost as much as possible so we will optimize these operations as well. We will achieve this by improvements done to the matrix multiplication which is discussed next.

3.2.2 Multiplication

How we perform matrix multiplication if of paramount importance to our application as it is significantly more time consuming than plain addition and subtraction. This comes as no surprise as matrix addition and subtraction are implemented in $O(n^2)$, whereas multiplication is done in $O(n^3)$. Although there are different approaches to multiplication that reduce the overall time complexity up to even $O(n^{2.373})$ (due to Williams), the computational time for a matrix of size 5000 by 5000 elements is still large for our purposes.

Since even the best implementation of matrix multiplication won't give us a very good result, we will take a closer look at the nature of the problem and determine that almost any matrix multiplication involves sparse matrices, consequently making us solve additions and multiplications of 0.

The nature of the problem allows us to use a different storage mechanism than the usual bi-dimensional array used in standard libraries as it would be clearly inefficient since we will end up using a lot of memory storing values of 0. We will discuss some approaches of representing matrices, which become more and more interesting as the number of non-zero elements decreases.

- Coordinate list

  This format stores the non-zero elements of the matrix as tuples of the form (row, column, value) and is particularly fast when inserting, deleting or searching for an element especially when implemented by a good data structure such as a hash table. Another important factor is it supports fast conversions to other formats, which guarantees better computational time for arithmetic operations.

- Yale format

  The Yale Sparse Matrix Format stores an initial sparse matrix of dimension m by n, say, in row form using three one-dimensional arrays. We will refer to NNZ as the number of *non-zero* entries of the matrix.

  - The first array, of length NNZ, holds all non-zero entries of the matrix in left-to-right, top-to-bottom (row-major) order.
  - The second array, of length $m + 1$ (i.e. one entry per row, plus one). This array acts as an index into the first array as it points to the index of the first occurrence of a non-zero element from the first index.

    Thus, the first m entries hold the indexes (as stored in the first array) of the first occurrence of a non-zero element for each row in turn. So for any i, which is a valid row index, the entry i in the second array will have the index from the first array, which corresponds to the first non-zero element on row i.

    Finally, the last index holds the total number of non-zero elements in the matrix.

– The third array, of length NNZ, contains the column index of each non-zero entry in the matrix.

We will now look at an example to clarify the Yale format. Suppose we have the following matrix:

$$\begin{pmatrix} 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 5 & 6 & 0 & 0 & 0 \\ 0 & 0 & 7 & 8 & 9 \end{pmatrix}.$$

We note that $NNZ = 9$. The first array will therefore be:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix},$$

as it simply contains the non-zero entries. Moreover, the second array is as:

$$\begin{pmatrix} 0 & 2 & 2 & 3 & 4 & 6 & 9 \end{pmatrix},$$

as the first non-zero element in the first row is 1 with index 0 in the first array. The second and the third rows have the same index since there is no non-zero element in the second row and in the third row, we encounter 3, which has index 2 in the first array. The last entry coincides with NNZ.

Lastly, the third array is:

$$\begin{pmatrix} 2 & 3 & 1 & 4 & 0 & 1 & 2 & 3 & 4 \end{pmatrix},$$

as the first non-zero entry, 1, is in column 2. The next entry, 2, is in column 3 and so on.

- Compressed Sparse Row

  The compressed sparse row is an implementation of the Yale format described above with the slight difference of storing the column information before the row information. Nevertheless, in our case, the matrices comprise of information nicely distributed across the rows and columns (i.e. its transpose would be just as dense on rows as it is on columns).

- Compressed Sparse Column

  With compressed sparse column, we have a similar implementation to the Compressed Sparse Row, with the only exception that the indexing is done by column instead of rows and the array of columns is transformed in the array of rows.

3.2.3 Transpose

Transposing a matrix is still not an expensive operation, especially when compared with matrix multiplication. A naive implementation would yield a good computational time, but for optimality we refer again to the sparse storage of a matrix whereby it becomes easier to transpose as we only iterate through the non-zero elements of the matrix (significantly fewer than n^2 - the size of the matrix) swapping the first two elements of the triplet in the case where Coordinate list storage is used.

3.2.4 Cholesky

Cholesky factorization is an important computational step in solving the nonlinear equations of our problem. It appears when a direct method is used, thus reducing the system of linear equations to solving:

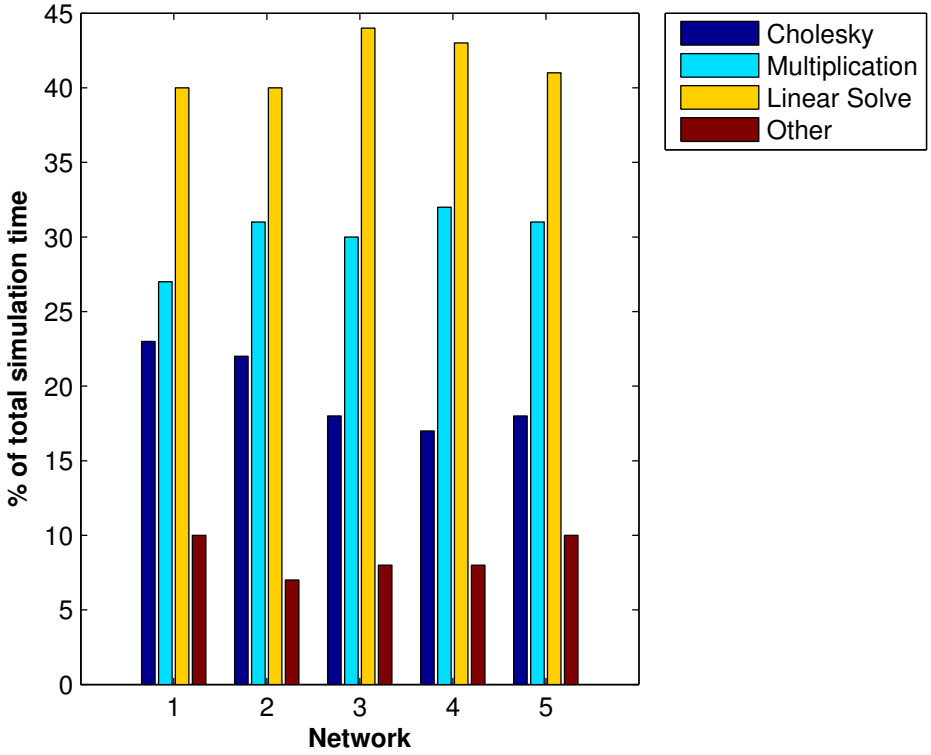
$$LL^T x = b,$$

with L a lower triangular matrix.

This decomposition of matrix A into LL^T can only be done if matrix A is Hermitian (complex generalization of symmetric) and positive-definite.

Moreover, it appears in iterative methods used such as the Conjugate Gradient Method as a speed up as the factorization is used as a preconditioner. In this case, due to the sparse nature of the problem, it will be more accurate to refer to it as the Incomplete Cholesky factorization since a full, normal factorization will require more time since it assumes to be done on a dense system.

Figure 3.2: Amount of time spent (%) from the total time of the simulation performing different tasks. Please note that this is in the naive implementation case and the purpose of the figure is to illustrate where improvements can be made.



In Figure 3.1, you can see how much time each part of the simulation requires relative to the whole time required. We see that matrix-matrix and matrix-vector multiplications require significant CPU time and so we need to address this issue by using a more efficient storage scheme for matrices.

3.3 Solving systems of linear equations

The recursive Newton-Raphson scheme involves solving an equation of type

$$H_{k+1} = A^{-1}b.$$

This is equivalent to solving a linear system of equations:

$$Ax = b.$$

There are many ways in which one can go about this and we will discuss some of the approaches we have considered as part of our analysis and state the ones that are the most convenient to use in our case. To start off, we will divide the discussion into two separate techniques:

3.3.1 Direct Methods

We will start by taking a look at some direct methods of solving systems of linear equations and see what each of them involve in terms of implementation.

a. *Gaussian Elimination*
In Gaussian Elimination, we first manipulate the system so as to reduce is to a triangular form where the last equation typically contains one unknown, the penultimate contains two and so on. If there there are n equations with n unknowns, then if the rank of the matrix is equal to the rank of the extended matrix, we obtain a unique solution.

Using this triangular system, we proceed onto the substitution stage, whereby we substitute the unknowns from equation p with the values we have found from the previous equations ($p+1$, $p+2$, ..., n).

The latter step involves obtaining the *echelon* form, which is achieved through various row operations:

- Swapping two rows
- Scaling a row
- Add to one row a scalar multiple of the other

b. *LU Factorization*
With LU Factorization, we decompose our matrix A in terms of upper and lower triangular matrices so as to rewrite the initial problem as

$$LUx = b.$$

Thus, the method basically reduces to applying the second step of Gaussian Elimination twice. Once to solve

$$Ly = b$$

and then to solve

$$Ux = y.$$

Given that both matrices are triangular, the process is very efficient as the substitution stage can be solved quickly.

c. *LDU Factorization*
A variation of the LU Factorization whereby matrix D is diagonal and L and U are lower and upper triangular, respectively, with the addition of them having only elements of 1 on the diagonal.

d. *LLT Factorization*
This is the Cholesky Factorization, which decomposes a Hermitian positive semidefinite matrix as

$$A = LL^T,$$

with L a lower triangular matrix. When applicable, this method is considered to be more efficient than the LU Factorization.

e. *LDLT Factorization*
This is a variant of the Cholesky Factorization, which uses matrix D as a diagonal matrix and allows only elements of 1 on the diagonal of matrix L. An advantage of this variation is that we do not require any square root computations as we simply set to 1 the element on the diagonal matrix.

f. *Singular Value Decomposition*
SVD is presented as a future consideration to the project. In this methods, we factorize a matrix A as:

$$A = U\Sigma V^*,$$

where U is a unitary matrix, Σ is a rectangular diagonal matrix and V^* is the conjugate transpose of V, another unitary matrix. The diagonal entries of Σ represent the singular values of A, while the columns of U and V are the left-singular and right-singular vectors of A, respectively.

3.3.2 Iterative Methods

As opposed to direct methods, iterative methods do not provide an exact solution to the system we are solving. The upside is that we can choose to step whenever we want and being an iterative method we know that we will always approach the solution, whereas with direct methods we do not have the option of stopping at a particular point.

The question of when to stop is problem dependent and it relies on the accuracy level desired. In general, there is an ϵ level after which we are satisfied with the solution obtained.

These are some of the iterative methods for solving systems of linear equations that we considered as part of the analysis:

a. *Jacobi*
Jacobi is perhaps the most basic and simplest iterative method used in practice as it solves $Ax = b$ by taking an initial guess of what x might be and then applying the following iterative procedure

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i}^{n} a_{ij} x_j^{(k)} \right).$$

And we do so until the residual difference is less than the pre-established tolerance.

b. *Gauß-Seidel*
Gauß-Seidel comes with a slight improvement over the Jacobi method as it gives a faster convergence rate by using more up-to-date data. Thus, the iterative procedure used to update the latest unknowns becomes

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right),$$

which uses, in addition to the updates of the k^{th} iteration, the already updated, previously calculated unknowns of the $(k+1)^{th}$ iteration.

A useful result is that convergence of both Gauß-Seidel and Jacobi is achieved if the matrix A is strictly row diagonally dominant, which means that the following condition must hold

$$|a_{ii}| > \sum_{j \neq i} |a_{ji}|.$$

However, there might be situations in which the above does not hold and the methods still converge.

c. *Successive over-relaxation*
Successive over-relaxation (SOR) tries to give a generalization of Gauß-Seidel and offers more optimal convergence by using

$$x_i^{(k+1)} = (1-\omega) x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right),$$

where $0 < \omega < 2$ is a necessary condition for the convergence of SOR. If the matrix A is positive definite as well, then the condition for convergence is also sufficient.

d. *Conjugate Gradient Method*
Looking back at what our initial problem was, we can observe some characteristics of our A matrix:

$$H^{k+1} = -(A_{21} N^{-1} A_{11}^{-1} A_{12})^{-1} (A_{21} N^{-1} (Q^k + A_{11}^{-1} A_{10} H_0) + (q - A_{21} Q^k)),$$

where

$$A = (A_{21} N^{-1} A_{11}^{-1} A_{12}),$$
$$x = H^{k+1},$$
$$b = -(A_{21} N^{-1} (Q^k + A_{11}^{-1} A_{10} H_0) + (q - A_{21} Q^k)).$$

In our case as the equation can be rearranged to

$$(A_{21} N^{-1} A_{11}^{-1} A_{12})^{-1} H^{k+1} = -(A_{21} N^{-1} (Q^k + A_{11}^{-1} A_{10} H_0) + (q - A_{21} Q^k)).$$

We note that the resulting matrix A is symmetric, positive definite and Stieltjes type (non-positive off-diagonal entries). The solution of the system can be transformed to the solution to the minimization of a quadratic convex function:

$$J(x) = \frac{1}{2} x^T A x - b^T x,$$

which can be solved using the following recursive conjugate scheme:

$$r_o = Ax_o - b$$

$$p_o = r_o,$$

for iteration $k = 0$. And

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k + \alpha_k A p_k$$

$$\beta_{k+1} = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}$$

$$p_{k+1} = r_{k+1} + \beta_{k+1} p_k,$$

for $k > 0$.

The Conjugate Gradient Method is considered a fast iterative method [13]. However, it may require a large number of iterations before it converges to the solution. In general, if the size of matrix A is nn it may require up to $2nn$ or $2.5nn$ iterations to converge, which makes it infeasible even for $nn \approx 1000$.

e. *Modified Conjugate Gradient Method*
In addition, if we use the aforementioned properties of A being symmetric, positive definite, we might use the Cholesky factorization:

$$A = MM^T.$$

Now we can slightly modify the recursive scheme to include M as a preconditioner to the Conjugate Gradient Scheme:

$$r_o = Ax_o - b$$

$$p_o = r_o,$$

for iteration $k = 0$. And

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k + \alpha_k A p_k$$

$$s_{k+1} = (M^T)^{-1} M^{-1} r_{k+1}$$

$$\beta_{k+1} = -\frac{s_{k+1}^T A p_k}{p_k^T A p_k}$$

$$p_{k+1} = s_k + \beta_{k+1} p_k,$$

which usually reaches convergence in about 30–50 iterations, regardless of how large the system we need to solve is (discussed in [9] and [14]). One further note is that performing the full Cholesky factorization is inefficient and instead we will choose to perform an incomplete Cholesky factorization, which takes into account the sparsity of our problem.

3.4 Libraries and capabilities

According to the main points discussed in sections 3.2 and 3.3, we focused our attention on optimizing the matrix multiplication and decomposition part, as well as the linear system solver.

We will talk about the numerical analysis we have performed, starting from a naive implementation which was sub-optimal, up to the current one which performs better than the industry standard.

1. **LAPACKPP**
   This library is C++ specific library and is based on $LAPACK$, the C version and $BLAS$. It is a very old library and provides a very robust implementation for linear algebra operations. It also has the capability of running efficiently on shared-memory and parallel processors as opposed to other packages that don't handle this problem very well as they have a complicated memory hierarchy, which restricts them to do more memory management operations, rather than floating point operations.

   At the beginning, we have mainly focused on the dense capabilities of the package and, as expected, performance was sub-optimal. Dense matrix-matrix or even matrix-vector multiplications would take longer than what the state of the art solvers would take for solving the entire optimisation problem. More concretely, for relatively large systems of 2000x2000 elements, we were solving the whole optimisation problem in $O(\text{minutes})$ as opposed to the standard $O(100ms)$.

   When we explored the sparse nature of the problem, we found that the performance has improved significantly going down from $O(\text{minutes})$ to $O(s)$. However, there still is room for improvement and we shall see that we can bring this number even below the state-of-the-art solvers' $O(100ms)$.

