Improving the Solution Method in Wanda

Literature Study

by

L. Huijzer

Student number:	4258878	
Project duration:	December 5, 2017 – August 31, 2018 (expected)	
Thesis committee:	Dr. ir. M.B. van Gijzen,	TU Delft, supervisor
	Prof. Dr. Ir. C. Vuik,	TU Delft
	Prof. Dr. Ir. A.W. Heemink,	TU Delft
	Ir. S. van der Zwan,	Deltares

TU Delft

Deltares
Enabling Delta Life

Abstract

The Wanda software package developed by Deltares can used for computing both steady state and transient fluid flow in pipeline systems. Steady state simulations are used for initial system design and transient flow simulations are used for doing water hammer analysis in pipeline systems. The flow speed and pressure are the main quantities of interest. For both types of simulations a system consisting of both linear and non-linear equations needs to be solved for the unknowns, speed and pressure. This system is solved by linearising the equations using the Newton-Raphson method and solving the resulting system of linear equations. Currently, this is done by using a matrix solver from the proprietary IMSL numerical library. The problem is that this solver sometimes either crashes or gets stuck in an infinite loop when dealing with singular matrices, while the proprietary nature of the library only allows for limited troubleshooting. Moreover, the library requires a paid license.

The goal of this thesis project is to find a solution method that is both robust and easily maintainable, while being at least as fast as the current solution method. Several techniques are considered. To increase robustness, problematic pipeline systems could potentially be identified before attempting to solve the system of equations by considering a graph representation of the system of equations or by re-evaluating the underlying physical model. Another approach is to detect singular matrices via a rank-revealing decomposition or condition number estimation. To meet the performance requirement, fill-in reduction strategies, alternatives to the Newton-Raphson method and optimisations for other parts of the solution method are considered. The choice of numerical library also plays a role in performance. The Fortran libraries LAPACK and MUMPS are considered as alternatives to the current library as they are open source and offered under a permissive license.

Nomenclature

Symbol	Quantity	Unit
A	Cross-sectional area of the fluid	m^2
c	Pressure wave propagation speed	ms^{-1}
g	Gravitational acceleration	ms^{-2}
H	Hydraulic or piezometric head (pressure)	m
K	Bulk modulus	$kgm^{-1}s^{-2}$
O	Wetted circumference of the fluid	m
p	Pressure	$kgm^{-1}s^{-2}$
p_v	Vapor pressure	$kgm^{-1}s^{-2}$
Q	Volumetric flow rate	m^3s^{-1}
t	Time	s
v	Velocity (average)	ms^{-1}
x	Space	m
z	Elevation level	m
η	Dynamic viscosity	$kgm^{-1}s^{-2}$
λ	Friction coefficient	s^2m^{-5}
ν	Kinematic viscosity	m^2s^{-1}
ρ	Density	kgm^{-3}

Contents

1 Introduction

Designing a pipeline system such as a drinking water network involves a lot of considerations. The right pumping equipment and pipeline diameter need to be selected so as to guarantee sufficient drinking water being available for every household. But these are not the only considerations. What happens, for example, when a pump trips or a valve unexpectedly closes? These kinds of situations cause pressure waves, called water hammer, to propagate through the system. Water hammer can cause damages to pumps and other equipment and can even result in broken pipes. Apart from the economic costs, the public health is also in play when dirt and other matter leaks into the drinking water network. In order to prevent these kind of problems a well designed pipeline system is required which is able to handle potential water hammer scenarios. Since doing experiments and using analytical methods are infeasible on such a scale, computational methods are required. This is where Wanda comes into play.

The current solution method in Wanda requires some attention. Improvements can be made in terms of robustness, maintainability and performance. Robustness is an issue since there are pipeline systems for which the solution method either crashes or gets stuck in an infinite loop. Connected to this issue is maintainability: part of the current solution method is based on proprietary software and hence does not lend itself to troubleshooting. A move to an open source implementation with a permissive license is desirable. The previous two aspects are key, but yielding ground in terms of performance is not an option. The focus of this literature study is on finding methods that yield improvements in these three areas.

1.1. Report Structure

Chapter 2 starts with introducing the Wanda model. The fluid dynamics for the component types will be treated, after which both the steady state and transient flow models and their numerical implementation will be covered. Chapter 3 illustrates, using examples, the problem with the current solution method. Test cases are introduced which are to be used to test both the robustness and performance of solution methods, as well as some preliminary test results of the current method. Chapter 4 introduces numerical methods which could improve the solution method in terms of robustness, maintainability and performance. The final chapter, Chapter 5, sets the scope of the research using research questions and proposes an approach to answering these questions.

1.2. Notation and Conventions

The following notation and conventions will be used throughout this report.

Important terms and definitions are written in **bold font** when introduced for the first time.

Matrices will be denoted by upper case letters, e.g.,

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{1.1}$$

and vectors in lower case bold font, e.g.,

$$\mathbf{u} = [u_1 \; u_2 \; \dots \; u_n]^\top. \tag{1.2}$$

In pseudo code, the following notation will be used for matrix and vector indices.

Algorithm 1.1 Notation Example

```
$M(i,j)$                 denotes the element $M_{ij}$
$M(i,:)$                 denotes the $i^{\text{th}}$ row
$M(:,j)$                 denotes the $j^{\text{th}}$ column
$u(k:l)$                 denotes the elements $k$ to $l$
$M(k:l,j)$               denotes the elements $k$ to $l$ of column $j$
$M(i,:)\cdot M(j,:)$     denotes the inner product between the $i^{\text{th}}$ and $j^{\text{th}}$ rows
```

Approximate solutions to equations are written with a hat. For example, $\mathbf{u}$ is an exact solution to $M\mathbf{u} = \mathbf{b}$, while $\hat{\mathbf{u}}$ is an approximate solution to the same equation.

Example 1.1. Examples are given in boxes.

2
Wanda

This chapter serves as a general introduction to Wanda. It includes some background information on Wanda, the description of the physical model and the numerical implementation. The material covered here is based on the notes on fluid dynamics and Wanda as provided by Deltares [4, 6], which in turn is (partially) based on the book on fluid dynamics by Wylie and Streeter [28].

2.1. Background

Wanda is a software package that allows for the design, control and optimisation of pipeline systems. It can be used to simulate gas or oil pipeline systems, drinking water networks, sewage systems and various other systems involving the transportation of fluids. These systems consist of hydraulic components such as pumps and reservoirs, as well as pipes. Important aspects of pipeline systems can be studied. These aspects include:

- Flow capacity, velocity and distribution
- Cavitation, water hammer and dynamic behaviour
- Pressure and safety
- Pump efficiency, system characteristics
- Performance monitoring and control induced pressure waves

This shows that Wanda can be used throughout the lifetime of a pipeline system. From initial design up to real time control and evaluation.

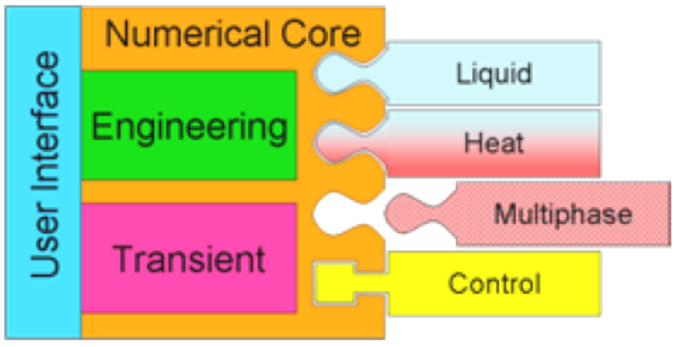


Figure 2.1: Wanda architecture.

As shown in Fig. 2.1, Wanda can be used to simulate liquid flow, heat transfer and multiphase systems. The control module allows for a system to be linked to a control system. This can be used to monitor and control the flow in a pipeline system.

Wanda can be used for two types of simulations, steady and transient flow. The steady state flow of a pipeline network can be used for the design of a pipeline system. A well designed network allows for a balanced flow and pressure. These type of calculations can be used for component selection, e.g., for the selection of pipe diameter and pumping equipment. The other simulation type calculates transient (or unsteady) flow in a pipeline system. This can be used to evaluate a system's performance and safety by measuring the pressure and flow in certain situations. These situations can include pump start-up, pump trip and other control actions such as closing a valve. Phenomena such as water hammer and cavitation are included in the model. **Water hammer** is a pressure wave travelling through the system, which can be caused by, e.g., the sudden closing of a valve. These waves can cause damages to pumps and other parts of the system. **Cavitation** is the formation of vapour cavities in liquid, which impacts the flow in a system and implosion of cavities can cause damages to pumps and other components. For economic and safety reasons, it is of special importance to evaluate the impact of these phenomena on a pipeline system.

2.2. Fluid Dynamics

The material discussed in this chapter only applies to the liquid module of Wanda. For the other modules the concepts are similar with the main difference being the relevant quantities and the equations governing the dynamics.

The Wanda model simulates one-dimensional fluid dynamics. The basic fluid properties are density ρ $[kgm^{-3}]$, pressure p $[kgm^{-1}s^{-2}]$ and speed v $[ms^{-1}]$. Related to p is the **vapour pressure** p_v $[kgm^{-1}s^{-2}]$. Vapour pressure is the absolute pressure at which a fluid vaporises. Vaporisation due to an increase in temperature at a given pressure is called **boiling**, whereas vaporisation due to a decrease in pressure at a given temperature is called **cavitation**. There are two additional properties of fluids that determine their behaviour. The first is **compressibility** K $[kgm^{-1}s^{-2}]$, which is a measure of the relative change in volume to change in pressure. Secondly, kinematic viscosity η $[kgm^{-1}s^{-2}]$ or dynamic viscosity ν $[m^2s^{-1}]$ is relevant. **Viscosity** is a measure of resistance to deformation by stress, which can, for example, be caused by friction along the pipe wall. This slows the fluid down.

For the purposes of which Wanda has been developed viscosity is always relevant, but compressibility not necessarily. More specifically, compressibility is only relevant when considering transient flow. A more detailed treatment of why this is the case will be given in Sections 2.4 and 2.5.

2.2.1. Pipeline Fluid Dynamics

The Wanda software package models fluid dynamics in pipeline systems. For this purpose the quantities of interest differ slightly from the ones which are of interests in fluid dynamics in general. Wanda users are primarily interested in the flow through and pressure in pipeline systems. These properties are measured by the **volumetric flow rate** Q $[m^3s^{-1}]$ and **energy head** H $[m]$. Other relevant quantities can be derived from these two quantities. Volumetric flow rate is actually just a scaling of v since $Q = Av$, where A denotes the cross-sectional area of the fluid and v the average speed in A. Note that free-surface flow is not treated here and that pipelines are assumed to be always completely filled.

The energy head H is given as

$$H = \frac{p}{\rho g} + z, \tag{2.1}$$

where z denotes the height difference between the piezometric reference level and the height level at which H is measured. Formally H includes the additional term $v^2/2g$, but it is omitted here since it is usually small compared to the other quantities. Wanda internally does use the full definition for computations. Note that, since the model is only one-dimensional, p denotes the centreline pressure in the pipeline.

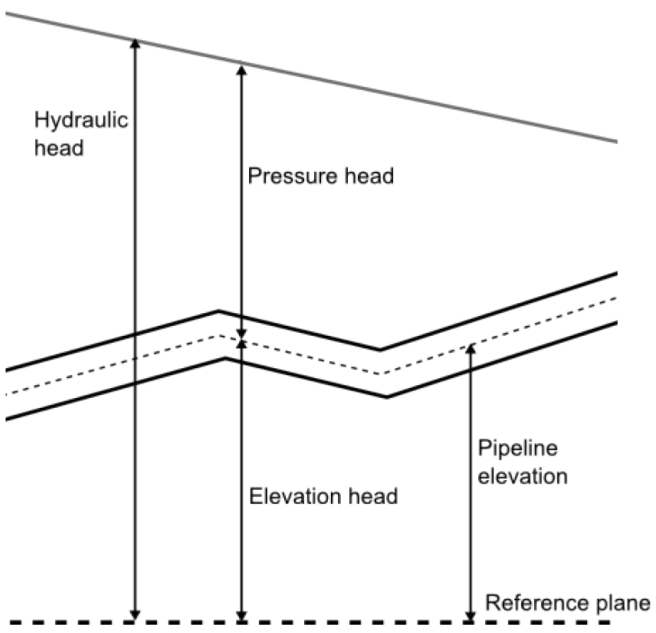


Figure 2.2: Hydraulic head.

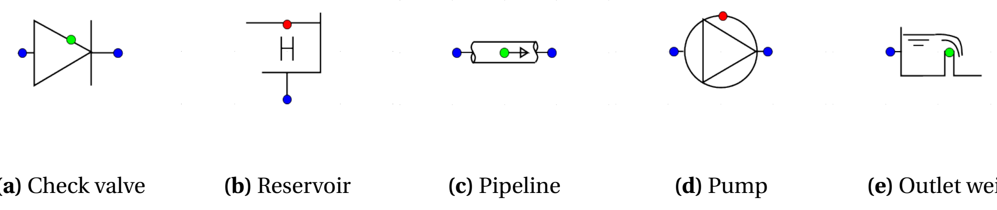
As shown in Fig. 2.2, H is the sum of pressure head $(p/\rho g)$ and elevation head (z), as measured relative to a fixed reference plane.

These two quantities are the main quantities of interest in the Wanda model.

2.3. The Wanda Model

In order to understand how Wanda models fluid dynamics in pipeline systems, it is first necessary to introduce the broad framework of how these pipeline networks are modelled. For this purpose this section starts with an example.