2. **Boost**
   As Boost was already an implementation dependency, we decided to implement the solver using its support for fast linear algebra operations. However, since the implementation is based on $BLAS$ and $LAPACK$ with only the bindings being changed to agree with the new namespace, we obtained similar results to the ones described above.

   Nonetheless, the API seemed nicer and easier to use and it also integrated better with the existing codebase since we didn't require any other dependencies. This was a reason for adopting it as a solution, but we were still trying to investigate other options.

3. **SuiteSparse**
   This library is widely used in different large projects including Google projects or Matlab for implementing operations such as finding the inverse or matrix decompositions. It implements a variety of packages and it also relies partially on $BLAS$ for sparse QR decomposition, Cholesky and LU decomposition.

   The problem with *SuiteSparse* is that it had a rather complicated API, which required the developer to do more preparation than required and, in particular, decompositions were not as fast as expected. The Cholesky factorization used as a preconditioner for the Conjugate Gradient Method required more time than the whole algorithm.

4. **Eigen**
   Finally, we consider a more recent linear algebra library, which is widely used too, by companies such as Google and in robotics and computer graphics applications.

   Eigen is a good choice not only because it has its own implementation and thus no extra dependencies, but it also has the nicest and simplest API among all the tools with which we have experimented.

   Moreover, the results we obtain are improved when compared to the other libraries up to even $O(10ms)$ in some cases where $O(100ms)$ is required by the standard in industry. Consequently, our implementation is currently based on Eigen.

   The reason why this outperforms even the state-of-the-art hydraulic solvers is because there is no extra cost associated with carrying on the operations and the API is more complete and flexible than in the other packages. Hence, we do not require explicit conversions between matrix types, nor do we face the problem of not being able to matrices in their decomposed form.

An important note is that we have not considered the use of GPU or FPGA computation due to the fact that the boost in performance is only achieved for considerably larger linear systems [5]. For our purposes (systems no larger than 5000 by 5000), CPU performance is optimal.

3.5 Sensitivity to initial conditions

An important trait of any optimisation algorithm or iterative algorithm is the convergence rate at which the optimality constraints are met and thus, the optimal solution is found.

A direct impact to the convergence rate is given by the initial estimate of the solutions. While the problem we solve is indeed convex and has a unique solution as it can be seen in section 2.3.3, convergence is guaranteed regardless of the initial solution we choose. Nonetheless, we will investigate what impact choosing the initial estimate has on the number of iterations required for convergence.

The iterative system we need to solve is:

$$H^{k+1} = -(A_{21}N^{-1}A_{11}^{-1}A_{12})^{-1}(A_{21}N^{-1}(Q^k + A_{11}^{-1}A_{10}H_0) + (q - A_{21}Q^k))$$
$$Q^{k+1} = (I - N^{-1})Q^k - N^{-1}A_{11}^{-1}(A_{12}H^k + A_{10}H_0),$$

which requires choosing good values for H and Q at time $k = 0$.

From a practical perspective, it was determined that even a naive initial setting yields fast convergence for a large class of water distribution networks. The reason for this comes from the fact that regardless of the topological structure of the network, customer demand at junctions will virtually always correspond to a certain value. In other words, demand does not vary significantly enough for us to see significant change in pressure or water flowrate, too.

Table 3.1: Number of iterations required for convergence for different networks and different values of H with fixed Q.

H	Network 1 (7 x 9)	Network 2 (91 x 113)	Network 3 (865 x 950)	Network 4 (2303 x 2369)	Network 5 (4577 x 4648)
130	7	8	27	25	43
1300	7	8	27	25	43
13000	7	8	27	25	43
130000	7	8	27	25	43
13	7	8	27	25	43
0.13	7	8	27	25	43
0.0013	7	8	27	25	43

Moreover, it was noted that values for piezometric head H and water flowrate Q at optimality are *relatively* close in several cases to $H = 130$ and $Q = 0.03$. Therefore, the initialization done is as:

$$H_i = 130 \ \forall i$$

$$Q_i = 0.03 \ \forall i.$$

We will take three approaches to this analysis by first varying just H, then varying just Q and finally, systematically varying both H and Q.

3.5.1 Varying head loss H

We have let H vary as shown in Table 3.1 and found that convergence is not affected in any way even for large variations of H while keeping the water flowrate Q constant.

The reason why we see a difference in the number of iterations required for convergence comes from the network size, but also from the solutions we obtain. In other words, the closer the initial estimate is to the optimal solution, the fewer number of iterations required for convergence. This is because Newton-Raphson has a higher rate of convergence when we are close to the solution [17].

3.5.2 Varying water flowrate Q

Next, we will investigate what happens with the convergence rate of the iterative algorithm when we vary Q and keep H at a fixed value. Table 3.2 displays the results of the simulations carried on.

It is interesting to note that Q does indeed affect the number of iterations required for convergence by the Newton-Raphson iterative scheme, however, we are also guaranteed to obtain the optimal solution after still a relatively small number of iterations. Despite changes in iterations are relatively large (e.g. 10 to 36 for Network 4), the number in itself is still feasible computationally and it does not add too much overhead as the time required is still under one second.

One other interesting fact is the minimum iterations required is network-dependent and is clearly achieved for the value that is closest to the solution. Hence, each network presents with certain

Table 3.2: Number of iterations required for convergence for different networks and different values of Q with fixed H.

Q	Network 1 (7 x 9)	Network 2 (91 x 113)	Network 3 (865 x 950)	Network 4 (2303 x 2369)	Network 5 (4577 x 4648)
0.03	7	8	27	25	43
0.003	6	9	22	11	32
0.0003	9	9	20	10	27
0.00003	11	9	21	15	24
0.000003	14	9	14	22	26
0.3	10	11	29	25	36
3	13	14	33	28	49
300	18	21	42	34	56
300000	24	27	44	36	63

Table 3.3: Average value for optimal Q for each of the networks.

Network 1 (7 x 9)	Network 2 (91 x 113)	Network 3 (865 x 950)	Network 4 (2303 x 2369)	Network 5 (4577 x 4648)
0.00432971	0.0261513	2.24896e-05	7.70001e-05	2.18882e-05

optimal values for Q and being as close as possible to the solution from the beginning improves the complexity of the algorithm. In Table 3.3 you may view the average value for Q for each of the network that corresponds to the closest value of initial Q found in table 3.2 for which the number of iterations is also minimal.

3.5.3 Systematically varying both H and Q

As it may be hinted from the previous two results, systematically varying both variables of the system will produce no different alteration than the one which was present when varying Q. As a consequence, the results that were displayed in Table 3.2 are similar to the ones obtained in this case. For any value of Q, regardless of the value of H (choose any value from Table 3.1), we obtain the same converge performance as if we would have kept H fixed.

3.6 Headloss function

In this section we will discuss the importance and impact of the head loss functions we use in our computation. This may represent the only non-linear term in our equations and, even more, it may be a non-differentiable function, both of which may create problems when applying Newton-Raphson or proving or finding an optimal solution.

3.6.1 Hazen-Williams equation

The Hazen-Williams equation is an empirical relationship that relates the water flowrate through a pipe and the physical properties of them, which give a measure of the friction in the pipe that

then translates into the loss in water pressure or piezometric head.

The equation is given by the relation

$$f(Q_i) = R_i|Q_i|^{n_i-1}Q_i, \tag{3.1}$$

where R_i is a constant that gives information about the material properties and friction coefficient. Q_i represents the water flowrate through a pipe i.

3.6.2 $A11$ in optimality conditions

In our problem, $A11$ denotes a diagonal matrix whose entries represent the head loss function laws expressed at each pipe of the network. However, they are not precisely the laws for each pipe, but rather the laws after integration. This is necessary because we were trying to obtain a convex optimisation problem whose KKT conditions were similar to the ones of the original problem, so that we would be able to prove existence and uniqueness of the problem. We refer to section 2.3.3 for more details.

Consequently, the matrix takes the form

$$A_{11} = \begin{pmatrix} R_1|Q_1|^{n_1-1} & & & \\ & R_2|Q_2|^{n_2-1} & & 0 \\ & & \ddots & \\ 0 & & & R_{np}|Q_{np}|^{n_{np}-1} \end{pmatrix},$$

which represents the nonlinear component of our system of equations. Moreover, it is important to guarantee that this matrix does not become singular at any point of the computation as otherwise we will no longer satisfy the condition of the system having a unique solution.

In practice, water flow through a pipe may get very close to 0 when the pressure of the two nodes, which are connected through the pipe, is equivalent and the elevation of the two nodes is the same as well. In this case, Q will be reduced significantly and it may produce bad effects from a computational point of view. Therefore, our model proposes a lower bound on the diagonal entries of the $A11$ matrix that is set to 10^{-5}.

The problem with this restriction is that it does not accurately reflect the water distribution network from a physical point of view and that our optimal solution might be influenced and not be the one desired. The optimality conditions for this problem, the ones, which are checked after each iteration, are:

$$A11Q + A12H = A10H, \tag{3.2}$$

where, as before, $A12$ and $A10$ are matrices that provide the connection of nodes in the network and Q and H are the flowrate and head, respectively and are the variables for which we solve.

Since we cannot modify the $A11$ matrix in the Newton-Raphson iteration as we require it not to

be singular, we have kept it as is in the iteration, but checked the optimality condition given at equation (3.2) using the *actual* values in $A11$, which may possibly be 0. Nonetheless, the results showed no difference in terms of rate of convergence for the two scenarios. Thus, they are similar to the ones presented in Table 3.1.

4 Smooth Framework

The aim of this chapter is to prove that a new optimisation framework is better for solving the hydraulic equations. Currently, in the formulation of the hydraulic equations we have a non-differentiable function, which does not satisfy the conditions in which Newton-Raphson can be applied. In order to address the problem, we are going to write a smooth formulation of the problem, one that uses a continuously differentiable function as an objective function.

Furthermore, we will establish that the number of iterations required for convergence is better than the usual subgradient methods [2] and we will discuss the situations where our smooth formulation best approximates the original problem.

4.1 Introduction

The main difficulty we have when solving the hydraulic problem comes from the fact that the head loss function used is a non-differentiable function and we are currently using Newton-Raphson to solve the nonlinear set of equations. This is a big problem since Newton-Raphson assumes that our objective function is twice continuously differentiable.

The Newton-Raphson method uses the Taylor expansion to minimize a general nonlinear function, which is assumed to be difficult to minimize. Hence, we transform

$$\min_{x \in \mathbb{R}^n} f(x), \tag{4.1}$$

into minimizing the function $f(x)$ obtained from the Taylor expansion:

$$f(x) = f(x_k) + \nabla f(x)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x)(x - x_k). \tag{4.2}$$

We will now apply FONC (see section 2.1.6) to obtain:

$$\nabla f(x_k) + \nabla^2 f(x_k)(x - x_k) = 0 \tag{4.3}$$

as our necessary condition for a minimum. By assuming that the Hessian is positive definite (so the function is strictly locally convex), we obtain the following iterative scheme:

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k), \tag{4.4}$$

where x from equation (4.3) was replaced by x_{k+1} in equation (4.4) just to make clear which point we will use at the next iteration.

A key point to notice is that only when f is continuously differentiable and $\nabla f(x^*) = 0$, we can say that x^* is an optimal solution. Furthermore, the rate of convergence is quadratic when we choose the starting point x_0 *sufficiently* close to the solution.

However, under the assumption of directed water flow, our problem uses the new variant of Hazen-Williams that contains a non-differentiable function

$$F(Q) = K|Q|^n, \tag{4.5}$$

which would make the use of Newton-Raphson impossible. Nevertheless, the method has been used for some time in practice.

In [2], we see that subgradient methods are the correct choice for solving non-smooth optimisation problems. However, we will not consider those as there is a better method we can use through smoothing, which improves the theoretical complexity whilst maintaining the accuracy in results.

At a high overview, the plan can be listed as such:

1. Find a μ-smooth approximation of the non-differentiable function by proving that it is (α, β, K)-smoothable. We refer to section 2.4.2 for a formal definition.
2. Prove that the method used to solve the optimisation problem is *fast* enough. We will consider the Newton-Raphson and the two stages that occur. When the current iterate is far away from the solution and the current iterate is close enough to the solution. We refer to section 2.4.4 to see an example.
3. Finally, prove that there is an optimal μ parameter for our smoothing function such that after $O(\log_2(\log_2(1/\epsilon)))$ iterations, the current solution lies within ϵ from the optimal solution. The theorem in section 2.4.5 gives an insight on how we might achieve this.

4.2 Smooth Approximation

We will start this section by considering a number of candidates for the smoothing function. There are three common ways in we can achieve this.

1. The square root function defined as:

$$f_\mu(x) = \sqrt{x^2 + \mu^2} - \mu. \tag{4.6}$$

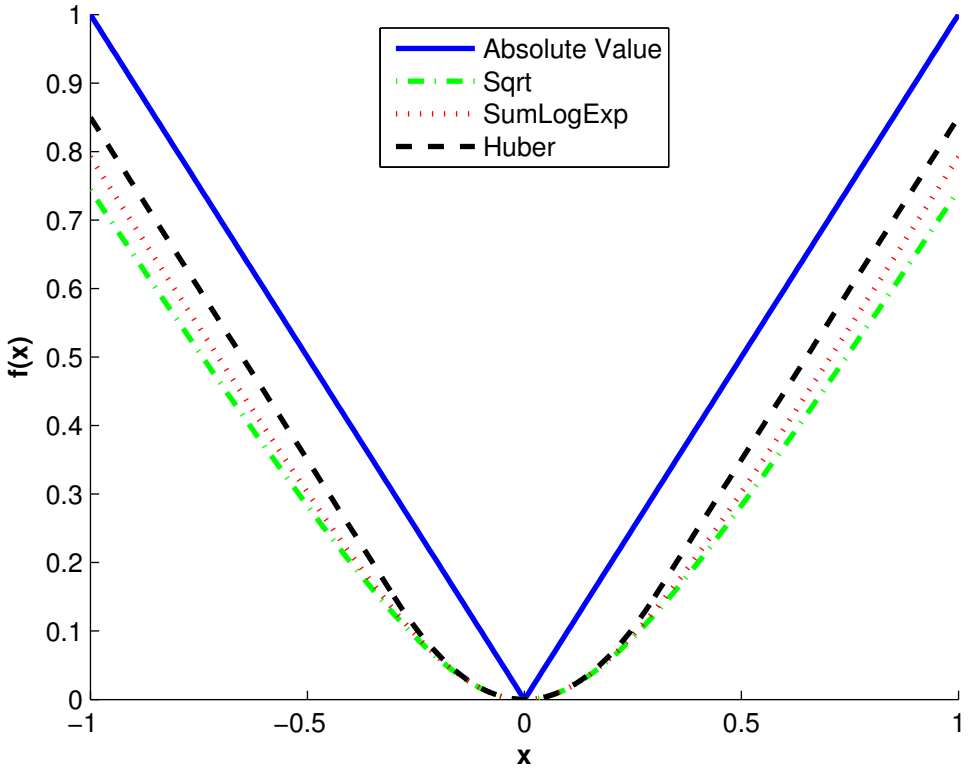
2. The logarithm-exponential function defined as:

$$f_\mu(x) = \mu \log\left(\frac{e^{\frac{x}{\mu}} + e^{\frac{-x}{\mu}}}{2}\right). \tag{4.7}$$

3. The Huber function defined as:

$$f_\mu(x) = \begin{cases} \frac{x^2}{2\mu}, & \text{if } |x| \leq \mu \\ |x| - \frac{\mu}{2}, & \text{else} \end{cases}. \tag{4.8}$$

Figure 4.1: A plot of the absolute value and the smooth approximations listed in section 4.2 for $\mu = 0.3$. We see that Huber function best approximates the absolute values.



As proved in [2] and as it can be seen in Figure 4.1, the Huber function seems to be the best approximation of the absolute value so from now on, we will refer to it when conducting the proofs.

Moreover, as we decrease the value of μ, the smooth approximations will be closer to the absolute value function and if we let $\mu \to 0$, we will get precisely the absolute value.

4.2.1 Proof of smoothable function

We will now prove that the absolute value is a (α, β, K)-smoothable function. As per the definition in section 2.4.2, suppose $f_\mu(x)$ is the Huber function that is a continuously differentiable convex function. We will start by showing part $i.$ of the definition of smoothable functions given in section 2.4.2:

Suppose $x \geq 0$.
Now suppose that $|x| \leq \mu$. Then, we will show:

$$x - \beta_1 \mu \leq \frac{x^2}{2\mu} \leq x + \beta_2 \mu. \tag{4.9}$$

Taking the first inequality, we get:

$$x - \beta_1\mu \leq \frac{x^2}{2\mu} \tag{4.10}$$

$$2\mu x - 2\beta_1\mu^2 \leq x^2 \text{ , as } \mu > 0. \tag{4.11}$$

Let $\beta_1 = 1$ to get:

$$2\mu x - 2\mu^2 \leq x^2, \tag{4.12}$$

which can be written, by going backwards to:

$$x - \mu \leq \frac{x^2}{2\mu}, \tag{4.13}$$

which is clearly true since the following inequalities are true:

$$x - \mu \leq 0 \leq \frac{x^2}{2\mu}. \tag{4.14}$$

Taking the second inequality, we get:

$$\frac{x^2}{2\mu} \leq x + \beta_2\mu \tag{4.15}$$

$$x^2 \leq 2\mu x + 2\beta_2\mu^2 \tag{4.16}$$

$$\mu x \leq 2\mu^2 + 2\beta_2\mu^2 \tag{4.17}$$

$$x \leq 2\mu + 2\beta_2\mu \tag{4.18}$$

$$x \leq 2\mu(1 + \beta_2). \tag{4.19}$$

Let $\beta_2 = -\frac{1}{2}$ and we are done with this case.

Suppose now that $|x| > \mu$. Then, we will show:

$$x - \beta_1\mu \leq |x| - \frac{\mu}{2} \leq x + \beta_2\mu. \tag{4.20}$$

Taking, again, the first inequality, we have:

$$x - \mu \leq x - \frac{\mu}{2} \text{ , since } x \geq 0 \text{ and } \beta_1 = 1, \tag{4.21}$$

which clearly holds. Taking the second inequality yields:

$$x - \frac{\mu}{2} \leq x - \frac{\mu}{2} \text{ , since } \beta_2 = -\frac{1}{2}, \tag{4.22}$$

which holds trivially.

Now suppose $x < 0$. As both the absolute value and Huber functions are even, we conclude the same results.

We are done with part i. of the (2.4.2) definition and now we will look at part ii. Suppose we have the Huber function defined as:

$$f_\mu(x) = \begin{cases} \frac{x^2}{2\mu}, \text{ if } |x| \leq \mu \\ x - \frac{\mu}{2}, \text{ if } x > \mu \\ -x - \frac{\mu}{2}, \text{ else} \end{cases} . \tag{4.23}$$

We will need to show that there are some $K \geq 0$ and $\alpha > 0$ such that the following holds:

$$|f_\mu^{'}(x) - f_\mu^{'}(y)| \leq \left(K + \frac{\alpha}{\mu}\right)|x - y| \tag{4.24}$$

and where

$$f_\mu^{'}(x) = \begin{cases} \frac{x}{\mu}, \text{ if } |x| \leq \mu \\ 1 - \frac{\mu}{2}, \text{ if } x > \mu \\ -1 - \frac{\mu}{2}, \text{ else} \end{cases} . \tag{4.25}$$

Suppose $|x| \leq \mu$. Then equation (4.24) becomes:

$$\left|\frac{x}{\mu} - \frac{y}{\mu}\right| \leq \left(K + \frac{\alpha}{\mu}\right)|x - y| \tag{4.26}$$

$$\frac{1}{\mu} \leq \left(K + \frac{\alpha}{\mu}\right), \tag{4.27}$$

which is satisfied if we take, say, $K = 0$ and $\alpha = 1$.

Now suppose $x > \mu$. We get:

$$\left|1 - \frac{1}{\mu} - 1 + \frac{1}{\mu}\right| \leq \left(K + \frac{\alpha}{\mu}\right)|x - y| \tag{4.28}$$

$$0 \leq \left(K + \frac{\alpha}{\mu}\right), \tag{4.29}$$

which holds for $K = 0$, $\alpha = 1$ as $\mu > 0$.

In the last case we have that $x < -\mu$.

$$\left|-1 - \frac{1}{\mu} + 1 + \frac{1}{\mu}\right| \leq \left(K + \frac{\alpha}{\mu}\right)|x - y|, \tag{4.30}$$

which again reduces to the second case.

The proof is now complete and we can say that the absolute value is a $(1, 0.5, 0)$-smoothable function with the Huber function as the μ-smooth approximation.

4.2.2 Proof of smoothable function II

Another important proof that we need to perform is that on:

$$f(x) = c_2 \frac{|x|^{n+1}}{n+1} \tag{4.31}$$

for which we consider the following smooth function:

$$f_\mu(x) = \begin{cases} c_2 \frac{\left(\frac{x^2}{2\mu}\right)^{n+1}}{n+1} & \text{, if } |x| \leq \mu \\ c_2 \frac{\left(|x| - \frac{\mu}{2}\right)^{n+1}}{n+1} & \text{, else} \end{cases} . \tag{4.32}$$

Suppose $|x| \leq \mu$. We require to find $\beta_1 + \beta_2 = \beta > 0$ in:

$$f(x) - \beta_1 \mu \leq f_\mu(x) \leq f(x) + \beta_2 \mu. \tag{4.33}$$

Starting with the right hand side inequality, we have:

$$c_2 \frac{\left(\frac{x^2}{2\mu}\right)^{n+1}}{n+1} \leq c_2 \frac{|x|^{n+1}}{n+1} + \beta_2 \mu. \tag{4.34}$$

Suppose $\beta_2 = 0$. We get:

$$\left(\frac{x^2}{2\mu}\right)^{n+1} \leq |x|^{n+1}, \tag{4.35}$$

which holds for $x = 0$. Suppose $x \neq 0$ and since both sides will evaluate to a positive value, we can get rid of the absolute value and simply the expression to:

$$x^{n+1} \leq (2\mu)^{n+1}, \tag{4.36}$$

which holds as $x \leq \mu$.

For the second inequality we have:

$$c_2 \frac{|x|^{n+1}}{n+1} - \beta_1 \mu \leq c_2 \frac{\left(\frac{x^2}{2\mu}\right)^{n+1}}{n+1} \tag{4.37}$$

$$|x|^{n+1} - \frac{(n+1)\beta_1 \mu}{c_2} \leq \left(\frac{x^2}{2\mu}\right)^{n+1} . \tag{4.38}$$

When $x = 0$, the inequality is trivially satisfied. Suppose $x \neq 0$. Each of the terms is positive, so we can write the following:

$$1 - \frac{(n+1)\beta_1 \mu}{c_2 x^{n+1}} \leq \left(\frac{x}{2\mu}\right)^{n+1} . \tag{4.39}$$

It will now suffice to choose a β_1 such that the left hand side of the inequality will be at most 0. Thus:

$$(n+1)\beta_1\mu \geq c_2 x^{n+1} \tag{4.40}$$

$$\beta_1 \geq \frac{c_2 x^{n+1}}{(n+1)\mu}. \tag{4.41}$$

At this stage, we may rely on the fact that even though $\mu > 0$, the values that μ can take are very close to 0. Since $x < \mu < 1$, we may take an upper bound for β_1 as:

$$\beta_1 = \frac{c_2}{n+1}. \tag{4.42}$$

Thus we have found our $\beta = \frac{c_2}{n+1}$, which is a useful result for later in the proof. Finding α and K will be part of determining the Lipschitz constant, which is done next.

4.3 Fast Optimisation Method

We will now investigate the convergence properties of Newton-Raphson and deduce a bound of the type

$$D(x_k) - D^* \leq \lambda,$$

where D is the objective function of our optimisation problem and λ is a real number that might depend on the number of iterations k.

We proceed by looking at the convergence analysis and see the two cases that arise when using Newton-Raphson.

4.3.1 Newton convergence analysis

Next, we would like to establish that the optimisation method we use in order to solve the hydraulic equations is fast as defined in section 2.4.4. However, it is not always the case that this is true and so we would define a *fast* notion of our own so that we can establish a bound on the error in solution that we get after a certain number of iterations.

The following statement can be shown about the Newton-Raphson algorithm and a proof of this is given in [17], pages 488–491. What we find by looking at the convergence analysis is that there are numbers η and γ with $0 < \eta \leq \frac{m^2}{L}$ and $\gamma > 0$ such that:

1. If $|f'(x_k)| \geq \eta$, then

$$f(x_{k+1}) - f(x_k) \leq -\gamma. \tag{4.43}$$

2. If $|f'(x_k)| < \eta$, then

$$\frac{L}{2m^2}|f(x_{k+1})| \leq \left(\frac{L}{2m^2}|f(x_k)|\right)^2, \tag{4.44}$$

where m comes from our strong convex assumption that f is strong convex with constant m. That is to say $\nabla^2 f(x) \succeq mI$. Also, L is the Lipschitz constant of the Hessian of f, which we assume to be Lipschitz continuous:

$$|f^{''}(x) - f^{''}(y)| \leq L|x - y|. \tag{4.45}$$

We will first analyse the implications of equation (4.32). Suppose that the premise is through. Thus,

$$|f^{'}(x_k)| < \eta \tag{4.46}$$

and as $\eta \leq \frac{m^2}{L}$, we obtain the following from equation (4.32):

$$|f(x_{k+1})| \leq \frac{L}{2m^2}|f(x_k)|^2 \tag{4.47}$$

$$|f(x_{k+1})| < \frac{L}{2m^2}\left(\frac{m^2}{L}\right)^2 \tag{4.48}$$

$$|f(x_{k+1})| < \frac{m^2}{2L}. \tag{4.49}$$

So indeed, there is a number η with $0 < \eta \leq \frac{m^2}{L}$ such that:

$$|f^{'}(x_{k+1})| < \eta \tag{4.50}$$

holds. And so this result holds for any $l \geq k$. Consequently, we can write $\forall l \geq k$:

$$\frac{L}{2m^2}|f(x_{l+1})| \leq \left(\frac{L}{2m^2}|f(x_l)|\right)^2. \tag{4.51}$$

Applying this recursively, we obtain:

$$\frac{L}{2m^2}|f(x_l)| \leq \left(\frac{L}{2m^2}|f(x_k)|\right)^{2^{l-k}} \leq \left(\frac{1}{2}\right)^{2^{l-k}} \tag{4.52}$$

and hence

$$f(x_l) - f(x^*) \leq \frac{1}{2m}|f(x_l)|^2 \leq \frac{2m^3}{L^2}\left(\frac{1}{2}\right)^{2^{l-k+1}}, \tag{4.53}$$

with $f(x^*)$ the optimal solution.

4.3.2 Determining the Lipschitz constant

The objective function we are looking at is of type:

$$F_\mu(x) = f(x) + g_\mu(x), \tag{4.54}$$

where

$$f(x) = c_1 x \tag{4.55}$$

$$g_\mu(x) = \begin{cases} c_2 \frac{\left(\frac{x^2}{2\mu}\right)^{n+1}}{n+1} \text{ , if } |x| \leq \mu \\ c_2 \frac{\left(|x| - \frac{\mu}{2}\right)^{n+1}}{n+1} \text{ , else} \end{cases} , \tag{4.56}$$

with $c1$ and $c2$ constants particular to the network. For simplicity, we will compute the result in terms of them, but, from a hydraulic perspective, they can be thought as:

$$c_1 = K_i \tag{4.57}$$

$$c_2 = A_{10i} H_0 + A_{13i} \eta, \tag{4.58}$$

where K_i is the constant from Hazen-Williams and the rest of the terms are network specific as described in section 2.3.2 and section 5.2.

From Definition 1 in section 2.4.2, we will find the Lipschitz constants of the Hessian of the two functions defined by equations (4.55) and (4.56). First, we compute their second order derivatives:

$$f''(x) = 0 \tag{4.59}$$

$$g''_\mu(x) = \begin{cases} c_2(2n+1)\frac{x^{2n}}{2^n \mu^{n+1}} \text{ , if } |x| \leq \mu \\ c_2 n \left(x - \frac{\mu}{2}\right)^{n-1} \text{ , if } x > \mu \\ c_2 n \left(-x - \frac{\mu}{2}\right)^{n-1} \text{ , else} \end{cases} . \tag{4.60}$$

We first consider f and obtain:

$$|f''(x) - f''(y)| \leq L_f |x - y| \tag{4.61}$$

$$0 \leq L_f |x - y|. \tag{4.62}$$

So we can set $L_f = 0$. Moving on to g_μ, suppose $|x| \leq \mu$ and we find:

$$|g''_\mu(x) - g''_\mu(y)| \leq L_g |x - y| \tag{4.63}$$

$$\frac{c_2(2n+1)}{2^n \mu^{n+1}} \left|x^{2n} - y^{2n}\right| \leq L_g |x - y|. \tag{4.64}$$

Now, we can say that we can bound the function we have so long as the domain of g_μ is a compact set. This is a reasonable assumption since the variable denotes the water flowrate through a pipe, which cannot possibly exceed certain limits. We will thus denote by Ψ the upper bound. Therefore,

$$|x| \leq \Psi \text{ , } \forall x \text{ in the domain.} \tag{4.65}$$

Hence, we can use the following fact:

$$|x^k - y^k| \leq |x - y| k \Psi^{k-1}. \tag{4.66}$$

Combining (4.64) with (4.56) we establish:

$$\frac{c_2(2n+1)}{2n2^n \Psi^{2n-1} \mu^{n+1}} \leq L_g \tag{4.67}$$

and so it will suffice to set:

$$L_g = \frac{c_2}{\mu^{n+1}} \tag{4.68}$$

and obtain the desired Lipschitz constant that will help us find the optimal smoothing parameter μ for which we achieve a lower bound for the rate of convergence of the algorithm. We conclude that:

$$L = L_f + L_g = \frac{c_2}{\mu^{n+1}}. \tag{4.69}$$

We refer to Appendix 10.2 for validation in the other cases when $|x| > \mu$.

4.4 Finding μ and establishing the lower bound

4.4.1 Quadratically Convergent Phase

When we perform this analysis we will refer to the definitions given by equations (4.54), (4.55) and (4.56). Thus, using equation (4.53) and the result in equation (4.69), we may write:

$$F_\mu(x_k) - F_\mu(x^*) \leq \frac{2m^3}{\left(\frac{c_2}{\mu^{n+1}}\right)^2} \left(\frac{1}{2}\right)^{2^{k-p+1}}, \tag{4.70}$$

which holds for any $k \geq p$, with p the constant that denotes the iteration after which we are *close* enough to the solution according to Newton-Raphson and we fall into the quadratically convergent phase.