Example 2.1. Fig. 2.4 depicts an example of a pipeline system configured in Wanda. This system is a sewage system and consists of two pumping stations which pump the fluid from the two reservoirs B_1 and B_3 through a pipeline to the outlet wier W_1. It consists of five types of components.



(a) Check valve **(b)** Reservoir **(c)** Pipeline **(d)** Pump **(e)** Outlet weir

Figure 2.3: Sewage system components.

The components used in the sewage system are depicted in Fig. 2.3. The green coloured check valves and pumps are in operation and the red ones are out of operation. The bottom pumping station consists of a reservoir B_3 from which fluid is pumped by pump P_5 through pipeline P_3 and P_2 to outlet weir W_1. Pump P_6 is out of operation. Each pump is protected by a check valve, which prevents fluid from flowing through the pump in the wrong direction, for example, when the pump is out of operation. At the blue connection points the components are connected via 'edges', which are referred

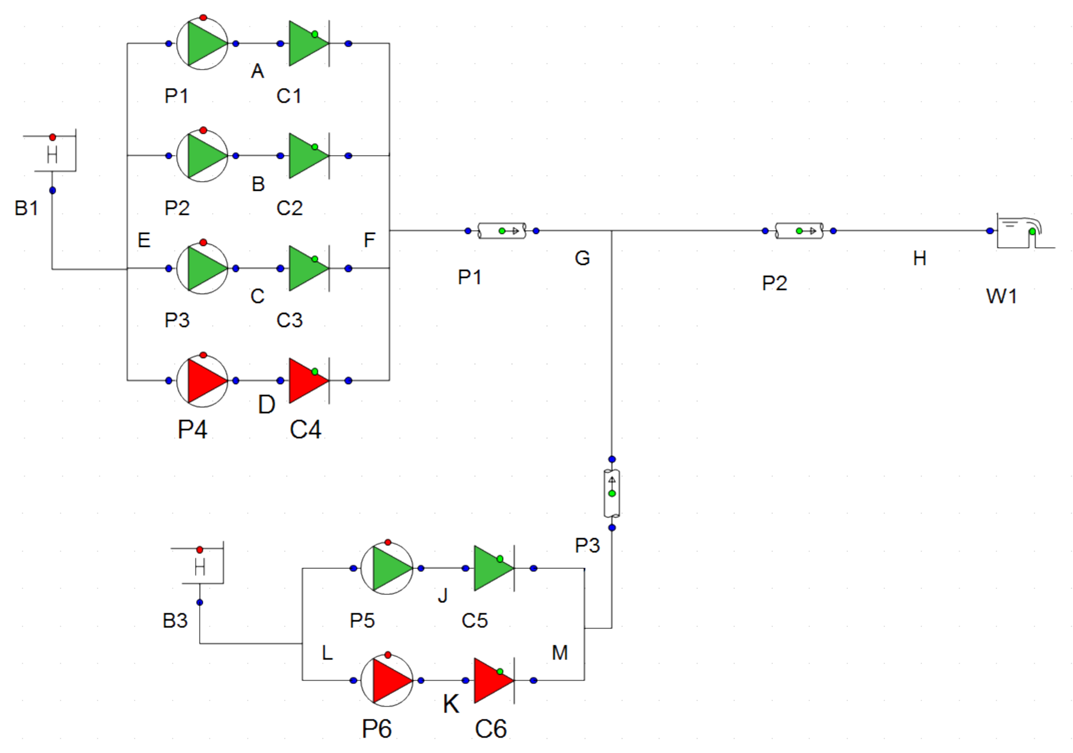


Figure 2.4: Typical sewage system.

to as hydraulic nodes, or H-nodes. These H-nodes represent physical connection pieces which have negligible influence on the flow. In Section 3.2 it will turn out that they are also helpful tools in the numerical implementation of the model. They will be treated in more detail later on. The green- and red-coloured points are the control connect points. At these points, components from the control module can be connected to the hydraulic components.

The next section introduces the basic types of components which are required to translate physical reality into a mathematical model and some additional definitions and concepts.

2.3.1. Component Types

What Wanda does is compute the unknowns H and Q in each component and in each H-node in the network. The components in Fig. 2.3 show that they have either one or two connection points with the rest of the network. This marks the division between fall type and supplier type components.

Fall Type Components

Fall type components are components which cause head loss in the fluid that flows through it and are characterised by having two connection points.

Example 2.2. The symbols used in Wanda for some of the fall type components are given in Fig. 2.5.

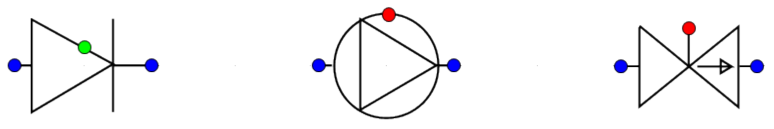


Figure 2.5: Fall type components: Check valve, pump and valve.

The fluid is assumed to be incompressible in fall type components. This assumption can be justified by the fact that in general the volume of other hydraulic components is very small compared to the volume of a pipe. Therefore, for these components, the change in flow due to compression is negligible. This entails that each fall type component can be described by a formula relating the volumetric flow rate Q_i through the component to the head loss $\Delta H = H_i - H_{i+1}$ over the length of the component, which has the general form

$$f(H_i, H_{i+1}, Q_i, t) = 0, \tag{2.2}$$

where H_i and H_{i+1} denote the head at points i and $i+1$ at the opposite ends of the component.

Example 2.3. An example of an equation describing a fall type components is the equation

$$H_i - H_{i+1} = CQ_i|Q_i|, \tag{2.3}$$

which describes the flow through an opened check valve. C can be considered to be the valve loss coefficient.

Note that H and Q are, in the transient case, time-dependent. For each fall type component Eq. (2.2) is supplemented by

$$Q_i = Q_{i+1}, \tag{2.4}$$

i.e., inflow is equal to outflow. These components cannot add or withdraw fluid from the network. Note that each fall type component adds two points (i and $i+1$), with in each point two unknowns (H and Q), and two equations to the network. In steady flow pipes too are fall type components, as fluid compressibility is no concern, and are described by the Darcy-Weisbach equation

$$\Delta H = \frac{\lambda L}{8A/O}\frac{Q_i|Q_i|}{gA^2}, \tag{2.5}$$

where λ denotes the friction coefficient, L the length of the pipe and O the pipe's cross-sectional perimeter [28]. How pipes are handled in transient flow will be described in Section 2.5.

Supplier Type Components

Supplier type components are components which add fluid to or withdraw fluid from the network and are characterised by having only one connection point.

Example 2.4. The reservoir, surge tower and vent are instances of supplier type components.

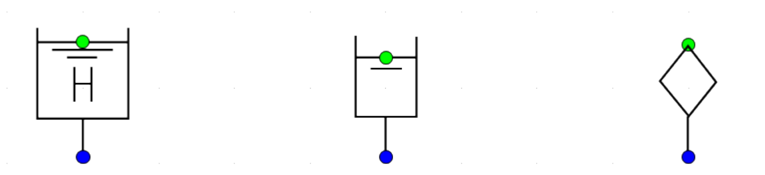


Figure 2.6: Supplier type components: Reservoir, surge tower and vent.

Supplier type components are described by a relation between the head level H_i and the in- or outgoing flow rate Q_i at the connection point. These equations have the general form

$$g(H_i, Q_i) = 0 \tag{2.6}$$

In Wanda, reservoirs can be of either finite or infinite size and can be used as a boundary condition to prescribe the H or Q at some point.

Example 2.5. The reservoir of infinite size is described by

$$H_i = c_i, \tag{2.7}$$

where $c_i \in \mathbb{R}$ is constant.

In fact, each supplier type component acts as a boundary condition; they allow fluid flow into or out of the network. Note that each supplier type component provides one point, with two unknowns, and one equation.

Component Phases

It is important to note that components can have different **phases**, that is, states. This entails that components may be described by different equations at different points in time.

Example 2.6. For a valve V Eq. (2.2) takes the following form

$$f_V(H_i, H_{i+1}, Q_i, t) = \begin{cases} H_i - H_{i+1} - d(\theta)|Q_i|Q_i, & \text{if } V \text{ is not closed at } t \\ Q_i, & \text{if } V \text{ is closed at } t \end{cases}, \tag{2.8}$$

where $d \in \mathbb{R}$ is a known variable which depends on the parameter θ which denotes how far opened the valve is.

Supplier type components such as outlets can also have different phases.

Hydraulic Nodes

In addition to these two hydraulic components, additional equations are provided by the 'edges'. As can be seen in Fig. 2.4, the components are not directly linked to each other. For example, pipe P_2 is connected to outlet W_1 via an 'edge' H. H is actually an example of a **hydraulic node (H-node)**.

Example 2.7. In this system, N_1, N_2 and N_3 are the H-nodes which connect the components.

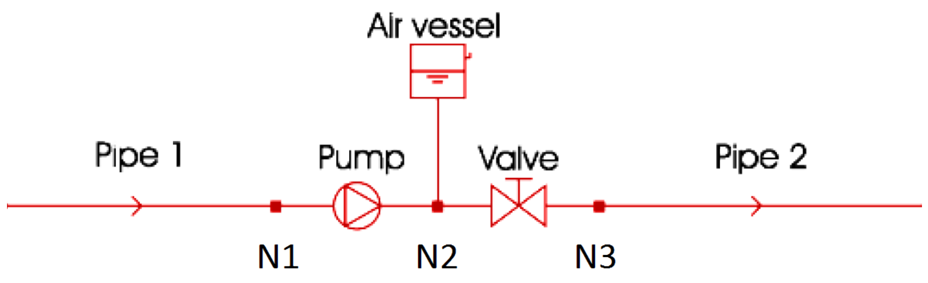


Figure 2.7: H-node example.

In principle, each H-node can be connected to an unlimited number of components. It was mentioned that fall type components only provide two equations for a total of four unknowns and supplier type components provide one equation for two unknowns. In other words, not enough equations to determine the unknowns. The other required equations are provided by the H-node(s) connected to the component and, in relation to this, each component provides one additional equation per H-node it is connected to. Each H-node N provides the equations

$$\begin{cases} Q_N + \sum_{i \in \mathcal{N}^+} Q_i - \sum_{j \in \mathcal{N}^-} Q_j = 0 \\ \qquad\qquad\qquad\qquad Q_N = 0 \end{cases}, \tag{2.9}$$

where $\mathcal{N}^+$ and $\mathcal{N}^-$ denote the sets of in- and outflow points connected to N, respectively. For each point connected to N Wanda chooses whether it is an in- or outflow point, i.e., this is just a convention and has nothing to do with the actual flow direction. These equations together enforce that the volumetric flow rates of the components connected to N are coupled, so N simply serves a middleman which connects the components, but does not of itself have any influence on the fluid flow. Furthermore, each component provides an equation of the form

$$H_i = H_N \tag{2.10}$$

for each of its end points i connected to H-node N. This simply enforces H to be equal in the points connected to N of the relevant components. Section 3.2 illustrates why adding these equations for each H-node is helpful. Note that an elevation level can be set for each H-node. This elevation level is also applied to the connected components. The elevation level of an H-node serves as a reference level for the head H in the connected components. It is easy to check that the total number of equations and unknowns for Fig. 2.7 in steady flow are both equal to 24, if it is assumed that the two points at the extremities of the pipes are not connected to any other components.

Nodal Sets

Every pipeline system is divided into **nodal sets** and pipelines. A nodal set is simply a set of non-pipe components and H-nodes in between pipes. Both ends of a pipe are connected to nodal sets, which are formed by H-nodes and components such a pumps and reservoirs, but also, for example, pipe branches or changes in pipe diameter. This concept plays a role in the solution method for transient flow in Section 2.5. The sewage system example of Fig. 2.4 consists of three pipelines and four nodal sets between pipelines. Each pumping station forms a nodal set and G with pipe junction and H are also nodal sets. As illustrated by this system, these nodal sets can contain numerous different components.

Variable Ordering

At each physical connect point and each H-node, there are two unknowns variables H_i and Q_i. The numbering of the physical connection points and H-nodes and thus the ordering of the equations and variables, is determined as follows. The starting point is the component which is added first in the Wanda user interface; from there on, a breadth-first search is done through the network where at each step an H-node and the physical connection points of the connected components are numbered. This equation and variable numbering is the ordering used by the solution method.

2.4. Steady Flow

A steady flow is, as the name suggests, a flow that does not change over time, i.e.,

$$\frac{\partial \rho}{\partial t} = \frac{\partial p}{\partial t} = \frac{\partial v}{\partial t} = 0 \tag{2.11}$$

Hence these properties can only vary with place. The fluid is assumed to be viscous, but compressibility is not taken into account. After all, if it were that case that $\rho(x_1) > \rho(x_2)$, then the fluid would tend to an equilibrium state where ρ is homogeneous, i.e., it would be unsteady. The steady flow is mainly used for designing a pipeline system as well as providing an initial value for transient flow.

As mentioned before, fluid flow in pipes is described by the Darcy-Weisbach equation

$$\Delta H = \frac{\lambda L}{8A/O} \frac{Q|Q|}{gA^2} \tag{2.12}$$

in case of steady state flow. Here $Q|Q|$ is used instead of Q^2 to be able to determine the direction of the flow. These equations coupled with equations describing the flow in components such as pumps and reservoirs lead to a system of equations that should be solved to obtain information about the characteristics of the flow (most importantly Q, H) through the network. Section 2.4.1 will describe the solution method in more detail.

2.4.1. Numerical Implementation

To summarise, in the steady flow case each fall type component provides Eq. (2.2), Eq. (2.4) and Eq. (2.10), which gives a total of four equations for the same number of unknowns. In steady flow, pipes are also considered fall type components since the fluid is incompressible. Supplier type components bring Eq. (2.6) and Eq. (2.10) to the table. Finally, each H-node provides the system of equations as given in Eq. (2.9). This results in a system of equations with the same number of equations as unknowns.

Equations are in general non-linear. The Newton-Raphson method is applied to obtain a solution to the system of equations. First the non-linear equations of the form Eq. (2.2) are linearised.

$$f(H_i^{(k+1)}, H_{i+1}^{(k+1)}, Q_i^{(k+1)}) = f(H_i^{(k)}, H_{i+1}^{(k)}, Q_i^{(k)}) + \left(\frac{\partial f}{\partial H_i}\right)^{(k)} [H_i^{(k+1)} - H_i^{(k)}] \tag{2.13}$$

$$+\left(\frac{\partial f}{\partial H_{i+1}}\right)^{(k)}[H_{i+1}^{(k+1)}-H_{i+1}^{(k)}]+\left(\frac{\partial f}{\partial Q_i}\right)^{(k)}[Q_i^{(k+1)}-Q_i^{(k)}]+\text{h.o.t.}$$

Here k denotes the iteration number. Equations of the form Eq. (2.6) are linearised in a similar manner, if necessary. The higher order terms are ignored, resulting in quadratic convergence behaviour. Now, by setting

$$f(H_i^{(k+1)},H_{i+1}^{(k+1)},Q_i^{(k+1)})=0, \tag{2.14}$$

an iterative procedure is obtained. For the whole system of equations this procedure can be written as

$$\mathbf{f}(\mathbf{u}^{(k+1)})\approx\mathbf{f}(\mathbf{u}^{(k)})+\mathbf{J}(\mathbf{u}^{(k)})\left[\mathbf{u}^{(k+1)}-\mathbf{u}^{(k)}\right]=0 \tag{2.15}$$

where $\mathbf{u}^{(k)}=[Q_1^{(k)}\ H_1^{(k)}\ \dots\ Q_n^{(k)}\ H_n^{(k)}]^\top$ and $\mathbf{J}(\mathbf{u}^{(k)})$ denotes the Jacobian matrix. Using some initial guess, the iterative process is continued until for each pipe P_m and each point j the condition

$$\begin{cases} H_j^{(k+1)}-H_j^{(k)}<\varepsilon_1 \\ Q_m^{(k+1)}-Q_m^{(k)}<\varepsilon_2 \end{cases} \tag{2.16}$$

holds for chosen $\varepsilon_1,\varepsilon_2\in\mathbb{R}$.

Example 2.8. Consider the following pipeline system.

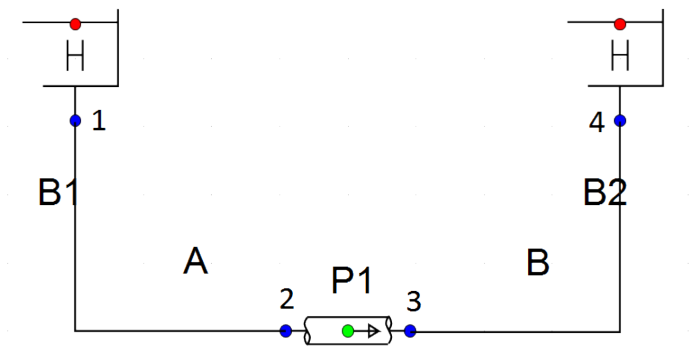


Figure 2.8: Small pipeline system.

The small pipeline system given in Fig. 2.8 leads to the following system of equations.

$$\begin{cases} H_1 &= c_1 \\ Q_A+Q_1-Q_2 &= 0 \\ Q_A &= 0 \\ H_A &= H_1 \\ H_A &= H_2 \\ H_2-H_3 &= \frac{\lambda L}{8A/O}\frac{Q_2|Q_2|}{A^2 g} \\ Q_2 &= Q_3 \\ H_B &= H_3 \\ H_B &= H_4 \\ Q_B &= 0 \\ Q_B+Q_3+Q_4 &= 0 \\ H_4 &= c_2 \end{cases} \tag{2.17}$$

After linearisation, this system can be written in matrix form as

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -c_3 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Q_1 \\ H_1 \\ Q_A \\ H_A \\ Q_2 \\ H_2 \\ Q_3 \\ H_3 \\ Q_B \\ H_B \\ Q_4 \\ H_4 \end{bmatrix} = \begin{bmatrix} c_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ c_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ c_2 \end{bmatrix}, \tag{2.18}$$

where c_3, c_4 can be determined using Eq. (2.13).

In general, each system can be written as

$$M\mathbf{u} = \mathbf{b}, \tag{2.19}$$

where $M \in \mathbb{R}^{n\times n}$, $\mathbf{b}, \mathbf{u} \in \mathbb{R}^n$ and n denotes the number of unknowns. The matrix and right hand side vector change each iteration of the Newton-Raphson method. The matrices are generally asymmetric, sparse and banded. Note that a (unique) solution need not exist, even though the number of unknowns is equal to the number of equations. If in Eq. (2.17) $Q_1 = Q_4 = 0$ were prescribed on the boundaries instead of H_1 and H_4, the equation

$$H_2 - H_3 = \frac{\lambda L}{8A/O}\frac{Q_2|Q_2|}{A^2 g} \tag{2.20}$$

has an infinite number of solutions. So at least one point with prescribed H is required for a unique solution to exist. Even then, as will be shown in Section 3.1, a unique solution need not exist for some pipeline systems.

2.5. Transient Flow

Transient flow can change in place and time. As mentioned before, the only components in which compressibility is taken into account are pipes. This allows transient flow to incorporate the phenomena cavitation and water hammer, which are caused by pressure waves in the network. Changes in v in a network lead to changes in pressure, which propagate through the network as pressure waves. If, for example, a valve suddenly closes, an overpressure wave will propagate upstream of the valve through the network. The momentum of the fluid will cause it to be compressed at the upstream side and decompressed at the downstream side of the valve. Compressibility of the fluid entails that the fluid density can vary throughout the pipe, hence the pipe's internal points need to be simulated as well. Note that in Wanda the cavitation is assumed to stay at the same place, whereas in reality it is possible that cavitation moves along with the flow. This assumption results in a much simpler model, while staying accurate enough to be useful.

In transient flow, the fluid flow in pipes is described by two conservation laws. First there is the **conservation of momentum**.

$$\frac{\partial v}{\partial t} + g\frac{\partial H}{\partial x} + \frac{\lambda}{8A/O}v|v| = 0 \tag{2.21}$$

The friction term results from the Darcy-Weisbach equation. This equation is derived from Eq. (A.1). The full derivation can be found in [25] and [28].

Secondly, the law of **conservation of mass** applies, as given by

$$\frac{\partial v}{\partial x} + \frac{g}{c^2}\frac{\partial H}{\partial t} = 0, \tag{2.22}$$

where

$$c = \frac{1}{\sqrt{\rho\left(\frac{1}{K} + \frac{1}{A}\frac{\mathrm{d}A}{\mathrm{d}p} + \frac{1}{\Delta x}\frac{\mathrm{d}\Delta x}{\mathrm{d}p}\right)}}, \tag{2.23}$$

denotes the (pressure) wave propagation speed. It is derived from the equation Eq. (A.2). One of the assumptions in the derivation is that changes in ρ have little effect on flow compared to changes in A and Δx (i.e. the element volume), hence ρ is assumed to be constant. Change in Δx means that the pipe either stretches or shrinks in length due to pressure. Changes in A mark the expansion or contraction of a cross-section of the pipe. In case of overpressure in stiff, thick-walled pipes and in case of under-pressure in all pipes, the changes in A and Δx are linearly dependent on the change in p. In other words, in these cases $\mathrm{d}A/\mathrm{d}p$ and $\mathrm{d}\Delta x/\mathrm{d}p$ are constant and hence c is constant. In general though, c will be a non-linear function of p. For the full derivation of the equation, the reader is referred to [4] or [28].

The system of Eqs. (2.21) and (2.22) consists of two equations. Note that Q can easily be obtained from these equations via the identity $Q = Av$. The quantities in c are known as ρ is constant and change in volume to pressure is a known property of the pipeline, hence the two unknowns H and v (or Q) can, in principle, be solved from this system. The solution method will be explained in the next section.

2.5.1. Numerical Implementation

The only difference between steady flow and transient flow is that in transient flow the fluid is compressible in pipes. This may seem like a small difference, but it actually has a big impact on the model. Now the fluid flow in each pipe is described by the system of equations

$$\begin{cases} \dfrac{\partial v}{\partial t} + g\dfrac{\partial H}{\partial x} + \dfrac{\lambda}{8A/O} v|v| = 0 \\ \dfrac{\partial H}{\partial t} + \dfrac{c^2}{g}\dfrac{\partial v}{\partial x} = 0 \end{cases}. \tag{2.24}$$

This system of equations will be transformed into one ordinary differential equation to which the method of characteristics is applied. Consider the linear combination

$$\frac{\partial v}{\partial t} + g\frac{\partial H}{\partial x} + \frac{\lambda}{8A/O} v|v| + \beta\left(\frac{\partial H}{\partial t} + \frac{c^2}{g}\frac{\partial v}{\partial x}\right) = 0, \tag{2.25}$$

for $\beta \in \mathbb{R}$. Any two distinct values of β again yield a system of equations which is equivalent to Eq. (2.24). The goal is to find two specific values for which Eq. (2.25) can be simplified. Rearrangement of the terms yields

$$\left(\frac{\partial v}{\partial t} + \beta\frac{c^2}{g}\frac{\partial v}{\partial x}\right) + \beta\left(\frac{\partial H}{\partial t} + \frac{g}{\beta}\frac{\partial H}{\partial x}\right) + \frac{\lambda}{8A/O} v|v| = 0 \tag{2.26}$$

The total derivatives for H and v, assuming $x = x(t)$, are given by

$$\frac{\mathrm{d}H}{\mathrm{d}t} = \frac{\partial H}{\partial t} + \frac{\partial H}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} \tag{2.27}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\partial v}{\partial t} + \frac{\partial v}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} \tag{2.28}$$

Now, by comparing these total derivatives with the expressions between brackets in Eq. (2.26), it is observed that if

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \beta\frac{c^2}{g} = \frac{g}{\beta} \tag{2.29}$$

Eq. (2.26) becomes

$$\beta\frac{\mathrm{d}H}{\mathrm{d}t} + \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\lambda}{8A/O} v|v| = 0, \tag{2.30}$$

an ordinary differential equation. Eq. (2.29) yields the two solutions

$$\beta = \pm\frac{g}{c} \tag{2.31}$$

for β. By substituting these solutions into Eq. (2.29), the relation

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \pm c \tag{2.32}$$

is obtained. Substituting these values into Eq. (2.30) results in the following two systems of equations.

$$\left.\begin{aligned} &\frac{g}{c}\frac{\mathrm{d}H}{\mathrm{d}t} + \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\lambda}{8A/O}v|v| = 0 \\ &\frac{\mathrm{d}x}{\mathrm{d}t} = c \end{aligned}\right\} C^+ \tag{2.33}$$

$$\left.\begin{aligned} &-\frac{g}{c}\frac{\mathrm{d}H}{\mathrm{d}t} + \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\lambda}{8A/O}v|v| = 0 \\ &\frac{\mathrm{d}x}{\mathrm{d}t} = -c \end{aligned}\right\} C^- \tag{2.34}$$

Consider a fixed point x' in a pipe. The equations C^+ and C^- describe that the H and v in x' are determined by changes in H and v from the points to the left and right of x', respectively. These changes travel as pressure waves with propagation speed $\pm c$ through the pipe.

A finite difference approach will be used to solve the equations above. The pipeline is discretised into parts of equal size Δx. The resulting internal pipeline nodes are called **water hammer nodes (W-nodes)**. The time-step is given as $\Delta t = \Delta x / c$.

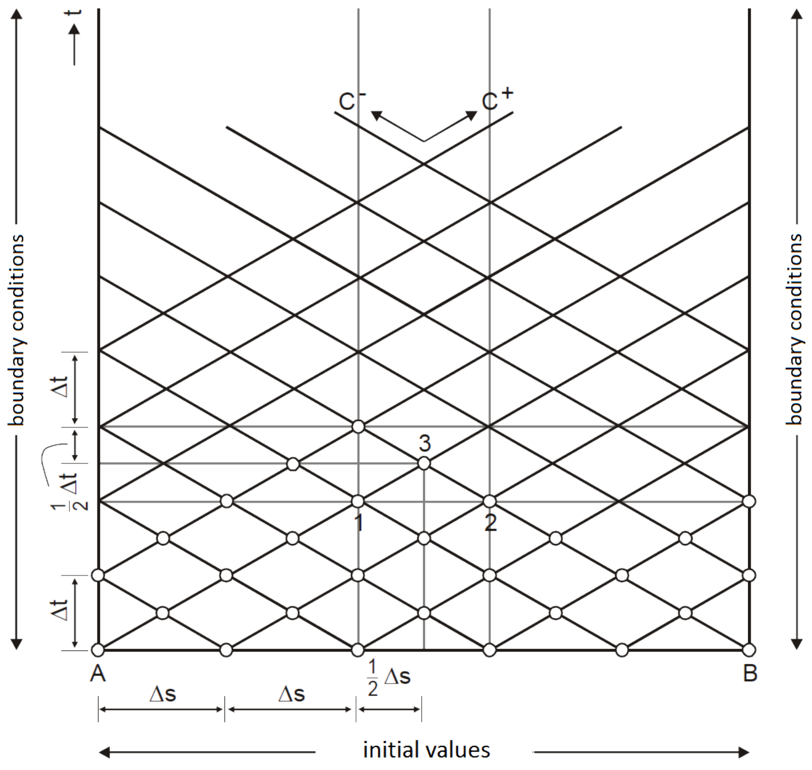


Figure 2.9: The x-t plane for constant c.