Now, since we know g_μ is a μ-smooth approximation of g, there are some $\beta_1 + \beta_2 = \beta > 0$ such that:

$$F(x) - \beta_1 \mu \leq F_\mu(x) \leq F(x) + \beta_2 \mu, \tag{4.71}$$

where $F(x) = f(x) + g(x)$, with g the non-smooth function. Since this holds for any x, it will hold in particular for:

$$F(x^*) \geq F_\mu(x^*) - \beta_2 \mu \tag{4.72}$$

$$F(x_k) \leq F_\mu(x_k) + \beta_1 \mu \tag{4.73}$$

and combining with (4.71), we will get:

$$F(x_k) - F(x^*) \leq F_\mu(x_k) - F_\mu(x^*) + (\beta_1 + \beta_2)\mu \leq \frac{2m^3}{\left(\frac{c_2}{\mu^{n+1}}\right)^2}\left(\frac{1}{2}\right)^{2^{k-p+1}} + \beta\mu \tag{4.74}$$

$$F(x_k) - F(x^*) \leq \frac{2m^3\mu^{2n+2}}{c_2^2}\left(\frac{1}{2}\right)^{2^{k-p+1}} + \beta\mu. \tag{4.75}$$

Minimization of the right-hand side with respect to $\mu > 0$ will be achieved as $\mu \to 0$. For convenience, we will choose the value of μ as:

$$\mu = \frac{1}{\sqrt{\beta}}\left(\frac{2m\sqrt{m}}{c_2}\left(\frac{1}{2}\right)^{2^{k-p}}\right)^{\frac{1}{2n+2}}. \tag{4.76}$$

We do not need to decrease μ further, as this value is enough to establish a theoretical bound on the rate of convergence.

Plugging the value of μ obtained in equation (4.76) into equation (4.75), we get:

$$F(x_k) - F(x^*) \leq \frac{4m\sqrt{m\beta}}{c_2}\left(\frac{1}{2}\right)^{2^{k-p}}. \tag{4.77}$$

Thus, given $\epsilon > 0$, to obtain an ϵ-optimal solution satisfying

$$F(x_k) - F(x^*) \leq \epsilon,$$

we need to find the values of k for which the following is satisfied:

$$\frac{4m\sqrt{m\beta}}{c_2}\left(\frac{1}{2}\right)^{2^{k-p}} \leq \epsilon. \tag{4.78}$$

Carrying on the computation, we see

$$\left(\frac{1}{2}\right)^{2^{k-p}} \leq \frac{\epsilon c_2}{4m\sqrt{m\beta}} \tag{4.79}$$

$$2^{k-p} \geq \log_2\left(\frac{4m\sqrt{m\beta}}{\epsilon c_2}\right) \tag{4.80}$$

$$k \geq \log_2\left(\log_2\left(\frac{4m\sqrt{m\beta}}{\epsilon c_2}\right)\right) + p. \tag{4.81}$$

Plugging the lower bound value of k in equation (4.76), we obtain the following value of μ which established the link between μ and ϵ:

$$\mu = \frac{1}{\sqrt{\beta}}\left(\frac{\epsilon}{2}\right)^{\frac{1}{2n+2}}. \tag{4.82}$$

4.4.2 Newton Damped Phase

We now refer to equation (4.43) where we find ourselves further away from the solution and require to find a bound on the number of iterations required. Simply, since f decreases by at least γ each iteration, the total number of iterations cannot exceed

$$\frac{f(x_0) - f(x^*)}{\gamma}. \tag{4.83}$$

We refer to [17] (pp 488–491) for a complete proof.

4.5 The New Result

Let $\{x_k\}$ be the sequence generated by the Newton-Raphson iterative algorithm when applied to a problem of the type:

$$\min F_\mu(x) = f(x) + g_\mu(x). \tag{4.84}$$

Suppose that the smoothing parameter is chosen as:

$$\mu = \frac{1}{\sqrt{\beta}} \left(\frac{\epsilon}{2}\right)^{\frac{1}{2n+2}}. \tag{4.85}$$

Then for

$$k \geq \log_2 \left(\log_2 \left(\frac{4m\sqrt{m\beta}}{\epsilon c_2} \right) \right) + p. \tag{4.86}$$

it holds that $F(x_k) - F(x^*) \leq \epsilon$.

This result follows from sections 4.2 – 4.4. We have thus established a lower bound on the number of iterations required for convergence of $O(\log_2(\log_2(1/\epsilon)))$ in the quadratically convergent phase, which outperforms both the usual subgradient methods' $O(1/\epsilon^2)$ and the results in [2] of $O(1/\epsilon)$.

5 Other Ideas

The aim of this chapter is to briefly present some of the other main directions that have been considered for the progress of the project.

5.1 Demand Driven or Pressure Driven

The Todini and Pilati mathematical model we are solving for our hydraulic analysis contains a set of nonlinear equations in which the customer demand q enters as a fixed term and we require to solve for optimal values of piezometric head H and water flowrate Q. Equations are presented as:

$$A_{12}H + F(Q) = -A_{10}H_0 \tag{5.1}$$

$$A_{21}Q = q. \tag{5.2}$$

An issue with this model is that it makes certain assumptions on how q will look, which, albeit roughly correct, do not always adhere to the real values. Typical values of q will peak around morning time and evening time, be lower during the day and at the lowest during the night. Moreover, this assumption is the same for any day of the week, regardless of the week.

An alternative view of this problem may be seen as removing the assumptions made for the customer demand and focus on the amount of pressure that must be met at each node of the junction for a certain time frame.

It seems more natural for one to estimate the physical attribute of pressure, rather the *uncertain* customer demand as we can have control over pressure, but not as much on the customer demand. Nonetheless, this proposal does not necessarily simplify or improve the task, as estimating pressure might prove to be an even more difficult problem since it does not always have a similar, monotonic behaviour as the customer demand.

5.2 Valve modelling

Valves represent an important part of a hydraulic network. Fundamentally, they are similar in structure to the usual pipes and together with them form the links of the network. However, due to ongoing enhancements in water control, models that have been released decades ago fall short in coping with all the new improvements added to a water network distribution.

We will illustrate some of the newest technologies used in the design and construction of a hydraulic network:

- Loggers

  These devices are installed at specific points of the network and their purpose is to determine the pressure, temperature, flow and similar useful characteristics. Thus, they provide important monitoring of the network and are useful for leakage detection, pressure control and, ultimately, for our optimisation purposes.

  Newest loggers are very robust and reliable and have many interesting features that makes them very popular in practice. They offer remote communication, relatively high memory storage, efficient battery consumption.

- Pump control

  Pumps are elements that occur in the hydraulic network and are similar to pipes or links in the network, but add more functionality as they are able to make control decisions such as to influence the water flowrate or head loss in order to reduce water leakage and waste.

- Pressure Reducing Valves (PRVs)

  As per their name, this class of valves improve upon normal network elements by having an extra choice of reducing the pressure when pressure is in excess, thus reducing waste as well as leakage or bursts through the network. Another important advantage is the remote control of pressure management, which requires fewer site visits to manage the pressure at certain points in the network.

  We will now carry on discussing the implications of modelling such new elements and we will focus particularly on PRVs as the Loggers are somewhat independent of the hydraulic analysis and pumps are slightly older than PRVs and perhaps don't present as much interest.

We can easily incorporate PRVs in our model if we treat this special type of valves as pipes. Simply extend the connectivity matrix $A12$ presented in equations (5.1) and (5.2), as well as flowrate vector Q and obtain results for optimal flowarates at the valves as well.

This method would clearly not be very good since we would like to capture the beneficial contribution of the PRVs as well. This is where the problem arises and for this reason we would like to extend the model described by Todini and Pilati by equations (5.1) and (5.2) as there is no such model currently.

In order to address this issue, a new problem formulation has been devised as:

$$F(H) = \min \sum_{j=1}^{nn} H_j, \tag{5.3}$$

where the objective function should be read as a minimization of pressure through the network consisting of nn nodes.

The equality constraints are defined as:

$$A_{12}H + F(Q) + A_{13}\eta + A_{10}H_0 = 0 \tag{5.4}$$

$$A_{21}Q - q = 0, \tag{5.5}$$

where the only difference between the original formulation is the addition of the $A_{13}\eta$ term. The A_{13} matrix denotes the connectivity of the valves in the network and η is the unknown vector representing the valve settings and is similar to Q in that it will solve for optimal flowrate and will influence the piezometric head H as well. Also, they denote the equations of conservation of energy and mass in the water network. The rest of the elements are as described in section 2.3.2.

Carrying on, we also have the following inequality constraints:

$$-Q \leq 0 \tag{5.6}$$

$$-\eta \leq 0 \tag{5.7}$$

$$H_{min} - H_j \leq 0 \ , \forall j \in nn, \tag{5.8}$$

where these constraints assure that the water flow, as well as the valve settings are always positive or that they preserve a certain direction. The last constraint is there to set a lower bound on the pressure at any given point within the network.

In order to solve the problem described by equations (5.3)–(5.8), a few methods were used including a gradient method and two direct methods. However, they were proven to be suboptimal mainly because they were sensitive to initial conditions, which means they were slow, not reliable and in some cases could not found an optimal solution.

Bearing in mind that the difficulty of the hydraulic equations lies in the fact that the number of nonlinear equations is high, sequential convex programming has been used in order to split the optimisation problem into two smaller ones. We proceed as such:

Subproblem A will take fixed η and solve the problem that consists of only equations (5.3)–(5.5). Initially, we set $\eta = 0$, which makes the problem identical to the initial Todini problem as described in section 2.3.2. We then have solutions, as before, for piezometric head and water flowrate. Suppose they are H_s and Q_s.

Subproblem B will be a linear program that solves the whole problem by linearising equation (5.4) using the already computed values of H_s and Q_s. We will then obtain optimal valve settings η as required. The process will continue until convergence is reached, which is checked either through the satisfiability of the KKT conditions or, simpler, checking that the values of H and Q obtained by the two subproblems are within some ϵ.

Linearisation of equation (5.4) is done as:

$$F(Q) = AQ + b, \tag{5.9}$$

with the following boundary conditions:

$$F(0) = 0 \tag{5.10}$$

$$F(Q) = KQ^{1.85}, \tag{5.11}$$

which cleary yields the following values for our linear function:

$$A = KQ^{0.85} \tag{5.12}$$

$$b = 0. \tag{5.13}$$

Now, one may think that this is not an improvement since Q is still nonlinear, but actually the value for A in (5.12) is obtained by the already computed Q_s from subproblem A, which makes it a simple constant when solving the linear program described by part B.

5.3 Robust Optimisation

The aim of robust optimisation is to formulate an optimisation problem in such a way as to take into account the uncertainty in your model parameters and output a solution that is robust enough in order to be optimal for the whole range of optimisation problems that take into account the uncertainty.

As mentioned in section 5.1, customer demand does not change from one day to another too much and only small variations can be observed through time. The question to be asked in this scenario is whether we can formulate a robust version of the problem described by equations (5.1) and (5.2) such that these variations in customer demand can be dealt with.

Thus, we will achieve a solution that is general enough for any day in a given time interval. At the moment, we are solving the hydraulic equations at discrete time intervals throughout the day. Instead, the new methodology would solve the problem for different times of the day only once and apply this result whenever customer demand q is within some ϵ from the original q.

Nonetheless, before we consider applying such an approach, we need to check what kind of variations in optimal solutions, a small change in customer demand q will produce. For this, we slightly change the model as:

$$A_{12}H + F(Q) = -A_{10}H_0 \tag{5.14}$$

$$A_{21}Q = (1 + \xi)q, \tag{5.15}$$

where ξ can be a uniform, normal, lognormal random variable with mean 0 and standard deviation ϵ. For our purposes, however, we may let ξ be as:

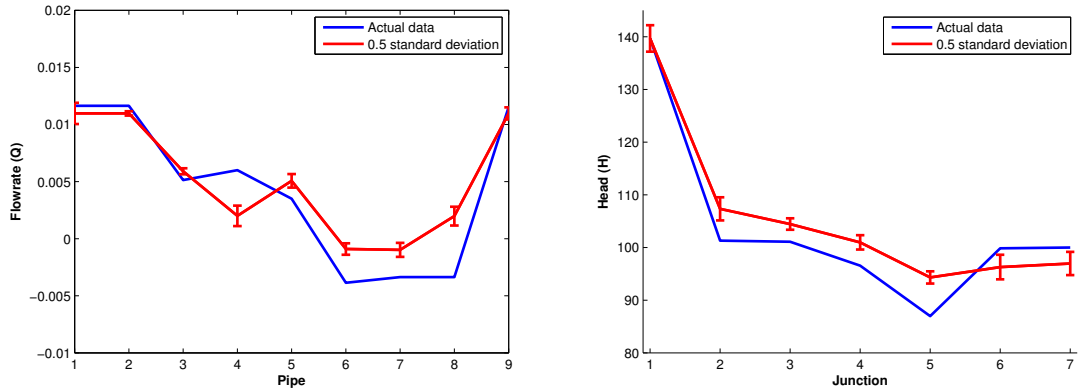
$$\xi \sim \mathcal{N}(0, \epsilon), \tag{5.16}$$

where ϵ is the standard deviation that we choose. Please note that an uncertainty of x means an uncertainty of $x\%$ in the input.

Running different simulations showed that the results at different uncertainty levels tend to be quite similar, but there is a point after which they diverge significantly.

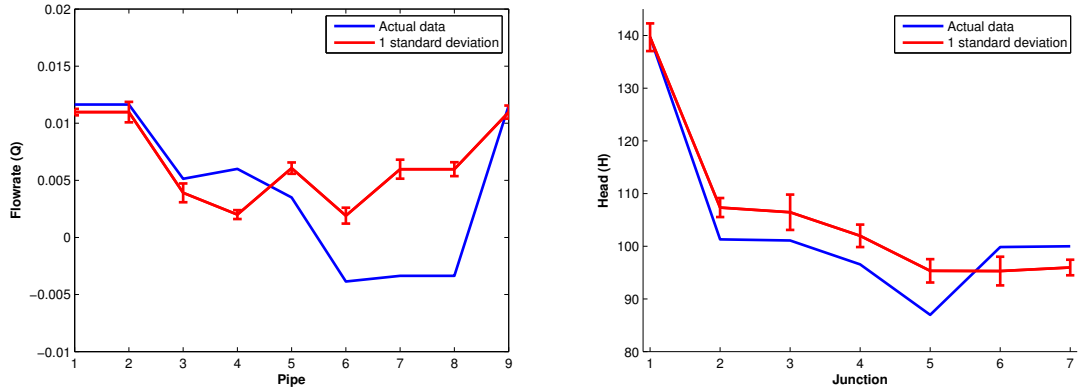
Figures 5.1 to 5.4 portray the results of the comparison performed. We have considered a small network and the results in blue are the ones of the original problem. The red lines denote solutions for the problem with the added ϵ error. Further on, the left hand side shows the results for the piezometric head H, while on the right hand side you may see the results for water flowrate Q.

Figure 5.1: Q (left) and H solutions for $\epsilon = .05$



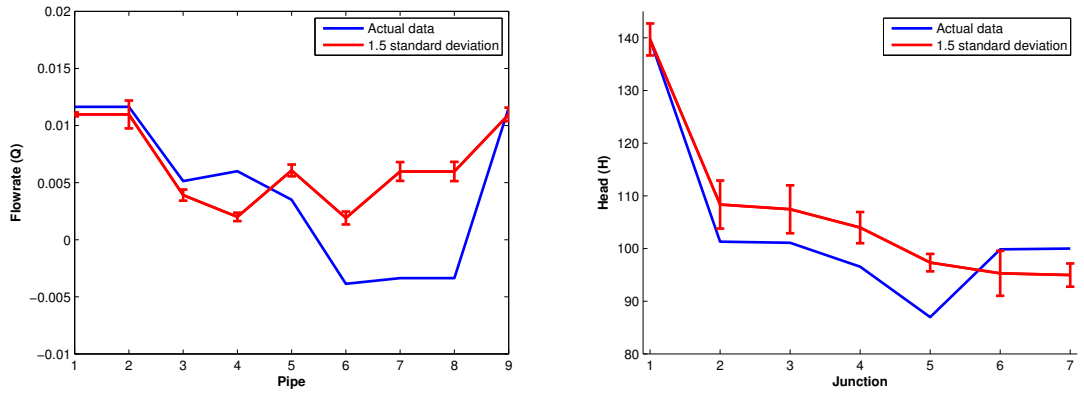
At the first stage, we have chosen an error of $\epsilon = .05$ and you can see that results follow roughly the same pattern, despite having some rather large discrepancy especially at junction 5 of Figure 5.1 of the piezometric head.