If c is constant, the x-t grid of a pipeline can be visualised as in Fig. 2.9. Note that the image uses s instead of x to denote place. Starting at $t = 0$, the Q and H in the pipeline are given by initial values which are obtained by computing the steady flow. The diagonal lines have slope c and $-c$. Along the line between P_1 and P_3 with slope c, the first equation of Eq. (2.33) holds, so integrating that equation from P_1 to P_3 results in an equation which can be solved for the unknowns in P_3.

$$\frac{g}{c}\int_{P_1}^{P_3}\frac{\mathrm{d}H}{\mathrm{d}t}\mathrm{d}t+\int_{P_1}^{P_3}\frac{\mathrm{d}v}{\mathrm{d}t}\mathrm{d}t+\int_{P_1}^{P_3}\frac{\lambda}{8A/O}v|v|\mathrm{d}t=0 \tag{2.35}$$

The first two terms can directly be integrated. The third term, however, is problematic since v along the characteristics is unknown. To solve this issue, a first order approximation for v is used, namely, simply the value of v in point P_1 will be used. Now, by denoting H_i and v_i as H and v in point P_i, respectively, the following equation is obtained.

$$H_3+\frac{c}{g}v_3-H_1-\frac{c}{g}v_1=-\frac{c}{g}\frac{\lambda}{8A/O}v_1|v_1|\frac{\Delta t}{2} \tag{2.36}$$

Similarly, along the line from point P_2 to P_3, Eq. (2.34) holds, which can be integrated from P_2 to P_3 and results in an additional equation for solving the two unknowns in P_3.

$$H_3-\frac{c}{g}v_3-H_2+\frac{c}{g}v_2=\frac{c}{g}\frac{\lambda}{8A/O}v_2|v_2|\frac{\Delta t}{2} \tag{2.37}$$

From these two equations, H_3 and v_3 at the next time step $t+\Delta t/2$ can be computed. By iterating over the grid points at each time step, a solution is obtained by solving the system of two equations given as above at each grid point.

Fig. 2.9 shows that, after calculating the state at $t+\Delta t/2$, there is a problem with calculating the unknowns at time $t+\Delta t$ for boundary points. Only one equation is available for these points. For a point at the left boundary of the plane the first equation of C^- will be added to the equations describing the nodal set connected to the left side of the pipe, similarly, the first equation of C^+ will be added to the equations describing the nodal set at the right side. The systems of equations describing the nodal sets, supplemented by the equations of the end points of the pipelines, are solved using the Newton-Raphson method. At every time step t this leads to a system of linear equations which can be written as

$$M(t)\mathbf{u}(t)=\mathbf{b}(t) \tag{2.38}$$

This matrix now has a block structure. For every nodal set S, $M(t)$ includes a block $M\_S(t)$ of size $2k\times 2k$, where k denotes the sum of the number of points in S and the number of pipe end points connected to S. Each block is linearly independent from each other block, since they are separated by pipes. These blocks, and the matrix itself, are again asymmetric, banded and sparse.

To summarise, the solution method for transient flow requires the following steps to calculate the unknowns at time step $t+\Delta t$ from the previous solution at time t.

1. First H and v for the W-nodes in each pipe will be calculated at time $t+\Delta t/2$ using the C^+ and C^- equations.

2. Next, the unknowns for the internal W-nodes in each pipe will be calculated at time $t+\Delta t$, based on the solution at time $t+\Delta t/2$, again using the C^+ and C^- equations. The end points of the pipes cannot be calculated in this step.

3. Now the solution at $t+\Delta t$ for the components in the nodal sets as well as the points at the extremities of a pipe will be calculated. The equations governing the components in the nodal sets will be supplemented by a C^+ or C^- equation for the boundary point(s) of the pipe(s) connected to the relevant nodal set.

In case c is not constant, obtaining a solution requires a bit more work, but the main idea remains the same. The solution method used in this case is simply an adjusted method of characteristics. A detailed treatment can be found in [28].

3 Problem Statement

The matrices produced by Wanda for solving steady or unsteady state problems are usually non-singular, however, matrices can be singular. There are two situations in which this happens. The first case is when the error occurs during steady state flow simulation. The other case is singularities due to phase changes in components during transient flow simulations. Both cases will be illustrated using examples. Furthermore, it will be explained why these cases are a problem for the current solution method. Finally, test cases are introduced which form the minimal set of problems a new solution method should perform well on.

3.1. Steady Flow Singularities

Systems which cannot be solved in steady state flow are usually systems which make no sense in the physical world. These errors often occur due to user error. Two steady flow examples will be given.

3.1.1. Undetermined Q

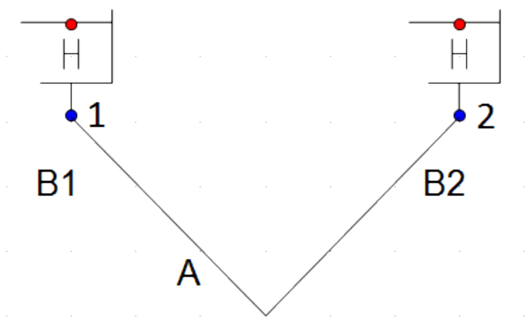


Figure 3.1: System leading to singular matrix due to user error.

Fig. 3.1 gives an example of such a system. This system consists of two boundary conditions B_1 and B_2 with prescribed H, which are connected by a hydraulic node A. It makes no physical sense, as it simply consists of two reservoirs directly connected to each other. In steady state this system leads to the following set of equations.

$$\begin{cases} H_1 &= c_1 \\ H_1 &= H_A \\ Q_A + Q_1 + Q_2 &= 0 \\ Q_A &= 0 \\ H_2 &= H_A \\ H_2 &= c_2 \end{cases} \tag{3.1}$$

In each hydraulic component and in each hydraulic node there are two unknowns Q and H, hence there are a total of six unknown variables. The system also gives rise to six linear equations. If $c_1 = c_2$ the system has an infinite number of solutions. There is no a priori preference for any particular solution. If $c_1 \neq c_2$ the system has no solution at all. In both cases the program should return an error message in which the user is notified of the mistake. The current matrix solver, however, simply crashes or gets stuck in an infinite loop on singular matrices and does not return an error message, hence a more robust matrix solver is desired.

3.1.2. Undetermined H

This example is a revisit of Example 2.8.

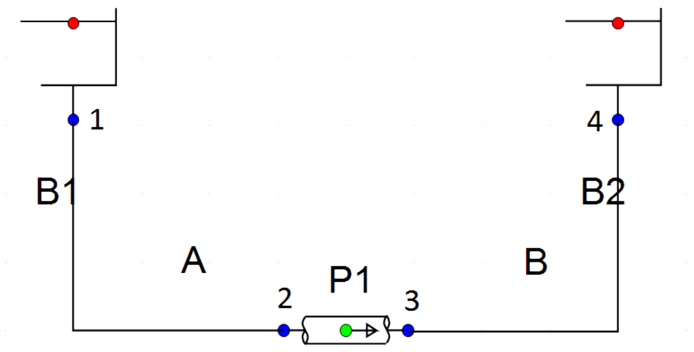


Figure 3.2: Small pipeline system.

The boundary conditions are intentionally left blank in Fig. 3.2. The system of equations is given as follows.

$$\begin{cases} f_1(H_1, Q_1) &= 0 \\ Q_A + Q_1 - Q_2 &= 0 \\ Q_A &= 0 \\ H_A &= H_1 \\ H_A &= H_2 \\ H_2 - H_3 &= \dfrac{\lambda L}{8A/O} \dfrac{Q_2|Q_2|}{A^2 g} \\ Q_2 &= Q_3 \\ H_B &= H_3 \\ H_B &= H_4 \\ Q_B &= 0 \\ Q_B + Q_3 + Q_4 &= 0 \\ f_2(H_4, Q_4) &= 0 \end{cases}, \tag{3.2}$$

where f_1 and f_2 denote the B_1 and B_2 boundary conditions, respectively. Assume the boundary conditions are chosen such that, if a flow exists, it flows from B_1 to B_2. There are now three cases to consider for the boundary conditions. Both boundary conditions prescribe H, exactly one prescribes Q and the other H, or both prescribe Q.

1. As in Eq. (2.17), both boundary conditions prescribing H results in a unique solution for the system. All the H_i's are determined by the boundary conditions and the Q_i's are determined by the Darcy-Weisbach equation.

2. If one boundary prescribes H and the other Q, the system is also uniquely determined. On one side of the pipe all H_i's are determined and at the other all Q_i's. The Darcy-Weisbach equations couples the variables on both sides.

3. There is, however, a problem with prescribing Q on both sides. Assume $Q_1 = c_1$ and $Q_4 = c_4$. Now H cannot be determined, as opposed to Q in the previous example. If $c_1 \neq -c_4$, the system has no solution at all. If $c_1 = -c_4$, it has an infinite number of solutions.

In this case Wanda asks to prescribe H on one of the H-nodes such that H becomes determined. The next section will describe in more detail how this problem is handled.

3.2. Transient Flow Singularities

A different matter is singularity due to, e.g., a closing valve or tripping pump. These actions can make (parts of) the pipeline system undetermined (without considering additional information) in transient flow. This issue should be avoided by Wanda itself as it is not due to user error. The following example illustrates how Wanda handles undetermined systems due to phase changes.

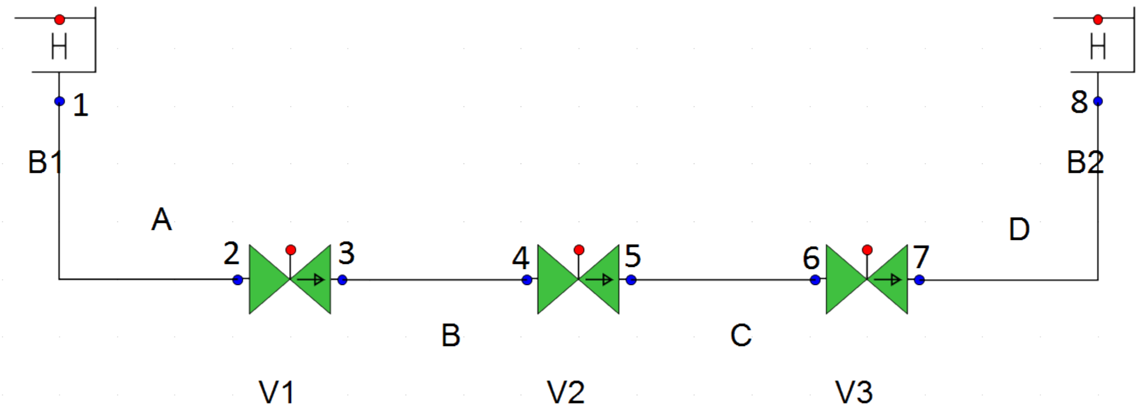


Figure 3.3: System leading to singular matrix due to phase transitions.

The pipeline system given in Fig. 3.3 consists of two supplier type components, three fall type components and four hydraulic nodes. This results in a total of twenty-four unknowns and the same number of equations. The system of equations describing the network is given by

$$\left\{\begin{aligned} H_1 &= c_1 \\ H_1 &= H_A \\ Q_A + Q_1 - Q_2 &= 0 \\ Q_A &= 0 \\ H_2 &= H_A \\ f_1(H_2, H_3, Q_2, t) &= 0 \\ Q_2 &= Q_3 \\ H_3 &= H_B \\ Q_B + Q_3 - Q_4 &= 0 \\ Q_B &= 0 \\ H_4 &= H_B \\ f_2(H_4, H_5, Q_4, t) &= 0 \\ Q_4 &= Q_5 \\ H_5 &= H_C \\ Q_C + Q_5 - Q_6 &= 0 \\ Q_C &= 0 \\ H_6 &= H_C \\ f_3(H_6, H_7, Q_6, t) &= 0 \\ Q_6 &= Q_7 \\ H_7 &= H_D \\ Q_D + Q_7 + Q_8 &= 0 \\ Q_D &= 0 \\ H_8 &= H_D \\ H_8 &= c_2 \end{aligned}\right. , \tag{3.3}$$

where

$$f_j(H_i, H_{i+1}, Q_i, t) = \begin{cases} H_i - H_{i+1} - d(\theta)|Q_i|Q_i, & \text{if } V_j \text{ is not closed at } t \\ Q_i, & \text{if } V_j \text{ is closed at } t \end{cases} \tag{3.4}$$

for a known $d(\theta)$ which depends on how far opened the valve is. Assume, if a flow exists, it flows from B_1 to B_2. Suppose that all valves are open at $t = 0$. If at most one of V_1 and V_3 is closed at the next time step $t = \Delta t$, the system is fully determined. If both are closed at $t = \Delta t$, H becomes undetermined in the part of the system between V_1 and V_3. This part of the system is described by

$$\begin{cases} Q_2 &= Q_3 \\ H_3 &= H_B \\ Q_B + Q_3 - Q_4 &= 0 \\ Q_B &= 0 \\ H_4 &= H_B \\ f_2(H_4, H_5, Q_4, t) &= 0 \\ Q_4 &= Q_5 \\ H_5 &= H_C \\ Q_C + Q_5 - Q_6 &= 0 \\ Q_C &= 0 \\ H_6 &= H_C \\ Q_6 &= Q_7 \end{cases}, \tag{3.5}$$

If V_1 and V_3 are closed, it is given that $Q_2 = Q_6 = 0$. From Eq. (3.5) it follows that

$$Q_3 = Q_B = Q_4 = Q_5 = Q_C = Q_6 = 0 \tag{3.6}$$

If V_2 is (partly) opened, it also follows that

$$H_3 = H_B = H_4 = H_5 = H_C = H_6, \tag{3.7}$$

but there is no way to determine H from the linear system. Since V_1 and V_3 were open at $t = 0$, there is information available from the previous time step. Closing these valves leads to no change in the amount of fluid in the part in between these valves, so it seems reasonable that H will stay the same as well. That is exactly what Wanda does. It replaces one of the H-node equations $Q_B = 0$ or $Q_C = 0$ with

$$H_B(t = \Delta t) = H_B(t = 0) \text{ or } H_C(t = \Delta t) = H_C(t = 0) \tag{3.8}$$

Now both Q and H are fully determined in the system. If V_2 is also closed at $t = \Delta t$, a similar thing happens, but now in both node B and C the H from the previous time step should be prescribed.