Figure 5.2: Q (left) and H solutions for $\epsilon = 1$



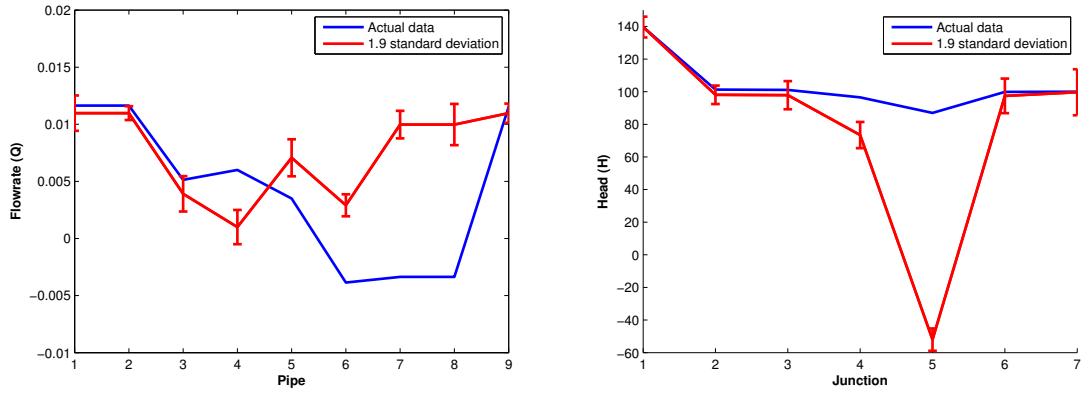
Carrying on with the analysis, we see a similar trend of results even at $\epsilon = 1$ uncertainty. Again, results are similar, but there are larger differences around, for instance, junction 5 of Figure 5.2.

Figure 5.3: Q (left) and H solutions for $\epsilon = 1.5$



We again increase the error at $\epsilon = 1.5$, but we see very little difference from the previous two steps. Nonetheless, it is important to have a systematic approach in order to be sure of any change in results that may occur.

Figure 5.4: Q (left) and H solutions for $\epsilon = 1.9$



An interesting observation is to be made when looking at Figure 5.4 as we see that the solution becomes infeasible from a physical point of view when H drops very low, in the negative area. The model we use does not account for this case as there is no constraint of the type $H > 0$ and thus, when a junction simply cannot handle the amount of pressure that goes that way, it outputs a wrong result.

6 Implementation

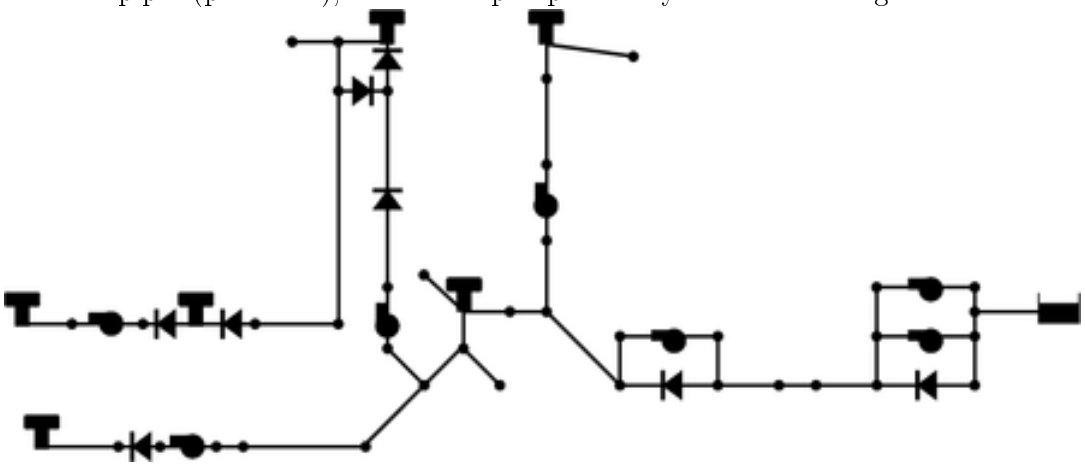
At the beginning of the project we had two main objectives in mind when developing the hydraulic solver. While performance is a key factor in any application, we wanted to build a software package that can be used by researchers in order to easily implement and validate any new ideas and theories in the field of hydraulic analysis.

Hence, we would first like to make the hydraulic software as fast as possible by looking both into the details of the implementation (see Chapter 3), but also considering a higher-level view by investigating and adopting new frameworks that increase optimality (see Chapter 4).

Furthermore, we want to keep in mind the principles of good software design and offer researchers a robust and reliable framework, which offers high modularity, loose coupling between different parts of the application and an ease of extension.

6.1 The Problem

Figure 6.1: Richmond Water Network. There are three types of nodes: junctions (simple dots in the figure), reservoirs (rightmost symbol) and tanks (leftmost symbol). There are three types of links: pipes (plain line), valves and pumps with symbols like in Figure 1.1.



We will focus our attention on the different elements which appear in a water distribution network. Figure 6.1 shows the Richmond water network with various nodes and links, which will be described next.

6.1.1 Epanet Files

The standard input to the problem comes in the form of Epanet input files that typically have a *.inp* extension. It consists of simple rules to describe all the characteristics of a water network, which also makes parsing its contents relatively easy. We will now study and discuss the meaning of the elements that appear in the Epanet input files. We will start looking at the most relevant of them, which have a direct implication in our problem:

- *Junctions*
  The junctions of a water network are the nodes of the network servicing the clients as per their demand. The information that we get for each junction is

  i. ID in order to uniquely identify a junction.

  ii. *Elevation*, which denotes the height of a particular junction above a certain, fixed reference point. From a strictly mathematical viewpoint, this quantity is important if we want to calculate the pressure at a specific node from the piezometric head or vice-versa.

  iii. *Demand*, which denotes the customer demand for water at the current node of the network. This is an important value as we would need to supply a certain pressure and water flow as to meet the given demand under the minimization of water loss that flows through the pipes.

  iv. *Pattern* is, in fact, a pattern ID which helps us identify a sequence of 96 numbers that would form the multipliers of the given demand. The reasoning is as follows: in order to get an accurate description of the demand throughout the day, a normal day is split into 96 parts (15 minutes) so that we would be able to read the demand at 15 minute intervals. We could then run the hydraulic analysis for any point of the day.

- *Reservoirs*
  The reservoirs of a water network are essentially the starting nodes of the network that feed the water to the pipes and junctions. In our model, they will represent the nodes for which the hydraulic head is known. They contain the following information:

  i. ID in order to uniquely identify a reservoir.

  ii. *Head*, which specifies the liquid pressure above a geodetic datum - a fixed point of reference - that would form our known head information.

  iii. *Pattern* is, as it was the case with junctions, a pattern ID which helps us identify a sequence of 96 numbers that would form the multipliers of the given demand. The reasoning is as follows: in order to get an accurate description of the pressure throughout the day, a normal day is split into 96 parts (15 minutes) so that we would be able to read the pressure at 15 minute intervals. We could then run the hydraulic analysis for any point of the day.

- *Tanks*

  i. ID in order to uniquely identify a tank

  ii. *Elevation*, which denotes the height of a particular tank above a reference geoid. This quantity is important when you want to retrieve the pressure from the piezometric head or vice-versa.

iii. *Initial level* this gives information about the initial pressure we have in the tank. Note that tanks behave similar to reservoirs, only that the pressure might vary in time. In the case of reservoirs, the *Head* remains fixed throughout the simulation.

iv. *Minimum level* denotes the minimum amount of pressure that the tank can have at any given point in time. This information is to be expected since the pressure of the tanks vary in time and so we would not want to exceed certain boundaries as otherwise it might prove difficult to achieve certain demand objectives.

v. *Maximum level* denotes the maximum amount of pressure that the tank can have at any given point in time. Similar to minimum, it is important to know when we exceed a certain boundary.

vi. *Diameter* denotes the diameter of the tank. Our model does not include this information.

vii. $MinimumVolume$ our model does not include this information.

viii. $Volume$ our model does not include this information.

We can observe that modelling the tanks is not complete. This is acceptable since the test cases we have and, in fact, the vast majority of cases we encounter do not include the information we didn't model.

- *Pipes*
Pipes represent one of the three types of links that we have in the network. In the water network, they make the link between the node-type elements that are the ones we have already seen: Junctions, Reservoirs and Tanks. In the input file, they are followed by a number of different characteristics:

i. ID in order to uniquely identify a pipe.

ii. $Node$ 1, which denotes one of the two nodes of this certain link. Each link has two nodes and in this case, the Epanet input file denotes the node where flow enters the pipe by 1.

iii. $Node$ 2, which denotes one of the two nodes of this certain link. This node corresponds to the node of the link through which water flow exits.

iv. *Length* denotes the length of the pipe in the water network.

v. *Diameter* denotes the diameter of the pipe in the water network.

vi. *Roughness* denotes the roughness of the pipe in the water network. In this case, by roughness we refer to the hydraulic roughness, which is the measure of the amount of frictional resistance that is faced by the water whilst passing through the pipe. This represents an important quantity as it helps us determine the amount of head loss in each pipe, which is of paramount importance when computing the optimal values for water pressure and flow.

vii. *Minor Loss* - the minor loss coefficient represent the losses in water velocity and, hence, pressure caused by fittings, bends, valves found in the network. They are named minor, as they are considered negligible in comparison to the friction losses, which are considered major. The head losses are computed from the velocity of the water and a certain constant K that is correlated to the material of the pipe. Moreover

the velocity v of the water is computed as

$$v = \frac{Q}{A},$$

where Q is the water flow rate through the pipe and A is the area of the pipe. The minor loss can then be computed as:

$$h = K\frac{v^2}{2g},$$

where, as above, K is a material-dependent constant, v the velocity and g the acceleration of gravity. Nonetheless, the minor loss for the pipes

viii. *Status* denotes the status of the current pipe. It will usually be *Open* in which case water flows normally through the pipe, but it might be *Closed* in which case we should ignore it.

- *Valves*
  The second type of links we see in the water network are valves. Contrary to the pipes, valves have the ability to regulate, direct, control the flow of the water, which makes their modelling slightly more complex.

  For now, we will take a look at the characteristics of the valves as they occur in the input file:

  i. *ID* uniquely identifies the valve in the water network.

  ii. *Node* 1 denotes, as it is the case with the pipes, one of the two nodes of this certain link. Each link has two nodes and in this case, the Epanet input file denotes the node where flow enters the valve by 1.

  iii. *Node* 2, which denotes one of the two nodes of this certain link. This node corresponds to the node of the link through which water flow exits.

  iv. *Diameter* denotes the diameter of the valve in the water network.

  v. *Type* - unlike pipes, valves come in different types.

  vi. *Setting* this information regulates the amount of pressure that is to be saved starting from this point and going in the direction of water flow that comes at this point. This information in not modelled.

  vii. *Minor Loss* this parameter is not modelled although it never occurs in our test cases.

  Perhaps an important factor we do not model is the full functionality of valves. This is still acceptable since we can run any simulation even without the additional features. Please see section 5.2 for additional details on this matter.

The above mentioned elements are the most important ones and they represent the network model together with its characteristics. Another key element is the *Patterns* section, which is a list of *key* - *value* pairs that represent the demand multipliers for particular nodes of the network. The way it works is that for each node of the network, you may have a *key* that points to a list of 96 multipliers that give you the demand at every 15 minutes of a day. Thus, you are able to solve the optimisation problem at various time intervals throughout the day.

6.1.2 Design overview

We will now take a brief look at the design of the hydraulic solver we have implemented. At a high-level view, one can view the application as composed of three big components. These are:

1. The Parser
2. The Hydraulic Network Model
3. The Solver

This partition is key to the design as it offers low coupling between the components as their communication is managed by a main process through a very easy to use API.

In particular, it offers the user a robust framework that can easily be extended to make use of a new algorithm or technique for solving the non-linear equations.

```
Solver * solver = SolverFactory::createInstance(args, solverType);
```

where *solverType*, the second argument, denotes the type of *Solver* to be used. Of course, an implementation will be required, but that is very easy to accomplish since we can inherit from the *Solver* interface and only implement the *solve* method.

```
class MyNewSolver : public Solver
{
  public:
    void solve()
    {
      // add implementation here
    };
};
```

Additionally, one may want to alter the pipes or valves or any other elements' configuration during the simulation. Of course, this can be achieved through the various access methods of the solver. Of particular interest might be the head loss function that is used.

```
solver->headLossFunction(new HazenWilliamsHeadLossFunction());
```

where the argument of the method takes an object of a type that extends *HeadLossFunction*. Similar to the solver, an implementation of any head loss function can easily be provided if one does not already exist.

We will now start talking in a bit more detail about the different main components that have been mentioned.

6.2 The Parser

The parser is the first main component of the application as it reads the input file and is able to prepare the application for building the network model. It is also the simplest part since the

format of the *Epanet* input files, which is used is very clear and easy to work with.

The structure of the input files is very well described by section 6.1.1 whereby each of the elements presented is preceded by a header of type [$JUNCTIONS$], [$RESERVOIRS$], etc, which is then followed by a table with all the relevant numbers.

We are using Boost's Tokenizer in order to parse the data and maintain a map of the information of the input file. The precondition of the parser is that the format is kept as is, while extending the input file will not affect the program whatsoever.

Finally, the information from the reader will be fed to some *Creator* objects, which will construct the model, fact that will be detailed next.

6.3 The Hydraulic Network Model

At this stage, we have the information network information stored in a map and through the *Creator* objects, we are able to create the data structures that will form the hydraulic network model. The way this works is that each of the *Creator* objects (e.g. *JunctionCreator*, *PipeCreator*) will be passed the relevant part of parsed file and will create the network components (i.e. *Junction*, *Pipe*) in the form of useful data structures.

Once this model is created we can then go on and create the matrices we require for solving the nonlinear system. Depending on the case, we will build matrices $A12$, $A21$, $A10$, $H0$, q, as well as making sure to do any precomputational step if any is required.

Once these procedures have terminated, we will pass control to the main loop again, which will start running the simulation and solve the hydraulic equations.

6.4 The Solver

As mentioned in chapter 3, there are a number of solvers that can be used to solve this particular problem. These solvers have the ability to find optimal solutions for H and Q for different time intervals and for different networks and run several simulations in parallel if desired.

At the end of the simulation, we are able to retrieve useful metrics such as the number of iterations required for the algorithm to converge or the time required by the algorithm to solve the system. Also, we are able to plot the piezometric head at each node and the water flowrate for each link in the network.

6.5 Summary

The entire codebase was developed by the author of the project and it is hosted on a private repository, but will also be made publicly available via github.

At the moment, we have written approximately 5000 lines of code for which we required close to 300 commits. All the dependencies required for the use of the software can be easily found online and installed. They are Boost, LAPACKPP, Eigen and optionally GnuPlot, together with a C++ compiler (preferably gcc version 4.8.2 or above and make).

Figure 6.2: Print screen running a simulation on a small network.

```
bogdan@crm:~/Smart Water Grid/src$ ./solver ../examples/Simple_Example.inp -smooth=HUBER -mu=0.001
======================
Node results
----------------------
ID      Head
======================
0       139.702
1       101.298
2       101.081
3       96.8809
4       87.8459
5       99.875
6       99.9835

======================
Link results
----------------------
ID      Flow
======================
0       0.0118659
1       0.0118659
2       0.00513501
3       0.006
4       0.0037309
5       -0.00386499
6       -0.00313409
7       -0.00313409
8       0.0118659

Converged after 9 iterations.

Simulation time: 0.104 ms!
```

7 | Project Evaluation

We will evaluate our project in terms of both the numerical analysis performed and the theoretical work done on adopting the smooth framework and see just how well they behave and how they improve upon the existing state of the art software.