By replacing equations in this manner, Wanda can adjust singular matrices to make them non-singular. For the example in Section 3.1.2 the system becomes undetermined if both boundary conditions prescribe Q. When calculating steady state flow, Wanda will take H equal to the elevation level of the H-node. Using this information Wanda will calculate a solution, but it will also return an error that H is prescribed on a node. To obtain a suitable solution, the user is prompted to prescribe H on the H-node. If node H_B is prescribed and $c_1 \neq -c_4$, the equation

$$Q_B = 0 \tag{3.9}$$

will fail to hold. It means that an H-node does have its own flow, which makes no sense physically. Wanda returns an additional error to notify the user of this.

3.3. Current Solution Method

These examples show that an underdetermined system where H is undetermined is modified in such a way that it becomes determined.[1] The method of determining if H is undetermined is run after every phase change. It is an expensive routine, hence it could be worth optimising.

[1] Similarly, in the heat module pressure p and/or T can become undetermined and are handled appropriately.

The main problem is the case where Q is undetermined. Wanda does not have a routine in place to detect this problem in all cases. The current matrix solver sometimes detects singular matrices, but when it does not it either crashes without giving any useful feedback or it gets stuck in an infinite loop. The current solver is the `LSLXG` routine from the Fortran-based, proprietary **International Mathematics and Statistics Library (IMSL)** provided by *Rogue Wave Software*. This routine obtains an LU-decomposition using a Markowitz pivoting strategy of the matrix, which is used to solve the system of linear equations [3]. Next chapter will include more detail on how a solution is obtained using the LU-decomposition. Another drawback of the IMSL routine is that it requires a paid license. The primary goal is to find a method to detect and appropriately handle singular matrices and a matrix solver that is at least as fast as the current one without requiring paid licenses. I.e., the desire is to find and implement a robust, maintainable and fast solution method. In order to evaluate and compare solution methods test cases are required. The next section introduces test cases which are to be used as benchmarks.

3.4. Test Cases

The main goal is handling singular matrices, hence test cases resulting in singular matrices are required. The cases where H is undetermined are already handled, hence especially cases where Q is undetermined are of interest. Tests for singular matrices are given in Section 3.4.1. Since speed is also important, large, non-singular test cases are included in Section 3.4.2. All these test cases together form a minimal set of problems on which a solution method should perform well.

3.4.1. Singular Test Cases

The following table shows the test cases and their most important characteristics.

Name	Type	Undet.	H-components	n	nnz
H boundary	Steady	Q	2	6	10
Shaft	Steady	Q	3	12	21
Unsteady shaft	Transient	Q	3	12	21
Q boundary with pipe	Steady	H	3	12	23
Valve phase changes	Transient	H	5	24	49

Table 3.1: Singular test cases.

Here n denotes the number of unknowns and nnz denotes the number of non-zero elements in the matrix for the steady state flow. Due to phase changes, in transient flow nnz can change per time step, hence it is omitted from the table. For each of the test cases a visualisation and a short description follows.

H Boundary

This is the problem as discussed in Section 3.1.1.

Shaft

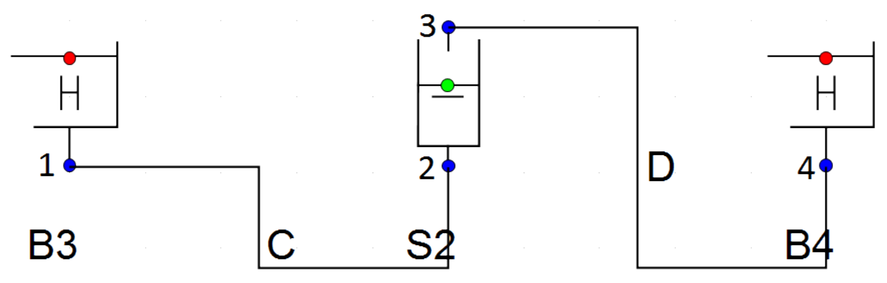


Figure 3.4: Shaft test case.

The network is given in Fig. 3.4. It consists of two boundary conditions with prescribed H and in between a shaft. It is given by the following system of equations.

$$\begin{cases} H_1 &= c_1 \\ H_C &= H_1 \\ Q_1 + Q_C + Q_2 &= 0 \\ Q_C &= 0 \\ H_C &= H_2 \\ f(H_2, H_3) &= 0 \\ Q_2 &= Q_3 \\ H_D &= H_3 \\ Q_3 + Q_D - Q_4 &= 0 \\ Q_D &= 0 \\ H_D &= H_4 \\ H_4 &= c_4 \end{cases}, \tag{3.10}$$

where

$$f(H_2, H_3) = \begin{cases} H_2 - H_3, & \text{if } S_2 \text{ is submerged} \\ H_3 - c_3, & \text{if } S_2 \text{ is partially filled} \end{cases} \tag{3.11}$$

If S_2 is submerged the upstream and downstream H are decoupled and $c_1 = c_3$ is required for a solution to exist. If S_2 is partially filled, both boundary conditions should be equal, i.e. $c_1 = c_4$, for a solution to exist. In both cases Q cannot be determined from the system.

Unsteady Shaft

The shaft example can also cause Q to become undetermined when doing transient flow simulations. Since no algorithm is in place to detect whether Q is undetermined due to phase changes, this issue should be handled appropriately by the solution method by producing a clear error message. In unsteady state the equations governing the shaft are given by

$$\begin{cases} Q_2 - Q_3 &= A\dfrac{\mathrm{d}H_2}{\mathrm{d}t} \\ f(H_2, H_3, Q_2, t) &= 0 \end{cases} \tag{3.12}$$

where

$$f(H_2, H_3, Q_2, t) = \begin{cases} H_2 - H_3, & \text{if } S_2 \text{ is submerged at } t \\ H_3 - c_3, & \text{if } S_2 \text{ is partially filled at } t \\ Q_2 & \text{if } S_2 \text{ is drained at the top at } t \end{cases} \tag{3.13}$$

If the shaft is drained at the top $t = 0$ and either submerged or filled at some time step $\Delta t > 0$, Q becomes undetermined.

Q Boundary With Pipe

This is the example as given in Section 3.1.2.

Valve Phase Changes

See Section 3.2.

3.4.2. Large Test Cases

The table below shows some large test problems with their most important characteristics.

Name	H-components	n	nnz	b_l	b_u
Drinking water	82	348	766	56	58
Noord-Holland	1543	6342	14264	308	310
Noord-Holland 2	1178	10718	23829	632	634

Table 3.2: Large test cases.

Here b_l and b_u denote the lower and upper bandwidth of the steady state matrix, respectively. The definitions of b_l and b_u are given in Section 4.1 and their significance is explained in Section 4.3.2.

Drinking Water

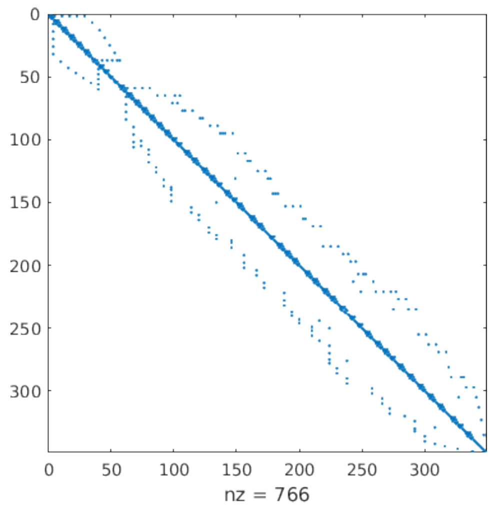
This test case represents a drinking water network with one pumping station. A visualisation is given in Fig. A.3. The sparsity pattern of the steady state matrix is given in Fig. 3.5a.

Noord-Holland 1

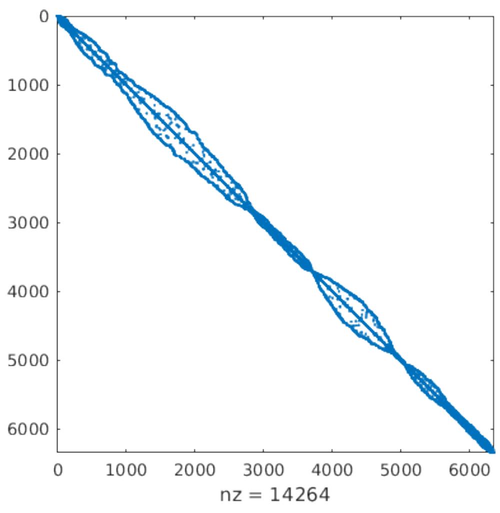
The Noord-Holland 1 test case is a large test case representing a part of the drinking water network in Noord-Holland. A visualisation is given in Fig. A.4. The sparsity pattern of the steady state matrix is given in Fig. 3.5b.

Noord-Holland 2

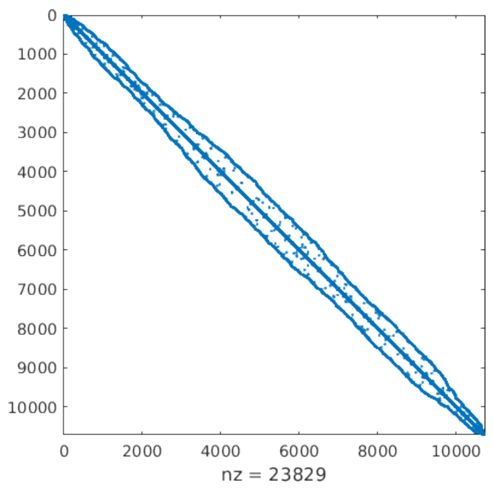
This case also represents part of the drinking water network in Noord-Holland. A visualisation is given in Fig. A.5. The sparsity pattern of the steady state matrix is given in Fig. 3.5c.



(a) Drinking water



(b) Noord-Holland 1

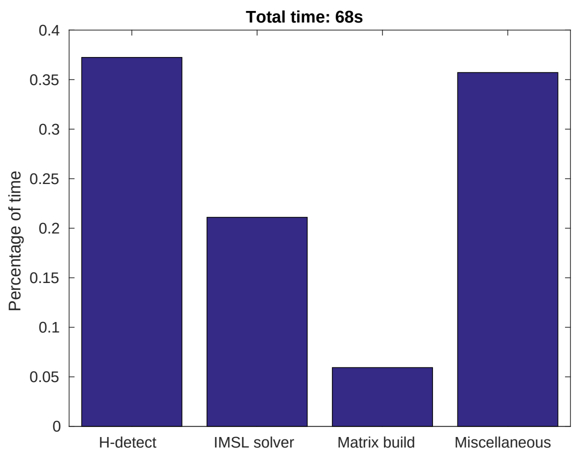


(c) Noord-Holland 2

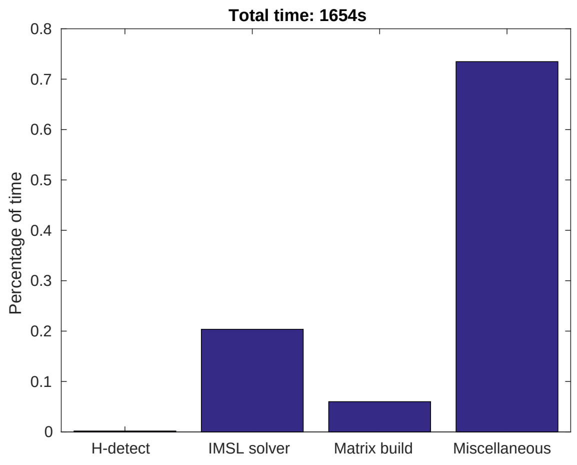
Figure 3.5: Sparsity pattern large test cases

3.5. Solution Method Profiling

Profiling results of the current solution method of Wanda for each of the large test cases are given here. The results are obtained by simulating a transient flow scenario, which differs per case. Note that these results should not be considered as final benchmarks. The results are obtained over just one run and are meant to give an indication of how long each part of the solution method takes. This information can be used to identify the bottlenecks which are worth optimising performance-wise.



(a) Noord-Holland 1



(b) Noord-Holland 2

Figure 3.6: Profiling results test cases.

Fig. 3.6 shows the percentage of computation time for the most significant parts of the solution methods. It turns out that the drinking water case is too small for any meaningful test results; the actual solution method takes an insignificant amount of time compared to all the other routines in Wanda. For this reason the results of this case are omitted here.

H-detect is the routine that searches for nodes where H is undetermined after phase changes have occurred and, if applicable, tries to fix this. For the Noord-Holland 1 case, this routine takes a significant amount of time. This indicates that there exist cases involving so many phase changes that it may be worth optimising this algorithm. For the Noord-Holland 2 case, this routine is insignificant. The IMSL solver is the part which solves the systems of linear equations. For both cases, it takes about 20% of the total computation time, which is significant. The matrix build routine is the routine which gathers all the relevant information from the components and builds the actual matrix. The miscellaneous category includes all the other routines. Most of these routines deal with the logistics necessary for carrying out the simulation and are not really part of the solution method itself. For example, the most time consuming routine in the Noord-Holland 2 test case is the routine that writes the results to a file. This routine takes more than 20% of the total computation time.

The results show that the IMSL solver and possibly the H-detect algorithm are parts of the solution method worth optimising.

4 Numerical Methods

Last chapter illustrated the problem with the current solution method. The goal of this chapter is twofold. It introduces methods which would improve the robustness of the solution method. Furthermore, methods that replace the current IMSL routine are introduced. The content of this chapter is for a significant part based on Golub and Van Loan [16].

4.1. Preliminaries

Let $M \in \mathbb{R}^{n\times n}$ and $\mathbf{b}, \mathbf{u} \in \mathbb{R}^n$. This chapter is centred around finding a solution $\mathbf{u}$ to the equation

$$M\mathbf{u} = \mathbf{b}. \tag{4.1}$$

First some preliminary concepts are introduced.

4.1.1. Sparse and Band Matrices

A matrix $M \in \mathbb{R}^{m\times n}$ is called **sparse** if only a small number of its entries are non-zero. There is no formal definition of sparsity. Sparse matrices are interesting since, compared to dense matrices, they generally require less storage space and usually less operations are required for computations involving sparse matrices.

Example 4.1. The IMSL[a] and MUMPS[b] numerical libraries, use the **coordinate format** for storing matrices [3, 5]. For example, the matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 3 \\ 0 & 4 & 5 \end{bmatrix} \tag{4.2}$$

is stored as

$$\begin{array}{r|ccccc} \text{i} & 1 & 2 & 2 & 3 & 3 \\ \text{j} & 2 & 1 & 3 & 2 & 3 \\ \text{value} & 1 & 2 & 3 & 4 & 5 \end{array} \tag{4.3}$$

including the dimension n and $\text{nnz}(M)$, which equal 3 and 5, respectively. For this example the storage is ordered on row and column, but for both the MUMPS and IMSL solvers this is not necessary.

[a] See Section 3.3.
[b] See Section 4.6.

A **band matrix** is a special type of sparse matrix. The upper and lower bandwidth of a matrix is defined as follows [16].

Definition 4.2. Let $M \in \mathbb{R}^{n\times n}$. The **lower bandwidth** b_l of M is given as the minimum number such that $M_{ij} = 0$ whenever $i - j > b_l$. Similarly, the **upper bandwidth** b_u is the minimum number such that $M_{ij} = 0$ whenever $j - i > b_u$.

Example 4.3. The matrix

$$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 0 & 7 \end{bmatrix} \tag{4.4}$$

has lower bandwidth 2 and upper bandwidth 1.

A nice property of band matrices is that only the values within the band need to be stored. Let $M \in \mathbb{R}^{m\times n}$. The transformation

$$\begin{aligned} i &\mapsto b_u + i - j + 1, \quad && \max(1, j - b_u) \le i \le \min(1, j + b_l) \\ j &\mapsto j, && 1 \le j \le n \end{aligned} \tag{4.5}$$

defines the **band matrix storage**, such that M can be stored in $M_b \in \mathbb{R}^{(b_l+b_u+1)\times n}$. This storage format is, for example, used in LAPACK[1] routines for band matrices [9].

Example 4.4. The matrix

$$\begin{bmatrix} M_{11} & M_{12} & 0 & 0 & 0 \\ M_{21} & M_{22} & M_{23} & 0 & 0 \\ M_{31} & M_{32} & M_{33} & M_{34} & 0 \\ 0 & M_{42} & M_{43} & M_{44} & M_{45} \\ 0 & 0 & M_{53} & M_{54} & M_{55} \end{bmatrix} \tag{4.6}$$

with $b_l = 2$ and $b_u = 1$ can be stored in band matrix format as

$$\begin{bmatrix} 0 & M_{12} & M_{23} & M_{34} & M_{45} \\ M_{11} & M_{22} & M_{33} & M_{44} & M_{55} \\ M_{21} & M_{32} & M_{43} & M_{54} & 0 \\ M_{31} & M_{42} & M_{53} & 0 & 0 \end{bmatrix} \tag{4.7}$$

4.1.2. Norms

A measure of distance between, and size of matrices and vectors is required for, among other purposes, error analysis. This is formalised in the concept of vector and matrix norms. An example of a vector norm is the p-norm [16, 26].

Definition 4.5. Let $p \in [1, \infty)$ and $\mathbf{u} \in \mathbb{R}^n$. The p**-norm** of $\mathbf{u}$ is given by

$$\|\mathbf{u}\|_p = \left[\sum_{i=1}^{n} |u_i|^p\right]^{1/p}$$

The most regularly used norms are the 1-,2- and ∞-norms.

$$\|\mathbf{u}\|_1 = \sum_{i=1}^{n} |u_i| \tag{4.8}$$

$$\|\mathbf{u}\|_2 = \left(\sum_{i=1}^{n} |u_i|^2\right)^{1/2} \tag{4.9}$$

$$\|\mathbf{u}\|_\infty = \max_{1\le i\le n} |u_i| \tag{4.10}$$

Using the vector norm it is possible to define a measure of distance between matrices and a measure of size of matrices.

Definition 4.6. Let $\|\cdot\|_p$ denote the p-norm on $\mathbb{R}^n$ and $M \in \mathbb{R}^{m\times n}$. The **matrix norm** is defined by

$$\|M\|_p = \sup_{\mathbf{u}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{\|M\mathbf{u}\|_p}{\|\mathbf{u}\|_p}$$

[1] See Section 4.6.

The matrix norm has the **submultiplicative** property

$$\|M_1 M_2\|_p \leq \|M_1\|_p \|M_2\|_p \tag{4.11}$$

for $M_1 \in \mathbb{R}^{m\times k}$ and $M_2 \in \mathbb{R}^{k\times n}$. This property will turn out to be important in perturbation analysis. The matrix equivalents of Eqs. (4.8) to (4.10) for $M \in \mathbb{R}^{m\times n}$ can be computed using the following formulas.

$$\|M\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} |M_{ij}| \tag{4.12}$$

$$\|M\|_2 = \sqrt{\lambda_{\max}(M^\top M)} \tag{4.13}$$

$$\|M\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^{n} |M_{ij}| \tag{4.14}$$

Here $\lambda_{\max}(\cdot)$ denotes the largest eigenvalue of a matrix. Eqs. (4.12) and (4.14) represent the maximum absolute column and row sums, respectively.

4.1.3. Rounding Errors

Analytically, it is possible to solve Eq. (4.1) exactly. When using numerical methods it is usually not possible to obtain an exact solution, rather, the goal is to approximate the solution as well as possible and necessary. Computers use **finite precision arithmetic** where numbers are stored as **floating point numbers**, which are of the form

$$\pm 0.d_1 d_2 \ldots d_t \cdot \beta^e, \tag{4.15}$$

where $d_1 > 0$ and $0 \leq d_i < \beta$. Here $0.d_1 d_2 \ldots d_t$, β and e are called the mantissa, base and exponent respectively [27]. For numerical calculations Wanda mostly uses double precision numbers, which means that $t = 53$, $\beta = 2$ and $-1024 \leq e \leq 1023$. **Rounding errors** occur when a real number x is rounded to the nearest floating point number $fl(x)$. Now

$$fl(x) = x(1+\varepsilon), \tag{4.16}$$

where

$$|\varepsilon| \leq \frac{1}{2}\beta^{1-t} = \varepsilon_0 \tag{4.17}$$

and ε_0 is called the **machine precision**. For double precision this means

$$|\varepsilon| \leq \frac{1}{2}\beta^{1-53} \approx 10^{-16}, \tag{4.18}$$

so double precision is accurate up to about 16 decimal digits.

Each operation of adding, subtracting, multiplying and dividing two floating point numbers is called a **floating point operation**, or **flop**. Flops provide a good way to quantify the computational complexity of an algorithm.

4.1.4. Quantifying Solution Errors

Vector norms can be used to quantify the error between a solution $\mathbf{u}$ to Eq. (4.1) and its (numerical) approximation $\hat{\mathbf{u}}$. The **absolute error** is given as

$$\|\mathbf{u} - \hat{\mathbf{u}}\| \tag{4.19}$$

and the **relative error** is given as

$$\frac{\|\mathbf{u} - \hat{\mathbf{u}}\|}{\|\mathbf{u}\|} \tag{4.20}$$

Since $\mathbf{u}$ is generally unknown, a more practical way to quantify the error is via the **residual** and **relative residual**, which are defined by

$$\|\mathbf{b} - M\hat{\mathbf{u}}\| \qquad \text{and} \qquad \frac{\|\mathbf{b} - M\hat{\mathbf{u}}\|}{\|\mathbf{b}\|}, \tag{4.21}$$

respectively. Note, however, that a small residual does not necessarily mean that the absolute (and relative) error is also small.

4.2. Condition Number

The goal is to find a method to detect singular matrices. Analytically, one could use the determinant to find out whether a given matrix is singular. Due to rounding errors it is in general impossible to determine whether a given matrix is singular in finite precision. The determinant of a singular matrix in finite precision may not be exactly equal to zero, so determining whether a matrix is singular amounts to determining whether the determinant is close enough to zero. However, a small determinant does not imply singularity. On the other hand, a small determinant may appear to be zero in finite precision, but the matrix may be invertible. The following example illustrates that a small determinant does not mean that the matrix is in fact singular.

Example 4.7. Consider the matrix $a \cdot I_n \in \mathbb{R}^{n\times n}$ where I_n denotes the identity matrix [16]. Now $\det(a \cdot I_n) = a^n$. Choosing a arbitrarily small results in an arbitrarily small determinant, however, the matrix is just a scaling of the identity matrix. It is easy to scale this matrix such that the determinant becomes zero in finite precision arithmetic, while a is still a non-zero number in the same precision, certainly for large n.

It is clear that determinants are not an option for determining singularity in finite precision arithmetic. Determining when the determinant of a matrix equals zero is infeasible due to the presence of rounding errors. Alternative methods are required.

Since there is no clear way of distinghuishing singular matrices from non-singular matrices in finite precision arithmetic, matrices with 'bad properties' are often referred to as **nearly singular matrices**. Consider the following example.

Example 4.8. Consider the following equation.

$$M\mathbf{u} = \begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathbf{b} \tag{4.22}$$

Intuitively, M is nearly singular as row 1 is almost a scalar multiple of row 2. The solution is given as $\mathbf{u} = [1\ 0]^\top$. Now, if a perturbation $\mathbf{b} \to \mathbf{b} + \Delta\mathbf{b}$ of the form

$$M\mathbf{u}' = \begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} 0.01 \\ 1 \end{bmatrix} = \mathbf{b} + \Delta\mathbf{b} \tag{4.23}$$

were to occur, the solution becomes $\mathbf{u}' = [0\ 1]^\top$. Now $\|\Delta\mathbf{b}\|_2 = 0.01$, but $\|\mathbf{u}' - \mathbf{u}\|_2 = \sqrt{2}$, which illustrates that a small perturbation in $\mathbf{b}$ can result in a large perturbation in $\mathbf{u}$.

The aim is now to find out some method to determine when matrices are nearly singular.

4.2.1. Right-Hand Side Perturbation

Using the matrix and vector norms, it is possible to quantify effect of a perturbation in M or $\mathbf{b}$ on $\mathbf{u}$ [26]. First assume only $\mathbf{b}$ is affected by a perturbation of the form

$$\mathbf{b} \to \mathbf{b} + \Delta\mathbf{b}, \tag{4.24}$$

where $\|\Delta\mathbf{b}\|_p \le \delta\|\mathbf{b}\|_p$ for some $\delta > 0$. The perturbed system is solved by

$$M(\mathbf{u} + \Delta\mathbf{u}) = \mathbf{b} + \Delta\mathbf{b}, \tag{4.25}$$

hence by linearity

$$M\Delta\mathbf{u} = \Delta\mathbf{b}. \tag{4.26}$$

This implies that $\Delta\mathbf{u} = M^{-1}\Delta\mathbf{b}$ and hence by submultiplicativity

$$\|\Delta\mathbf{u}\|_p = \|M^{-1}\Delta\mathbf{b}\|_p \leq \|M^{-1}\|_p\|\Delta\mathbf{b}\|_p \tag{4.27}$$

From Eq. (4.25) it also follows by linearity that

$$\|\mathbf{b}\|_p = \|M\mathbf{u}\|_p \leq \|M\|_p\|\mathbf{u}\|_p, \tag{4.28}$$

which implies that

$$\frac{1}{\|\mathbf{u}\|_p} \leq \|M\|_p\frac{1}{\|\mathbf{b}\|_p}. \tag{4.29}$$

By combining Eqs. (4.27) and (4.29) the following bound is obtained.

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \|M\|_p\|M^{-1}\|_p\frac{\|\Delta\mathbf{b}\|_p}{\|\mathbf{b}\|_p}\| \leq \delta\|M\|_p\|M^{-1}\|_p \tag{4.30}$$

The quantity $\|M\|_p\|M^{-1}\|_p$ determines the sensitivity of $\mathbf{u}$ to a perturbation in $\mathbf{b}$. A small perturbation in $\mathbf{b}$ could potentially cause a large perturbation in $\mathbf{u}$.

4.2.2. Definition and Properties

Definition 4.9. Let $M \in \mathbb{R}^{n\times n}$. The **condition number** of M using the p-norm is defined as [16]

$$\kappa_p(M) = \|M\|_p\|M^{-1}\|_p$$

Note that by submultiplicativity

$$\kappa_p(M) = \|M\|_p\|M^{-1}\|_p \geq \|MM^{-1}\|_p = \|I_n\|_p = 1 \tag{4.31}$$

A problem involving matrix with a small condition number is called **well-conditioned** (or **stable**) and one with a large condition number is called **ill-conditioned** (or **unstable**) [16]. This is dependent on what one defines as *small* and *large*. Furthermore, it also depends on which norm is used, although any two condition numbers κ_{p_1} and κ_{p_2} are equivalent in the sense that there exist $c_1, c_2 \in \mathbb{R}$ such that for all $M \in \mathbb{R}^{n\times n}$

$$c_1\kappa_{p_1}(M) \leq \kappa_{p_2}(M) \leq c_2\kappa_{p_2}(M). \tag{4.32}$$

For a singular matrix, $\kappa_p(M) = \infty$.