7.1 Overview

Our benchmark is composed of standard input files, which contain the topological structure of a water distribution network. We will use networks of different sizes ranging from very small (10 nodes and 10 links) to some of the largest that we can have (5000 nodes and 5000 links).

We have run the simulations on a personal laptop with the following specifications:

- Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz x 8
- 15.6 GiB Memory
- OS: Linux Ubuntu 14.04 LTS, 64-bit

We will systematically go over the results of the simulations for the numerical analysis case and the smooth framework case by looking at the most relevant characteristics of the optimisation algorithm used.

Finally, we will discuss about the current standard in industry in Epanet and CWSNet and how our implementation compares against them.

7.2 Numerical Analysis

We will now turn our attention to the numerical analysis we have done and view the results in terms of CPU time required or computational time, the rate of convergence of the algorithm used or the number of iterations required, as well as taking a look at the accuracy that we have and how that affects the performance of our solver.

7.2.1 Computational Time

We will start looking at the results obtained using the very first implementation, which did not explore the sparsity nature of the problem, nor different implementations. As it can be seen in

Table 7.1: A simulation time comparison of the *first* implementations of the hydraulic solver. The times shown are in ms. The tolerance level is $\epsilon = 10^{-6}$.

Solver (Size)	Network 1 (7 x 9)	Network 3 (865 x 950)	Network 4 (2303 x 2369)	Network 5 (4577 x 4648)
LLT Factorization	1.2	$>$ 30 seconds	$>$ 3 minutes	$>$ 6 minutes
LDLT Factorization	1.22	$>$ 30 seconds	$>$ 3 minutes	$>$ 6 minutes
LU Factorization	1.23	$>$ 30 seconds	$>$ 3 minutes	$>$ 6 minutes
Conjugate Gradient	5.7	$>$ 1 minute	$>$ 5 minutes	$>$ 10 minutes
Modified Conj. Grad.	5.9	$>$ 1 minute	$>$ 5 minutes	$>$ 10 minutes
Jacobi	6.5	$>$ 1 minute	$>$ 5 minutes	$>$ 10 minutes
Gauß-Seidel	6.6	$>$ 1 minute	$>$ 5 minutes	$>$ 10 minutes
GMRES	5.8	$>$ 1 minute	$>$ 5 minutes	$>$ 10 minutes
CWSNet	0.128	23.104	147.363	248.437

Table 7.1, the CPU time for small size problems is good, but still an order of magnitude worse as we move from our implementation that requires $O(ms)$ compared to the industry standard CWSNet that solves in $O(0.1ms)$.

This difference increases even further when we increase the number of nodes and links in the network. Thus, even for networks of size 1000 by 1000, we only get the optimal solution in over half a minute, whereas the standard CWSNet still performs in $O(10ms)$. The last two networks portray an even bigger discrepancy, our solver requiring several minutes, whereas CWSNet spends only $O(100ms)$.

This significant difference comes from ignoring the nature of the problem, which, as can be seen in Table 7.2, is of paramount importance to optimally solving the hydraulic equations.

In the latter table, one can view that the results are not just comparable to the standard in industry, but they are even better with special emphasis put on the larger difference where our direct solvers are able to find an optimal solution in $O(10ms)$ as opposed to CWSNet's $O(100ms)$.

This important result comes from exploring the very sparse nature of the problem and from eliminating all the unnecessary operations that were initially performed. The reason why we see such a steep decrease from the initial implementation is that the water distribution network is sparse as a node will only link to nodes from its neighbourhood.

If you take a look at the size of the networks we can observe that for n nodes, we have approximately n links, which means that our adjacency matrix will have $n(n-2)$ 0 elements (i.e. for each link, there are 2 non-zero elements). Consequently, if we take a look at Network 4, instead of performing matrix operations and storing $2369 \times 2369 = 5612161$ elements, we only consider $2369 \times 2 = 4738$. This means that more than 99.9% of the computations carried on in the naive implementation were additions and multiplications of 0.

Another important aspect is that the solvers that perform the best for us (in terms of CPU time for now) are the direct solvers and, in particular, the LLT solver. We will now take a look

Table 7.2: A simulation time comparison of the *latest* implementations of the hydraulic solver. The times shown are in *ms*. The tolerance level is $\epsilon = 10^{-6}$.

Solver (Size)	Network 1 (7 x 9)	Network 2 (91 x 113)	Network 3 (865 x 950)	Network 4 (2303 x 2369)	Network 5 (4577 x 4648)
LLT Factorization	0.089	4.902	10.456	22.417	41.356
LDLT Factorization	0.095	5.121	10.662	23.592	43.711
LU Factorization	0.096	5.088	10.701	24.032	44.882
Conjugate Gradient	1.43	27.639	74.122	149.064	420.01
Modified Conj. Grad.	1.54	30.43	78.058	157.332	423.947
Jacobi	1.67	35.912	79.51	161.232	425.553
Gauß-Seidel	1.68	35.983	79.984	162.735	425.342
GMRES	1.5	32.529	77.061	153.964	422.71
CWSNet	0.128	6.371	23.104	147.363	248.437

at the convergence rate of each of the solvers and see the number of iterations required by them.

7.2.2 Rate of Convergence

We will now turn on to see how convergence is achieved for the different implementations we use. In particular, we will look at the number of iterations coupled with the rate at which the error decreases.

Note that in this case, we will not consider the difference between the naive and the latest implementations as the methods do not change, only the underlying implementation and data structures are different, which should not affect the result we obtain after a certain iteration.

We will split the methods into two different categories:

- Direct Methods
- Iterative Methods

The motivation for this comes from the fact that each of the direct solvers and iterative solvers have very similar convergence properties, respectively. This can be seen in Table 7.3 where the number of iterations required for the direct methods is at about 50% of the iterative methods. This is part of the reason why direct methods provide better performance results in terms of overall simulation time.

For the figures in this section, the legend should be read as:

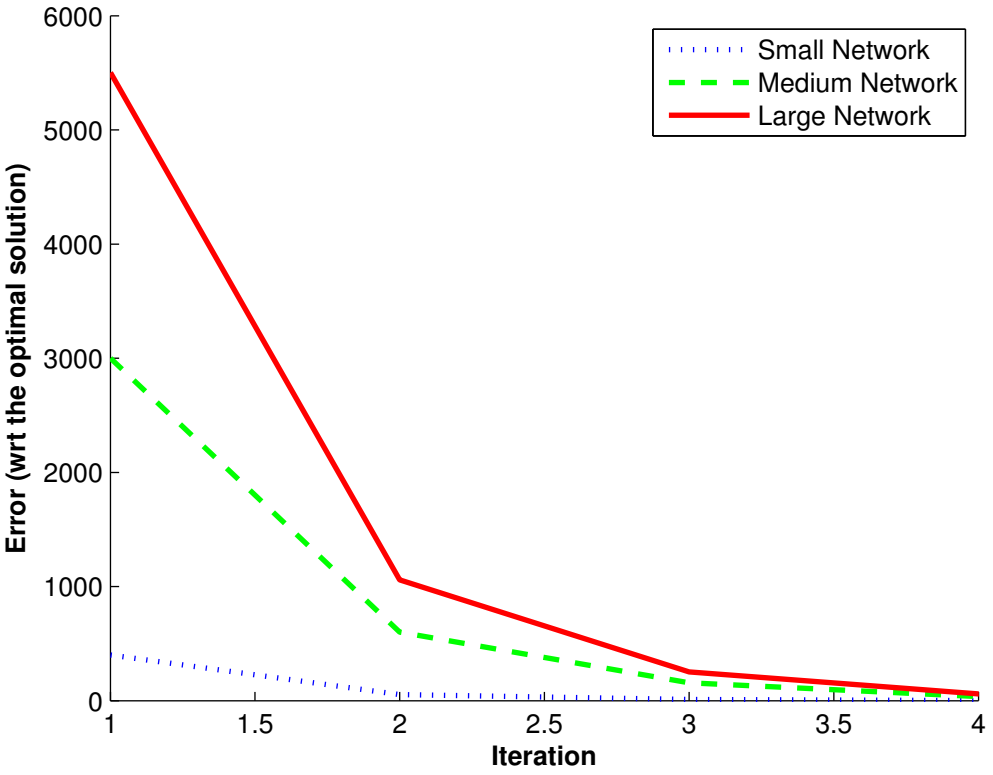
- Small Network - 91 x 113
- Medium Network - 2303 x 2369
- Large Network - 4577 x 4648

In Figure 7.1, we can see how the error decrease with each iteration for the direct methods. We can observe that convergence is achieved rapidly and within just six iterations the error becomes

Table 7.3: Number of iterations comparison required for convergence for different implementations. The tolerance level is $\epsilon = 10^{-6}$.

Solver (Size)	Network 1 (91 x 113)	Network 2 (2303 x 2369)	Network 3 (4577 x 4648)
LLT Factorization	7	15	32
LDLT Factorization	8	15	34
LU Factorization	8	16	34
Conjugate Gradient	15	27	54
Modified Conj. Grad.	15	28	56
Jacobi	15	29	58
Gauß-Seidel	15	29	58
GMRES	15	29	58

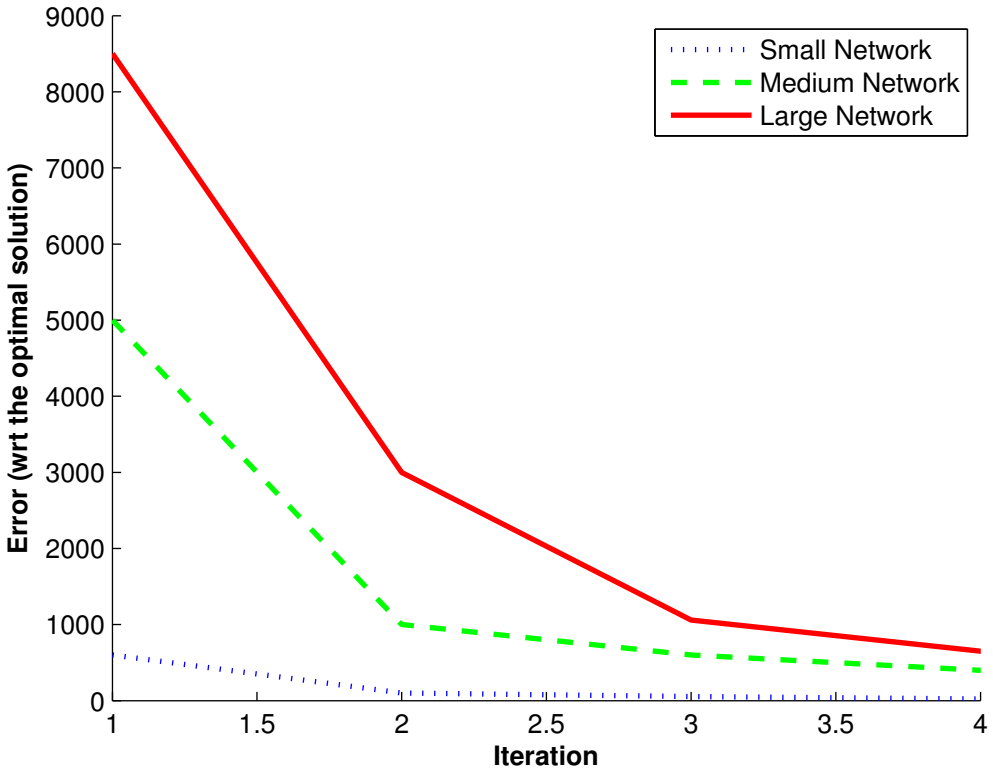
Figure 7.1: Convergence rate of direct methods for 3 different networks that vary in size.



very small even for the large network case.

This result agrees with the theoretical rate of convergence of Newton-Raphson for when the initial estimate of the solution comes close to the optimal solution. Another advantage comes from the fact that the objective function we are minimizing is very similar to a quadratic function, which makes Newton-Raphson an even better choice. For more details, please refer to section 4.3.1 on convergence analysis.

Figure 7.2: Convergence rate of iterative methods for 3 different networks that vary in size.

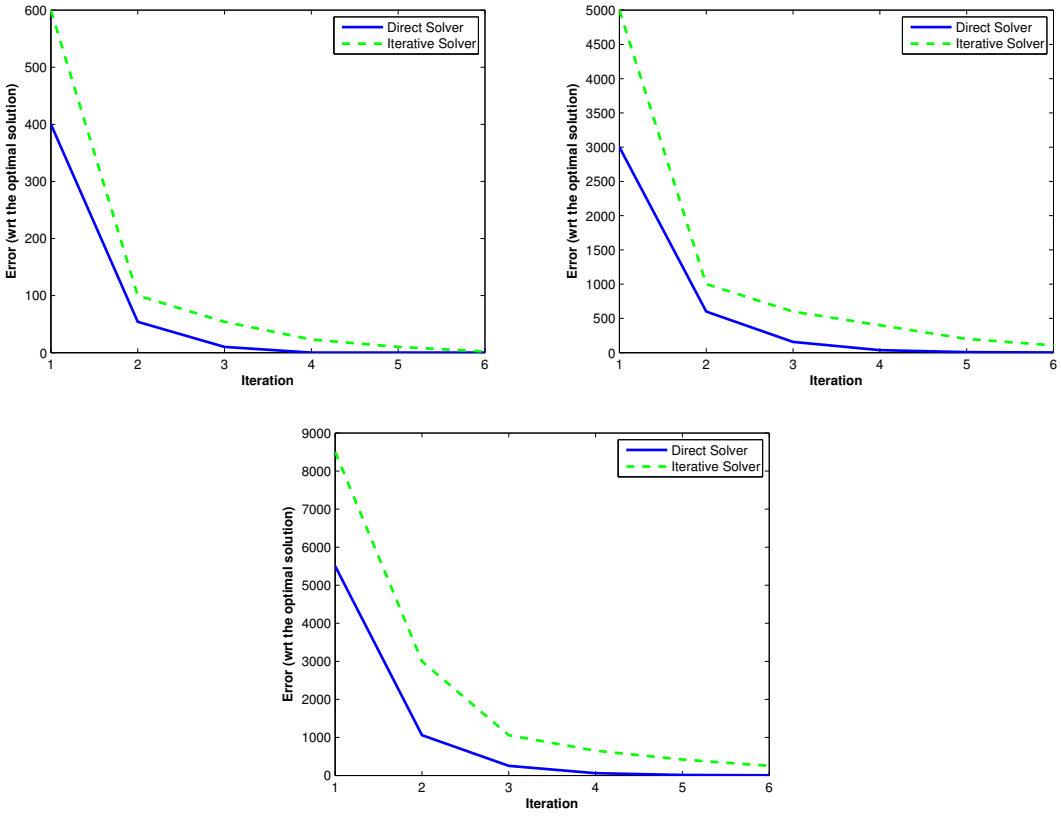


In Figure 7.2, we can see the same analysis done for the iterative methods. In this case, we achieve convergence slower than before, although the trend is similar. Figure 7.2 resembled Figure 7.1, only this time the error are larger and it is only after 12 iterations that we can say that the error becomes very small.

In Figure 7.3, we can see how the direct and iterative solvers compare on three different networks of various sizes. In all of them, the iterative solver arrives at the optimal solution in a larger number of steps than the direct method. In fact, the number of steps required is just under double of the number required for direct methods.

We have established practical results of convergence and now we will take a look at how the error accuracy affects the performance of our solver.

Figure 7.3: A comparison between direct and iterative methods for 3 different networks. Up left shows the small network. Up right is the medium network. Bottom centre is the large network.



7.2.3 Optimal Solution Accuracy

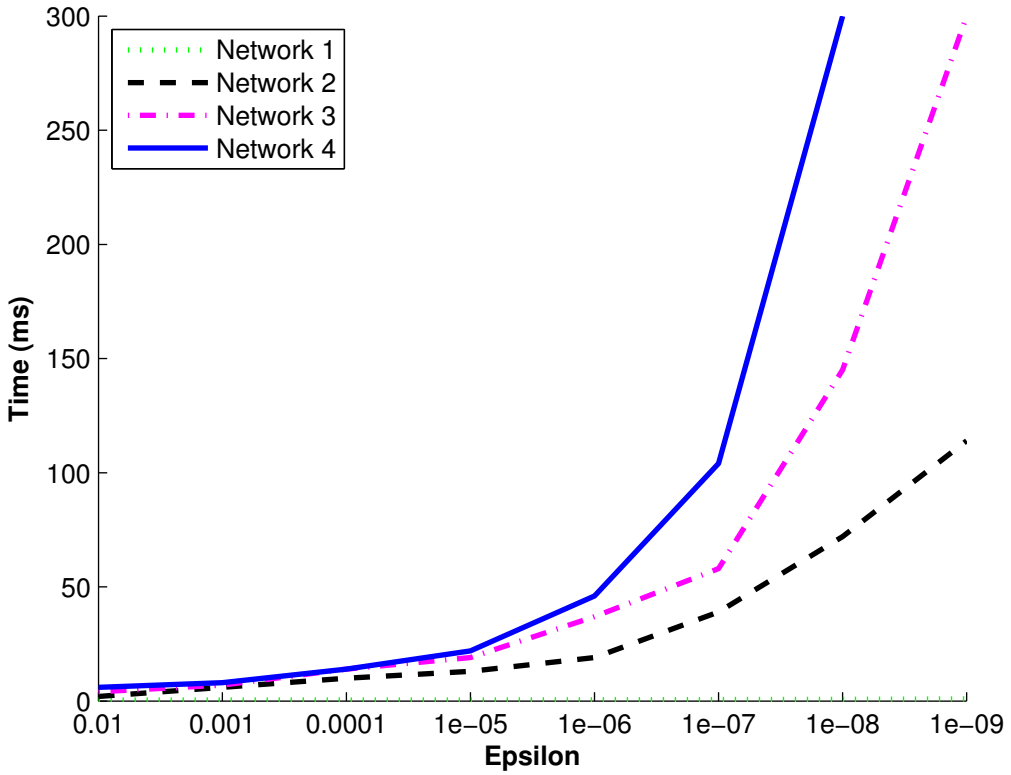
Before commencing the analysis of the ϵ-level accuracy, it is worth mentioning that from an engineering view point, a standard error in accuracy of $\epsilon = 10^{-6}$ is very good as the current existing implementations use this for their optimality check.

We will take a look at how the computational time and convergence rate are affected when varying the error of accuracy of our solution. For this, we will take five networks of different sizes and refer to the LLT implementation.

In Figure 7.4, you can see how the solution accuracy affects the CPU time spent on the simulation. Up until a level of $\epsilon = 10^{-6}$, we obtain the solution within $50ms$, but after that point, we really start to see that for large networks it becomes increasingly difficult to converge.

We can see a similar result in Figure 7.5, when the number of iteration increases rapidly as we require a better solution.

Figure 7.4: A comparison in terms of CPU time required for the hydraulic simulation between different networks of increasing size. Note that the computational time required increases rapidly as we increase the tolerance level ϵ.



7.3 Smooth Framework

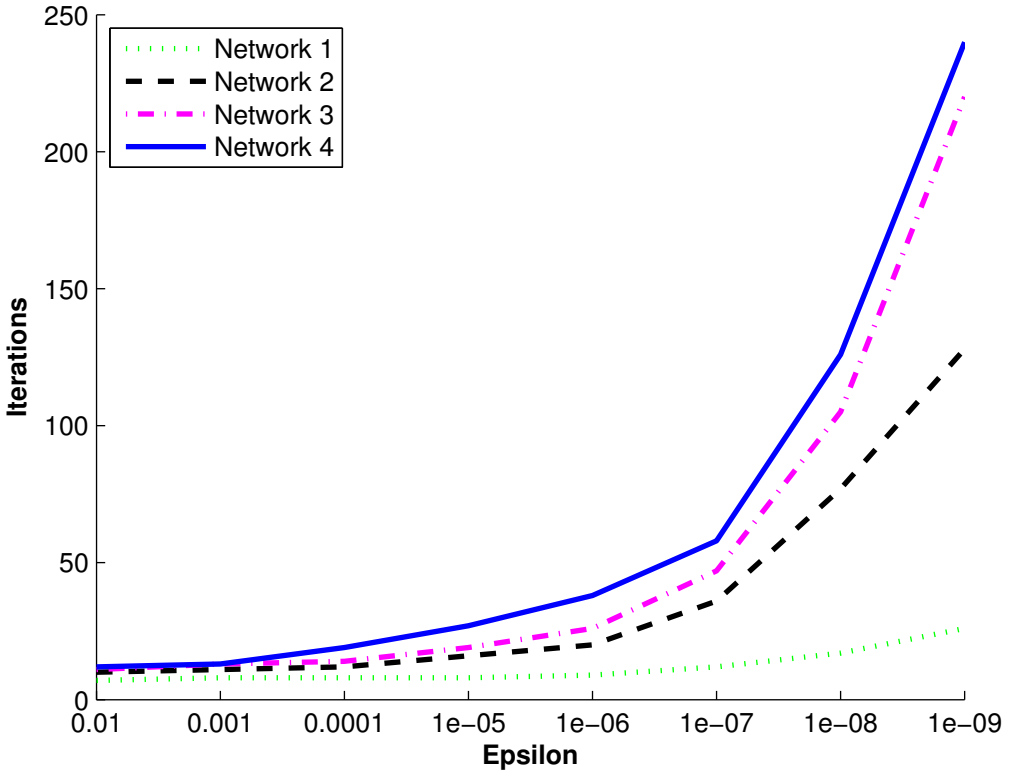
We will now focus our attention on the evaluation of the new theoretical framework that we have adopted in order to solve the hydraulic equations. Our main interest for this part is to assure that this method is at least as fast as the normal one or comparable.

The reason why we are not looking necessarily at speeding up the solver significantly is because adopting the smoothing approach already gives us a big benefit otherwise in that it provides a rigorous mathematical framework, which can be *safely* used by an optimisation algorithm such as Newton-Raphson. Moreover, it allows us to easily extend the model without having the fear that it will produce faulty results. If we think back at what has been discussed in Chapter 4, we find that the non-smooth version is not suitable to be used with Newton-Raphson as it does not satisfy the algorithm's conditions.

Nonetheless, we will go over the same analysis as in the previous section. We note that the values of μ used in our evaluation are chosen according to the results in Chapter 4, unless otherwise stated. Thus, we have:

$$\mu = \frac{1}{\sqrt{\beta}} \left(\frac{\epsilon}{2}\right)^{\frac{1}{2n+2}} \tag{7.1}$$

Figure 7.5: A comparison in terms of number of iterations required for the convergence of the optimisation algorithm between different networks of increasing size. Note that the number of iterations required increases rapidly as we increase the tolerance level ϵ.



7.3.1 Computational Time

CPU time will be considered for the fastest solver we have, which is the direct LLT factorization implementation. We will compare the results of this solver using the non-smooth function and the smooth function approximated by the Huber function.

The tests are conducted for various networks and we look at the computational time averaged over 1000 simulations that is required to solve *all* of the 96 optimisation problems for the 96 time intervals in a day. We simply choose all of the time intervals as it offers a more accurate description of the results.

where the Huber function is as described by equation (4.8) and the LogSumExp approximation function is:

$$f_\mu(x) = \mu \log\left(\frac{e^{\frac{x}{\mu}} + e^{\frac{-x}{\mu}}}{2}\right) \tag{7.2}$$

and the Sqrt approximation is as:

$$f_\mu(x) = \sqrt{x^2 + \mu^2} - \mu. \tag{7.3}$$

Both of them are smooth approximations for the absolute value, however the Huber function best approximates the absolute value and it is the reason why we use it the most since we also

Table 7.4: Non-smooth and Smooth implementations comparison in terms of computational time required to solve the hydraulic equations. The tolerance level is $\epsilon = 10^{-6}$.

Solver (Size)	Network 1 (91 x 113)	Network 2 (2303 x 2369)	Network 3 (4577 x 4648)
Non-smooth	24.43	9187	17198
Smooth (Huber)	22.59	8412	16550
Smooth (LogSumExp)	22.71	8452	16559
Smooth (Sqrt)	22.78	8455	16680

obtain the most accurate results.

As we can see in Table 7.4, the computational time does not change significantly, although we might see a slight improvement with our new method. Thus, for the three networks above, we get the following improvement when comparing the Non-smooth implementation to the Smooth (Huber) implementation:

$$\frac{22.59}{24.43} \times 100\% \approx 92.4\%,$$

for Network 1. Next, for Network 2, we have:

$$\frac{8412}{9187} \times 100\% \approx 91.5\%$$

and, lastly, for Network 3, we obtain:

$$\frac{16550}{17198} \times 100\% \approx 96.23\%,$$

which is not a substantial improvement in terms of performance, but it is a very good result for our purposes since not only can we safely use a mathematically rigorous framework and not worry about the algorithm producing wrong results, but we can also benefit from slightly better computational performance.

Another important fact to mention is that the difference between the solutions obtained through the two methods is marginal as it can be observed from Table 7.5. There you can see the optimal solutions for H and Q for a small newtork. At the very worst, the difference appears in among the last significant digits (fifth or after) and we can assume that the quantity lost is negligible.

Table 7.5: Non-smooth and Smooth implementations comparison in terms of optimal solution.

Link/Node	Non-smooth	Smooth (Huber)	Relative error (Smooth)
Q_1	0.0116387	0.011639	2.5×10^{-5}
Q_2	0.0116387	0.011639	2.5×10^{-5}
Q_3	0.00513486	0.00513486	0
Q_4	0.006	0.006	0
Q_5	0.00350389	0.00350411	6.5×10^{-5}
Q_6	-0.00386514	-0.00386514	0
Q_7	-0.00336125	-0.00336103	6.5×10^{-5}
Q_8	-0.00336125	-0.00336103	6.5×10^{-5}
Q_9	0.0116387	0.011639	2.5×10^{-5}
H_1	139.701	139.701	0
H_2	101.266	101.266	0
H_3	101.048	101.048	0
H_4	96.4666	96.467	4.1×10^{-6}
H_5	86.738	86.7391	1.2×10^{-5}
H_6	99.8351	99.8351	0
H_7	99.9782	99.9782	0

Table 7.6: Number of iterations comparison required for convergence for non-smooth and smooth cases. The tolerance level is $\epsilon = 10^{-6}$.

Solver (Size)	Network 1 (91 x 113)	Network 2 (2303 x 2369)	Network 3 (4577 x 4648)
Non-smooth	7	15	32
Smooth (Huber)	6	13	27
Smooth (LogSumExp)	6	13	28
Smooth (Sqrt)	6	13	29

7.3.2 Rate of Convergence

Much like in the case of computational time required, we see a slight decrease in the number of iterations required for convergence. This comes as no surprise since in both cases we are performing the same computations, but in the smooth case, the gradient is defined on the whole domain. Thus, we are not risking to take a faulty step that may occur in the non-smooth case.

Table 7.6 prints some results observed on the number of iterations in the non-smooth case and the smooth cases analysed.

A final important point in this case is that as we increase the value of μ, we obtain a better rate of convergence at the expense of moving away from the original optimal solution. We refer to Table 7.7 to validate this result.

Table 7.7: Number of iterations comparison required for convergence for non-smooth and smooth cases. We observe that as we increase the value of μ, the number of iterations decreases. This is because we achieve a *more quadratic* function which is better suited for the Newton-Raphson algorithm.

Solver (Size)	Network 1 (4577 x 4648)	Network 2 (2303 x 2369)	Network 3 (91 x 113)
Non-smooth	25	24	17
Smooth (optimal μ)	24	24	17
Smooth ($\mu = 0.01$)	17	19	17
Smooth ($\mu = 0.1$)	16	17	13
Smooth ($\mu = 1$)	15	16	11

7.3.3 Optimal Solution Accuracy

This section gives rise to a more interesting analysis since depending on the value of μ which we choose, we see a difference between the optimal solution of the non-smooth problem and the one of the smooth problem.

The reason why this happens is because we are changing the problem to use a μ-smooth approximation of the problem and so we are changing the objective function. Moreover, the more we increase μ, the better chance we have at rapid convergence since the problem becomes *smoother* and we can take larger steps towards the optimal solution. On the other hand, increasing μ decreases the similarity factor between the smooth approximation and the non-smooth function, resulting in us solving a more different problem.

Figure 7.6 best illustrates how increasing the value of μ increases the difference between the optimal solution of the non-smooth problem and the smooth problem. However, there is no exact threshold for the value of μ and the error we obtain is network-dependent. Running the same simulation on a different network yielded the same results as we varied μ from 10^{-6} to 10.

7.4 Comparison with CWSNet and Epanet

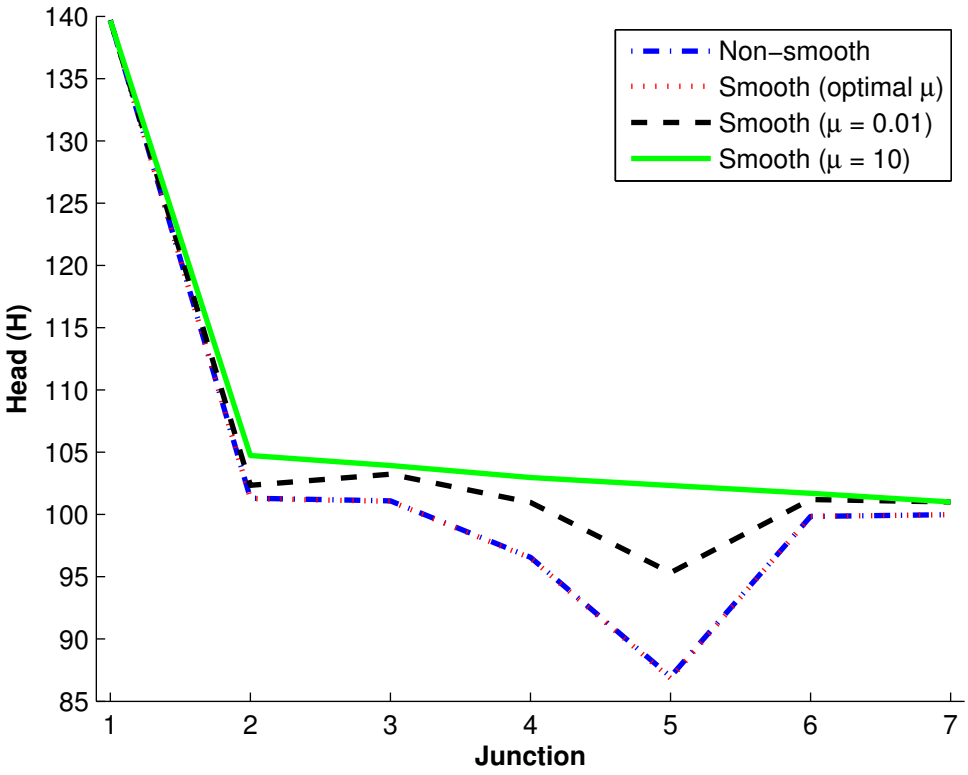
CWSNet and Epanet represent the standard tools used in industry for tackling the hydraulic simulation problem. Epanet was first to appear and is under development for more than two decades and it is the most complete software in the field. CWSNet is more recent, but still ten years old and it appeared as a motivation to improve performance of Epanet.

A large number of developers have contributed and still contribute to the Epanet project, while CWSNet was build in a small academic team over the course of ten years.

However, CWSNet has a large number of Epanet dependecies, especially in the critical part of the system. The linear solvers and internal data structures used for representing the hydraulic network is entirely taken from Epanet. As a consequence, they offer similar performance.