Example 4.10. Consider again Example 4.7. Calculating the condition number of $a^{-1}I_n$ shows that

$$\kappa_\infty(M) = \|a\cdot I_n\|_\infty\|(a\cdot I_n)^{-1}\|_\infty = a^{-1}\|I_n\|_\infty a\|I_n\|_\infty = 1, \tag{4.33}$$

hence indeed this scaling of the identity matrix is (very) well-conditioned.

Example 4.11. Consider Example 4.8. M and M^{-1} are given as

$$\begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -100 & 1 \\ 100 & 0 \end{bmatrix}, \tag{4.34}$$

respectively. Now

$$\kappa_\infty(M) = \|M\|_\infty\|M^{-1}\|_\infty = 2\cdot 101 = 202 \tag{4.35}$$

and given the perturbation of size $\|\Delta\mathbf{b}\|_\infty/\|\mathbf{b}\|_\infty = 0.01/1 = 0.01$

$$\frac{\|\Delta\mathbf{u}\|_\infty}{\|\mathbf{u}\|_\infty} \leq 202\cdot 0.01 = 2.02, \tag{4.36}$$

which explains the large perturbation in $\mathbf{u}$ of size $\sqrt{2}$.

4.2.3. Matrix and Right-Hand Side Perturbation

Now consider a perturbation in both M and $\mathbf{b}$ of the form

$$\mathbf{b} \to \mathbf{b} + \Delta\mathbf{b} \tag{4.37}$$

$$M \to M + \Delta M \tag{4.38}$$

where $\|\Delta\mathbf{b}\|_p \leq \delta\|\mathbf{b}\|_p$ and $\|\Delta M\|_p \leq \delta\|M\|_p$ for some $\delta > 0$. Now the perturbed solution $\mathbf{v} = \mathbf{u} + \Delta\mathbf{u}$ satisfies

$$(M + \Delta M)\mathbf{v} = \mathbf{b} + \Delta\mathbf{b}. \tag{4.39}$$

Assuming that $\delta\kappa_p(M) < 1$ (which prevents $A + \Delta A$ from becoming singular), it can be shown that [16]

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \frac{2\delta}{1 - \delta\kappa_p(M)}\kappa_p(M) \tag{4.40}$$

Example 4.12. Consider a small perturbation bounded by $\delta = 0.5 \cdot 10^{-6}$ and assume $\kappa_p(M) = 10^6$. Now the relative perturbation in $\mathbf{u}$ is bounded by

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \frac{2 \cdot 0.5 \cdot 10^{-6}}{1 - 0.5 \cdot 10^{-6} 10^6} 10^6 = 2 \tag{4.41}$$

illustrating that a small perturbation in both M and $\mathbf{b}$ can cause a relatively large perturbation in $\mathbf{u}$.

4.2.4. Ambiguity

The trouble with the condition number is that there is still some ambiguity involved. There is no general rule for when $\kappa_p(M)$ is too large, i.e., for determining when M is ill-conditioned. A useful heuristic states that, if $\varepsilon_0 \approx 10^{-r}$ and $\kappa_\infty(M) \approx 10^q$, then Gaussian elimination gives a solution which is accurate up to about $q - r$ decimal digits [16]. According to this heuristic, what really determines whether M is ill-conditioned with respect to the machine precision depends on the accuracy required for the underlying problem. Ultimately, to detect (nearly) singular matrices, some arbitrary cut-off value is required to qualify matrices as (nearly) singular or not.

4.2.5. Calculating the Condition Number

The definition of the condition number immediately poses a big problem since $\|M^{-1}\|$ is required. If M^{-1} were known, it could be used to immediately solving $M\mathbf{u} = \mathbf{b}$ without having to resort to matrix solvers, however, matrix inversion is computationally expensive. The next section will show how to estimate $\kappa_p(M)$ using the LU-factorisation of M, without requiring M^{-1} to be known.

4.3. *LU*-Factorisation

Solving a system of linear equations by hand typically amounts to applying Gaussian elimination. Similarly, for small systems as the ones in Wanda, direct solution methods are used as they offer sufficient performance. Direct solution methods use Gaussian elimination techniques to solve a system of the form Eq. (4.1). Using Gaussian elimination M is factored into

$$M = LU, \tag{4.42}$$

where $L, U \in \mathbb{R}^{n\times n}$ are lower and upper triangular matrices, respectively. Solving Eq. (4.1) now amounts to solving the forward step

$$L\mathbf{y} = \mathbf{b} \tag{4.43}$$

and backward step

$$U\mathbf{u} = \mathbf{y} \tag{4.44}$$

such that

$$M\mathbf{u} = LU\mathbf{u} = L\mathbf{y} = \mathbf{b}. \tag{4.45}$$

In Section 4.3.3 it will be shown how the LU-decomposition can be used to estimate the condition number of a matrix.

Example 4.13. Consider the following system of linear equations.

$$u_1 + 2u_2 = 1 \tag{4.46}$$
$$3u_1 + 4u_2 = 1 \tag{4.47}$$

Subtracting 3 times Eq. (4.46) from Eq. (4.47) results in the triangular system

$$u_1 + 2u_2 = 1 \tag{4.48}$$
$$-2u_2 = -2 \tag{4.49}$$

which can easily be solved. Similarly, the matrix equivalent to this system can be factored into

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} = LU \tag{4.50}$$

and can be used to obtain a solution using the forward and backward steps.

The main idea behind the LU-factorisation is applying Gaussian transformations $G_1, \dots, G_{n-1}$ to M, such that it is reduced to row echelon form, i.e. $U = G_{n-1}G_{n-2}\dots G_1 M$. Applying transformation G_k will set the elements of column k of M below the diagonal to zero. It is assumed that each transformation only includes adding a scalar multiple of one row to another. No row scaling or row interchanges are applied. The matrix L contains information about what Gaussian transformations are used at each step.

Let $M \in \mathbb{R}^{n\times n}$ and let $M^{(k)}$ denote M after applying the first k Gaussian transformations, i.e.

$$M^{(k)} = G_k G_{k-1} \dots G_1 M, \qquad k = 1, \dots, n-1 \tag{4.51}$$

Furthermore, let $M^{(0)} = M$.

Definition 4.14. The k^{th} Gauss-vector $\alpha_k \in \mathbb{R}^n$ is defined as [26]

$$\boldsymbol{\alpha}^{(k)} = [\underbrace{0, \dots, 0}_{k}, \alpha^{(k)}_{k+1}, \dots, \alpha^{(k)}_{n}], \tag{4.52}$$

where $\alpha^{(k)}_i = M^{(k-1)}_{ik} / M^{(k-1)}_{kk}$. The element $M^{(k-1)}_{kk}$ is called the k^{th} **pivot element**.

The k^{th} **Gaussian transformation** G_k can now be defined as

$$G_k = I - \boldsymbol{\alpha}^{(k)} \mathbf{e}_k^\top \tag{4.53}$$

From this definition it is immediately clear that an LU-decomposition is only possible if $M^{(k-1)}_{kk} \neq 0$ for $1 \le k \le n-1$. In finite precision a bound away from 0 is required. It can be proved that the LU-factorisation of M exists if M and all its principal submatrices are non-singular [16]. A principal submatrix $M' \in \mathbb{R}^{k\times k}$ of M is a matrix that can be obtained from M by removing $n-k$ rows and the same $n-k$ columns from M. This is also enough to guarantee that the pivot elements are non-zero in finite precision. The LU-decomposition is unique.

Now define

$$U = G_{n-1}G_{n-2}\dots G_1 M. \tag{4.54}$$

It is not hard to show that

$$G_k^{-1} = I + \boldsymbol{\alpha}^{(k)} \mathbf{e}_k^\top \tag{4.55}$$

and

$$(G_{n-1}G_{n-2}\dots G_1)^{-1} = G_1^{-1}G_2^{-1}\dots G_{n-1}^{-1} = \prod_{k=1}^{n-1}\left(I+\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top\right) = I + \sum_{k=1}^{n-1}\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top \tag{4.56}$$

Finally, if L is defined as

$$L = I + \sum_{k=1}^{n-1}\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top, \tag{4.57}$$

then $M = LU$ [16].

Example 4.15. For

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}, \tag{4.58}$$

the first Gaussian transformation is given as

$$G_1 = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \tag{4.59}$$

such that

$$G_1 M = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -4 & -8 \\ 0 & 1 & -1 \end{bmatrix}. \tag{4.60}$$

Similarly,

$$G_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/4 & 1 \end{bmatrix} \tag{4.61}$$

results in

$$G_2 G_1 M = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -4 & -8 \\ 0 & 0 & -3 \end{bmatrix} = U \tag{4.62}$$

and

$$L = G_2^{-1}G_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & -1/4 & 1 \end{bmatrix} \tag{4.63}$$

The example above illustrates that, at each iteration k, the matrix $M^{(k)}$ can be partitioned into

$$M^{(k)} = \begin{bmatrix} N_{11}^{(k)} & N_{12}^{(k)} \\ 0 & N_{22}^{(k)} \end{bmatrix}, \tag{4.64}$$

where $N_{11}^{(k)} \in \mathbb{R}^{k\times k}$, $N_{12}^{(k)} \in \mathbb{R}^{k\times(n-k)}$ and $N_{22}^{(k)} \in \mathbb{R}^{(n-k)\times(n-k)}$.

4.3.1. Computing the *LU*-Factorisation

In practice, to limit the storage space required, the *LU* decomposition is usually computed in such a way that it is stored in the original matrix M. Assume M is a banded matrix with lower and upper bandwidth b_l and b_u, respectively. The following algorithm computes the *LU* decomposition of M [16].

Algorithm 4.1 *LU* Factorisation

```
for $k = 1 \to n-1$ do
    if $M(k,k) = 0$ then
        Error: zero pivot
    end if
    for $i = k+1 \to \min\{k+b_l, n\}$ do
        $M(i,k) \leftarrow M(i,k)/M(k,k)$
    end for
    for $j = k+1 \to \min\{k+b_u, n\}$ do
        for $i = k+1 \to \min\{k+b_l, n\}$ do
            $M(i,j) = M(i,j) - M(i,k)M(k,j)$
        end for
    end for
end for
```

Note that the diagonal elements of L are all equal to 1, hence they do not need to be stored. The algorithm requires about $2b_l b_u n$ flops. After factorisation, the solution to Eq. (4.1) can be obtained by solving the forward and backward steps as in Eqs. (4.43) and (4.44). The following versions of the forward and backward steps overwrite the right-hand side **b** with the solution of the substitution steps.

Algorithm 4.2 *LU* Forward Substitution Step

```
for $j = 1 \to n$ do
    for $i = j+1 \to \min\{j+b_l, n\}$ do
        $b(i) \leftarrow b(i) - L(i,j)b(j)$
    end for
end for
```

Algorithm 4.3 *LU* Backward Substitution Step

```
for $j = n \to 1$ do
    $b(j) \leftarrow b(j)/U(j,j)$
    for $i = \max\{1, j-b_u\} \to j-1$ do
        $b(i) \leftarrow b(i) - U(i,j)b(j)$
    end for
end for
```

The forward and backward substitution steps for a banded matrix cost about $2nb_l$ and $2nb_u$ flops, respectively.

4.3.2. Pivoting

The current *LU*-factorisation algorithm has two drawbacks. One is that in finite precision rounding errors can cause a large perturbation in the matrix M, i.e. the algorithm is unstable. The other is the loss of sparsity. Both these problems can be (partially) avoided by a technique called pivoting.

Pivoting for Stability

Example 4.16. LU factorisation cannot be applied to the matrix

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{4.65}$$

as it has a zero pivot, while the matrix is well-conditioned [26].

Example 4.17. Consider the following equation in $\beta = 10$, $t = 3$ floating point arithmetic [16].

$$\begin{bmatrix} 0.001 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{u} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \tag{4.66}$$

The matrix is well-conditioned as $\kappa_\infty(M) = 3$. The LU decomposition is given by

$$\widehat{L} = \begin{bmatrix} 1 & 0 \\ 1000 & 1 \end{bmatrix}, \quad \widehat{U} = \begin{bmatrix} 0.001 & 1 \\ 0 & -1000 \end{bmatrix} \tag{4.67}$$

and

$$\widehat{L}\widehat{U} = \begin{bmatrix} 0.001 & 1 \\ 1 & 0 \end{bmatrix} \tag{4.68}$$

Solving the system using this decomposition results in the solution $\hat{\mathbf{u}} = [0\ 1]^\top$, while the exact solution is given by $\mathbf{u} = [1.002\ldots\ 0.998\ldots]^\top$.

Consider the LU-decomposition of M in finite precision arithmetic. It can be shown (see [16]) that the computed $\widehat{L}$ and $\widehat{U}$ satisfy

$$\widehat{L}\widehat{U} = M + \Delta M, \tag{4.69}$$

where

$$\|\Delta M\| \leq n\varepsilon_0 \|\widehat{L}\| \|\widehat{U}\| \tag{4.70}$$

For small problems, as is the case Wanda, $n\varepsilon_0$ is small. What could be problematic is large elements in $\widehat{L}$ or $\widehat{U}$. If one of the pivot elements is very small Definition 4.14 shows that elements of L can become very large. This could result in a solution $\hat{\mathbf{u}}$ to $\widehat{L}\widehat{U}\hat{\mathbf{u}} = \mathbf{b}$ which does a bad job at solving the original equation $M\hat{\mathbf{u}} = \mathbf{b}$. In order to avoid this, Gaussian elimination in combination with a technique called **pivoting** is applied.