To conclude, despite having significantly less resources than the other two projects (in terms

Figure 7.6: In this Figure, we can see how optimal solution is different as we vary the value of μ. We note that, the error becomes larger as we increase μ. Also, at optimality (μ as in equation (7.1)), the solution coincides with the original one, since the value of μ is very small.



of number of developers, time allocated, initial knowledge and background), we have managed to produce important results, which improve upon the computational performance of the solver. Moreover, the accuracy of the solution is very high as it can be seen in Table 7.8.

Table 7.8: Our solver's solution compared to CWSNet's and Epanet's solutions. We are showing the relative error for the Non-smooth and Smooth version, respectively. The tolerance level is $\epsilon = 10^{-6}$.

Link/Node	Relative Error (Non-smooth)	Relative Error (Smooth)	CWSNet	Epanet
Q_1	6.5×10^{-6}	5.8×10^{-6}	0.0117	0.0116
Q_2	2.3×10^{-6}	6.1×10^{-6}	0.0117	0.0116
Q_3	9.4×10^{-6}	1.9×10^{-6}	0.00515	0.00514
Q_4	0	0	0.006	0.006
Q_5	2.1×10^{-6}	4.4×10^{-6}	0.00358	0.00358
Q_6	7.7×10^{-6}	2.9×10^{-6}	-0.00385	-0.00385
Q_7	0	0	-0.00327	-0.00327
Q_8	0	0	-0.00327	-0.00327
Q_9	1.6×10^{-6}	5×10^{-6}	0.00117	0.00116
H_1	0	0	139.7	139.7
H_2	0	0	101.3	101.3
H_3	8.3×10^{-7}	6.2×10^{-7}	101.09	101.07
H_4	0	0	96.56	96.5
H_5	9.2×10^{-7}	8.1×10^{-7}	86.98	86.88
H_6	0	0	99.85	99.84
H_7	0	0	99.98	99.97

8 Conclusion

We have tackled the challenge of improving the standard solver for hydraulic simulations in two rather different perspectives. We have first performed a rigorous numerical analysis by looking at various techniques and methods to increase computational speed of matrix operations and linear solvers. On the other perspective, we have integrated and mathematically proved that a new optimisation framework is well suited to solve the hydraulic equations whilst improving upon the bound of complexity for the rate of convergence of the Newton-Raphson algorithm used.

8.1 Summary of Achievements

We have addressed, in turn, each of the objectives we have established at the beginning of the project.

First of all, we have introduced a novel methodology to solve the nonlinear system of hydraulic equations that offers many advantages over the original one, whilst maintaining the accuracy of optimal solutions. The idea is that the new optimisation framework replaces the non-smooth component by a μ-smooth approximation function and solves the smooth problem instead. We will mention a the advantages of this approach:

- We have achieved a new lower bound on the rate of convergence of the algorithm improving from the usual subgradient methods that perform in $O(1/\epsilon^2)$ to $O(\log_2(\log_2(1/\epsilon)))$. This result is proved in Chapter 4.
- We have placed the optimisation problem in a framework that is suited to be solved by standard first order and second order algorithms including Newton-Raphson algorithm. This achievement is of paramount importance because until now, we were solving the problem using Newton-Raphson, which assumes our objective function to be twice continuously differentiable and our objective function was not since it contained the absolute value function, which is not differentiable in 0. Our method solves this problem by making use of a smooth, differentiable approximation of the non-smooth objective function.
- Achieving this mathematical rigour, allows us to extend the model in multiple directions without worrying that the optimal solutions we obtain are not correct, because we are wrongly using an algorithm. For instance, we can impose constraints to our model which describe better the physical nature of the problem such as making the pressure at each node take a non-negative value $H \geq 0$.

Moving on, we have studied the optimisation problem in detail and we have rigorously analysed different methods through which one can solve the hydraulic equations, keeping in mind perfor-

mance is a key aspect. Thus, we were able to exploit the nature of the problem and find a good match in terms of linear solver and implementation, which proved to perform better than the industry standard by improving on the overall computational time required. The idea is that we exploited the sparse nature of the problem and used clever matrix storage schemes, which facilitate the fast execution of matrix multiplications and decompositions. Also, we ran simulations on various different linear solvers to determine the one that performs the best for our problem. The details of the analysis are given in Chapter 3.

We provide a robust and reliable implementation of the hydraulic solver, which can be applied to any general purpose water distribution network. This already gives researchers a strong framework where they can implement and validate any new theories and ideas they may have.

8.2 Future Work

As per the ideas outlined in Chapter 5, there are a few possible extensions and research work that can be done in relation to our project. These vary from extending the theoretical work that has been done, to improving the computational performance of the solver, or working on the implementation to extend the model in order to capture all the elements that may be present.

We will now view some of the possible next steps:

- An interesting extension is to study the uncertainty in the parameters of the optimisation model and in particular the customer demand q. The reason for this is because our model makes certain assumptions regarding the customer demand, which, albeit valid, may not always describe reality extremely accurately. It is for this reason that a *robust* formulation of the optimisation problem will be helpful as we will be able to find an optimal solution for the hydraulic equations subject to some ξ level of uncertainty in customer demand.
- Another interesting extension would be advancing the numerical analysis that was done to provide further improvements on the time complexity of the hydraulic solver. This can be done by implementing a concurrent linear solver that might achieve better speed to reaching the optimal solution. However, the complexity we currently achieve without carrying any parallel computations is of $O(10ms)$ even for large size problems, a result that is very good for practical applications.
- Complete the model we implemented by adding more elements and establishing their particularities. Elements in the water network tend to have the same behaviour and this holds especially for valves, which only differ slightly among them. This extension is purely implementation-based and did not fit the aim of our project very well.
- Another implementation based extension is the modelling of a *pressure*-driven model as opposed to the *demand*-driven one that we have implemented. For the same reasons as above, this extension did not fit the aim of our project as there was little research work to be done and more development work.

9 | Bibliography

[1] E. Todini and S. Pilati, *A Gradient Algorithm for the Analysis of Pipe Networks,* Computer applications in water supply: vol 1 – systems analysis and simulation, 1988.

[2] Amir Beck and Marc Teboulle, *Smoothing and First Order Methods: A Unified Framework,* Society for Industrial and Applied Mathematics, Journal on Optimisation, 2012.

[3] Robert Wright, Ivan Stoianov, Panos Parpas, Kevin Henderson, John King, *Adaptive Water Distribution Networks with Dynamically Reconfigurable Topology,* Journal of Hydroinformatics, `http://www.iwaponline.com/jh/up/jh2014086.htm`, 2014.

[4] Aharon Ben-Tal and Arkadi Nemirovski, *Robust solutions of Linear Programming problems contaminated with uncertain data,* Mathematical Programming, Volume 88, Issue 3, pp 411-424, 2000.

[5] Paul Grigoras, Gary Chow, Pavel Burovskiy, Wayne Luk, *An Efficient Sparse Conjugate Gradient Solver Using a Benes Permutation Network,* to appear in Proc. FPL, 2014.

[6] Y. Nesterov, *Universal gradient methods for convex optimisation problems,* Centre for Operations Research and Econometrics, 2013.

[7] Y. Nesterov, *Smooth minimization of non-smooth functions,* Mathematical Programming, Volume 103, pp. 273-299, 2005.

[8] Bradley J. Eck, M. ASCE and Martin Mevissen, *Fast non-linear optimisation for design problems on water networks,* World Environmental and Water Resources Congress 2013: pp. 696-705.

[9] Keshaw D. *The incomplete Choleski-Conjugate gradient method for the iterative solution of systems of linear equations,* Journal of Computational Physics, 26, pp. 43-65, 1978.

[10] Ayres F. *Theory and problems of matrixes,* Mc. Graw-Hill Co., New York, 1962.

[11] Ezio Todini and Lewis A. Rossman, M. ASCE, *Unified Framework for Deriving Simultaneous Equation Algorithms for Water Distribution Networks,* Journal of Hydraulic Engineering, 139(5), 511–526, 2013.

[12] Peter N. Brown and Youcef Saad, *Convergence Theory of Nonlinear Newton-Krylov Algorithms,* Society for Industrial and Applied Mathematics, Journal on Optimisation, 1994.

[13] Noreen Jamil, *A comparison of Direct and Indirect Solvers for Linear Systems of Equations,* Available online at `http://www.ijes.info/2/2/42542211.pdf`, 2012.

[14] Ajiz M, and Jennings A., *A robust incomplete Choleski conjugate gradient algorithm,* International Journal for numerical methods in engineering, Vol. 20, pp. 949-966, 1984.

[15] L. Vandenberghe (UCLA), *Smoothing EE236C,* Available online at http://www.seas.ucla.edu/~vandenbe/236C/lectures/smoothing.pdf, 2013-2014.

[16] David G. Luenberger (Stanford University) and Yinyu Ye (Stanford University), *Linear and Nonlinear Programming Third Edition,* International Series in Operations Research & Management Science, Vol 116, 2008.

[17] Stephen Boyd (Stanford University) and Lieven Vandenberghe (UCLA), *Convex Optimisation,* Cambridge University Press, 2009.

[18] R. T. Rockafellar, *Convex Analysis,* Princeton University Press, Vol. 28, 2009.

[19] Gilbert Strang, *Linear Algebra and its Applications,* Thomson Brooks/Cole, Available online at http://www.hua.edu.vn/khoa/fita/wp-content/uploads/2013/10/Linear_algebra_and_its_applications_Fourth_Edition.pdf, 2013.

[20] Kenneth Kutler, *Linear Algebra, Theory and Applications,* Available online at http://textbookequity.org/linear-algebra-theory-and-applications, 2013.

[21] Panos Parpas, *Computing For Optimal Decisions Course and Course Material,* Imperial College London, 2013.

[22] Abbas Edalat and Peter G. Harrison, *Computational Techniques Course and Course Material,* Imperial College London, 2013.

[23] Daniel Kuhn, *Operations Research Course and Course Material,* Imperial College London, 2012.

[24] Epanet, http://www.epa.gov/

[25] i2O, http://www.i2owater.com/home/

[26] CWSNet, http://emps.exeter.ac.uk/engineering/research/cws/resources/cwsnet/

[27] LAPACKPP, http://lapackpp.sourceforge.net/, http://www.netlib.org/lapack/

[28] Eigen, http://eigen.tuxfamily.org/index.php?title=Main_Page

[29] SuiteSparse, http://www.cise.ufl.edu/research/sparse/SuiteSparse/

[30] Books LLC, *Fluid Dynamics: Rheology, Mach Number, Bernoulli's Principle, Cavitation, Pump, Superfluid, Acoustic Theory, Grashof Number, Lift*, 2010.

[31] Paul Grigoras, Corina Ciobanu, Karol Pysniak, Radu Baltean-Lugojan, *Prize Winning and Distinguished Reports,* , Available online at http://www3.imperial.ac.uk/computing/teaching/ug/ug-distinguished-projects, 2013.

10 | Appendix

10.1 Proof of Theorem 1

Let $\mu > 0$ and denote $F = f + g_\mu$. By *Definition 1*, one has $L_F = L_f + K + \frac{\alpha}{\mu}$. Therefore, the sequence generated by the method $\mathcal{M}$ when applied to (2.20) satisfies for all $k \geq 1$:

$$H_\mu(x_k) - H_\mu^* \leq \left(L_f + K + \frac{\alpha}{\mu}\right) \frac{\Lambda}{k^2} \tag{10.1}$$

Since g_μ is a μ-smooth approximation of g with parameters (α, β, K), by *Definition 1*, there exists β_1, β_2 satisfying $\beta_1 + \beta_2 = \beta > 0$ for which:

$$H(x) - \beta_1\mu \leq H_\mu(x) \leq H(x) + \beta_2\mu \tag{10.2}$$

Thus, in particular, the following inequalities hold:

$$H^* \geq H_\mu^* - \beta_2\mu \tag{10.3}$$

$$H(x_k) \leq H_\mu(x_k) + \beta_1\mu \tag{10.4}$$

and hence, using (9.1), we obtain:

$$H(x_k) - H^* \leq H_\mu(x_k) - H_\mu^* + (\beta_1 + \beta_2)\mu \leq (L_f + K)\frac{\Lambda}{k^2} + \frac{\alpha\Lambda}{k^2}\frac{1}{\mu} + \beta\mu \tag{10.5}$$

Minimizing the right-hand side of (9.5) with respect to $\mu > 0$ we obtain:

$$\mu = \sqrt{\frac{\alpha\Lambda}{\beta}}\frac{1}{k} \tag{10.6}$$

Plugging the above expression for μ in (9.5) we obtain:

$$H(x_k) - H^* \leq (L_f + K)\frac{\Lambda}{k^2} + 2\sqrt{\alpha\beta\Lambda}\frac{1}{k} \tag{10.7}$$

Thus, given $\epsilon > 0$, to obtain an ϵ-optimal solution satisfying $H(x_k) - H^* \leq \epsilon$, it remains to find values k for which:

$$(L_f + K)\Lambda\frac{1}{k^2} + 2\sqrt{\alpha\beta\Lambda}\frac{1}{k} \leq \epsilon \tag{10.8}$$

Denoting $t = \sqrt{\Lambda}\frac{1}{k}$, (9.8) translates to:

$$(L_f + K)t^2 + 2\sqrt{\alpha\beta}t - \epsilon \leq 0 \tag{10.9}$$

which is equivalent to:

$$\sqrt{\Lambda}\frac{1}{k} = t \leq \frac{-\sqrt{\alpha\beta} + \sqrt{\alpha\beta + (L_f + K)\epsilon}}{L_f + K} = \frac{\epsilon}{\sqrt{\alpha\beta} + \sqrt{\alpha\beta + (L_f + K)\epsilon}}$$

Using the value of the upper bound just established for $\Lambda\frac{1}{k}$ in (9.6), we obtain the desired expression for μ stated in equation (2.1). We have thus shown that by choosing μ as in equation (2.1) and k satisfying:

$$k \geq \frac{\sqrt{\alpha\beta\Lambda} + \sqrt{\alpha\beta\Lambda + (L_f + K)\epsilon\Lambda}}{\epsilon} \tag{10.10}$$

we have $H(x_k) - H^* \leq \epsilon$.

To complete the proof and obtain the desired lower bound for k as given in equation (2.2), note that for any $A, B \geq 0$, the following inequality holds:

$$\sqrt{A} + \sqrt{A + B} \leq 2\sqrt{A} + \sqrt{B} \tag{10.11}$$

By invoking (9.11) with

$$A = \frac{\alpha\beta\Lambda}{\epsilon^2}$$
$$B = \frac{(L_f + K)\Lambda}{\epsilon}$$

together with (9.10), the desired result in equation (2.2) follows.

10.2 Validating L_g

Suppose now that $x > \mu$. We obtain:

$$c_2 n \left| \left(x - \frac{\mu}{2}\right)^{n-1} - \left(y - \frac{\mu}{2}\right)^{n-1} \right| \leq L_g |x - y| \tag{10.12}$$

Let $\tilde{x} = x - \frac{\mu}{2}$ and $\tilde{y} = y - \frac{\mu}{2}$. Using the fact that $\tilde{x} - \tilde{y} = x - y$, we obtain:

$$c_2 n |\tilde{x}^n - \tilde{y}^n| \leq L_g |x - y| \tag{10.13}$$

and now we can use equation (4.54) to obtain:

$$\frac{c_2 n}{n\Psi^{n-1}} \leq L_g \tag{10.14}$$

and since

$$\frac{c_2}{\Psi^{n-1}} \leq \frac{c_2}{\mu^{n+1}} \tag{10.15}$$

we have validated the Lipschitz constant found.

The last case is similar to the above. Just let $\tilde{x} = -x - \frac{\mu}{2}$ and $\tilde{y} = -y - \frac{\mu}{2}$. We will obtain all the results including (10.15), which will help us establish the Lipschitz constant as:

$$L_g = \frac{c_2}{\mu^{n+1}} \tag{10.16}$$