Popular strategies are **partial** and **complete pivoting**. In complete pivoting, prior to applying the Gaussian transformation G_k, permutation matrices P_k and Q_k are applied to $M^{(k-1)}$,

$$P_k M^{(k-1)} Q_k, \tag{4.71}$$

such that the k^{th} pivot element $(P_k M^{(k-1)} Q_k)_{kk}$ is the largest entry in absolute value in the matrix partition $N_{22}^{(k-1)}$ (see Eq. (4.64)) [16]. The matrices P_k and Q_k represent the row and column interchange necessary to achieve this, respectively. In other words, Algorithm 4.1 becomes [16]

Algorithm 4.4 *LU* Factorisation With Complete Pivoting

```
for $k = 1 \to n-1$ do
    Determine $a$ and $b$ such that $|M(a,b)|$ is the maximum element in $M(k:n, k:n)$
    $M(k,:) \leftrightarrow M(a,:)$
    $M(:,k) \leftrightarrow M(:,b)$
    if $M(k,k) = 0$ then
        Error: zero pivot
    end if
    for $i = k+1 \to \min\{k+b_l, n\}$ do
        $M(i,k) \leftarrow M(i,k)/M(k,k)$
    end for
    for $j = k+1 \to \min\{k+b_u, n\}$ do
        for $i = k+1 \to \min\{k+b_l, n\}$ do
            $M(i,j) = M(i,j) - M(i,k)M(k,j)$
        end for
    end for
end for
```

Complete pivoting thus requires the comparison of $(n-k)^2$ numbers at each iteration k. Partial pivoting is similar, but it only determines a and hence only applies the permutation P_k at each iteration. Partial pivoting requires the comparison of $n-k$ numbers at each iteration k.

The question now, of course, is if these pivoting strategies increase stability.

Definition 4.18. Let $M \in \mathbb{R}^{n\times n}$. The **growth factor** γ of the Gaussian elimination of M is defined as

$$\gamma = \frac{\max\{\sigma, \sigma_1, \dots, \sigma_{n-1}\}}{\sigma}, \tag{4.72}$$

where $\sigma = \max_{i,j} |M_{ij}|$ and $\sigma_k = \max_{i,j} |M_{ij}^{(k)}|$.

Note that $|U_{ij}| = |M_{ij}^{(n-1)}| \leq \gamma \cdot \max_{i,j} |M_{ij}|$, which motivates the definition. It can be shown that with partial pivoting

$$\widehat{L}\widehat{U} = M + \Delta M, \tag{4.73}$$

where $\|\Delta M\|_\infty \leq n^3 \varepsilon_0 \gamma \|M\|_\infty$ and ε_0 denotes the machine precision [16]. In practice, γ is usually of order 10.

Complete pivoting results in an LU-factorisation of the form

$$PAQ = LU \tag{4.74}$$

For partial pivoting $Q = I_n$, the identity matrix.

Pivoting for Sparsity

Another reason to apply pivoting is to keep LU as sparse as possible. If M has lower bandwidth of b_l and an upper bandwidth of b_u, L has a lower bandwidth of b_l and U an upper bandwidth of b_u [16]. The problem is that within their respective bandwidths, L and U usually become almost completely dense, hence more storage space is required and solving systems of linear equations using the LU-decomposition is not so efficient.

Example 4.19. Consider the discretisation matrix of the Laplacian in 2D [26].

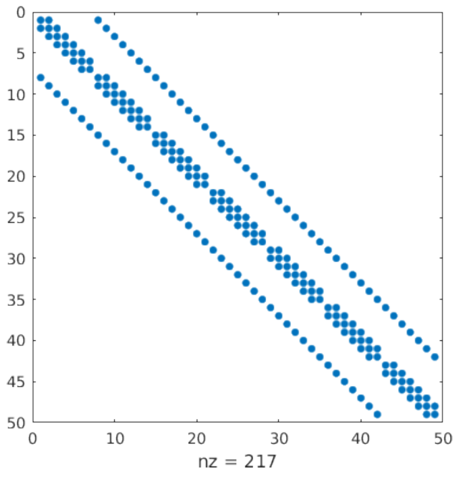
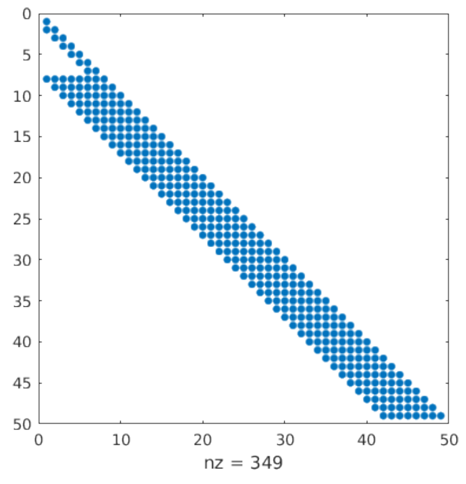
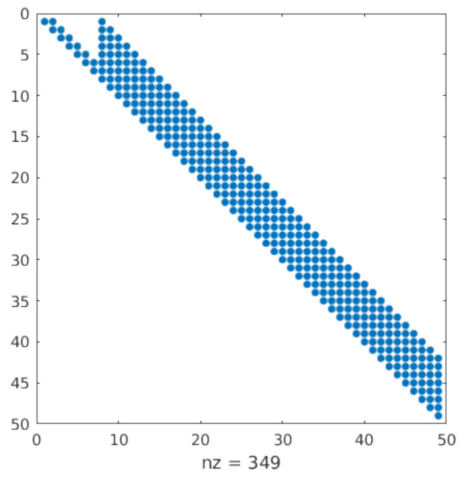


Figure 4.1: Sparsity pattern 2D Laplacian discretisation matrix

The matrix has an upper and lower bandwidth of 7, with numerous zero entries inside the band and only a total of 217 non-zero entries. The LU-decomposition is depicted below.



(a) L



(b) U

Figure 4.2: Sparsity pattern LU-decomposition 2D Laplacian discretisation

The LU-decomposition contains a lot more non-zero elements inside the band, in fact, the L and U matrices are almost completely dense within the band.

Pivoting strategies are applied to prevent this so-called **fill-in** from occurring. Note that these strategies are all heuristics.

The currently used IMSL routine `LSLXG` uses **Markowitz pivoting** [3]. Let $r_i^{(k)}$ denote the number of non-zero elements in row i of $N_{22}^{(k)}$, given as in Eq. (4.64), and let $c_j^{(k)}$ denote the number of non-zero elements of column j of the same matrix. Now compute

$$\chi_{ij}^{(k)} = (r_i^{(k)} - 1)(c_j^{(k)} - 1) \tag{4.75}$$

for each element in $N_{22}^{(k)}$. Apply row and column permutations such that the element which minimises $\chi_{ij}^{(k)}$ becomes the pivot element. In case of a tie, one can pick the largest element. During the iteration k of the LU-factorisation this pivot selection will cause $\chi_{ij}^{(k)}$ entries to be modified, which will not all result in fill-in, hence this choice is a local optimum for creating the least fill-in. In order to not get in trouble with stability, not all elements of $N_{22}^{(k)}$ are considered. For $0 < \delta < 1$ only the elements $\left(N_{22}^{(k)}\right)_{ij}$ such that

$$\left|N_{22}^{(k)}\right|_{ij} \geq \delta \left|N_{22}^{(k)}\right|_{ab} \tag{4.76}$$

for all $k \leq a, b \leq n$ are considered [21].

4.3.3. Computing the Condition Number

As mentioned in Section 4.2.5, the definition of the condition number requires $\|M^{-1}\|_p$ to be known. If M^{-1} were known, resorting to numerical methods would be unnecessary. An estimation of $\|M^{-1}\|_p$ is required. A naive estimation would be to solve $M\mathbf{u}_i = \mathbf{e}_i$ $(i = 1, \ldots, n)$, where $\mathbf{e}_i = [\underbrace{0 \ \ldots \ 0}_{i-1} \ 1 \ \underbrace{0 \ \ldots \ 0}_{n-i}]^\top$. Then

$$M[\mathbf{u}_1 \ \mathbf{u}_2 \ \ldots \ \mathbf{u}_n] = I_n \tag{4.77}$$

and hence the matrix $[\mathbf{u}_1 \ \mathbf{u}_2 \ldots \ \mathbf{u}_n]$ would provide a good estimate for M^{-1} [16]. The problem with this approach is that it would require solving n systems of linear equations. The goal is to compute an estimate of $\|M^{-1}\|_p$ in $\mathcal{O}(n^2)$ flops. This section is restricted to computing an estimation of $\|M^{-1}\|_\infty$ as proposed in [13]. This method is based on the observation that

$$M\mathbf{u} = \mathbf{b} \qquad \Longrightarrow \qquad \|M^{-1}\|_\infty \geq \|\mathbf{u}\|_\infty / \|\mathbf{b}\|_\infty \tag{4.78}$$

This inequality states that $\|\mathbf{u}\|_\infty / \|\mathbf{b}\|_\infty$ provides a lower bound on $\|M^{-1}\|_\infty$. The method provides a heuristic that tries to maximise $\|\mathbf{u}\|_\infty / \|\mathbf{b}\|_\infty$ in order to estimate $\|M^{-1}\|_\infty$. The final goal is to use the LU-decomposition for condition number estimation.

Let $T \in \mathbb{R}^{n\times n}$ be upper triangular and consider the following column version of solving $T\mathbf{y} = \mathbf{d}$ using backward substitution [16].

Algorithm 4.5 Column Version Backward Substitution

```
$p(1:n) = 0$
for $k = n \to 1$ do
    Choose $d(k)$
    $y(k) \leftarrow [d(k) - p(k)] / T(k,k)$
    $p(1:k-1) \leftarrow p(1:k-1) + T(1:k-1,k)y(k)$
end for
```

This algorithm does not use the usual backward substitution method, but it uses an auxiliary vector $\mathbf{p}$ to calculate the element $y(k)$ at each step. One way to heuristically maximise $\|\mathbf{y}\|_\infty / \|\mathbf{d}\|_\infty$ is to choose $d(k) \in \{-1,1\}$. This ensures that $\|\mathbf{d}\|_\infty = 1$, hence from Eq. (4.78) it follows that $\|\mathbf{y}\|_\infty$ provides the estimation for $\|T^{-1}\|_\infty$. This way of choosing $\mathbf{d}$ can be applied in such a way that both $y(k)$ and $p(1:k-1)$ grow at each iteration. Algorithm 4.6 is a version of Algorithm 4.5 that implements this heuristic [16].

Algorithm 4.6 Triangular Condition Estimation

```
$p(1:n) \leftarrow 0$
for $k = n \to 1$ do
    Choose $d(k) = 1$
    $y(k)^+ \leftarrow [d(k) - p(k)] / T(k,k)$
    $p(k)^+ \leftarrow p(1:k-1) + T(1:k-1,k)y(k)^+$
    Choose $d(k) = -1$
    $y(k)^- \leftarrow [d(k) - p(k)] / T(k,k)$
    $p(k)^- \leftarrow p(1:k-1) + T(1:k-1,k)y(k)^-$
    if $|y(k)^+| + \|p(k)^+\|_1 \geq |y(k)^-| + \|p(k)^-\|_1$ then
        $y(k) \leftarrow y(k)^+$
        $p(1:k-1) \leftarrow p(1:k-1)^+$
    else
        $y(k) \leftarrow y(k)^-$
        $p(1:k-1) \leftarrow p(1:k-1)^-$
    end if
end for
$\mathbf{y} \leftarrow \mathbf{y} / \|\mathbf{y}\|_\infty$
```

This algorithms considers both options $d(k) = 1$ and $d(k) = -1$ and uses the one which results in the most growth in $y(k)$ and $p(k)$. The heuristic chooses the local optimum at each iteration which hopefully approaches the global optimum.

It turns out that the lower bound that the estimate $\|\mathbf{y}\|_\infty / \|\mathbf{d}\|_\infty$ provides for $\|T^{-1}\|_\infty$ can be made even sharper [13, 17]. This can be done using the following steps.

1. Apply the lower triangular version of Algorithm 4.6 to $T^\top \mathbf{y} = \mathbf{d}$.

2. Solve $T\mathbf{x} = \mathbf{y}$.

3. Estimate $\|T^{-1}\|_\infty$ by $\|\mathbf{x}\|_\infty / \|\mathbf{y}\|_\infty$.

The motivation for step 2 is that a singular value decomposition analysis shows that if $\|\mathbf{y}\|_\infty / \|\mathbf{d}\|_\infty$ is large then $\|\mathbf{x}\|_\infty / \|\mathbf{y}\|_\infty$ is almost certainly at least as large and often produces an even better estimate [13].

Consider again the general matrix M with $PM = LU$ and assume for simplicity that $P = I_n$. Similar to the argument above, producing a large-norm solution to $(LU)^\top \mathbf{r} = \mathbf{d}$ and solving $LU\mathbf{z} = \mathbf{r}$ produces a sharp estimate for $\|M^{-1}\|_\infty$, namely $\|\mathbf{z}\|_\infty / \|\mathbf{r}\|_\infty$. A slight adjustment to the procedure is required as Algorithm 4.6 can only be applied to triangular matrices. This motivates the following procedure.

1. Apply the lower triangular version of Algorithm 4.6 to $U^\top \mathbf{y} = \mathbf{d}$.
2. Solve $L^\top \mathbf{r} = \mathbf{y}$.
3. Solve $L\mathbf{w} = \mathbf{r}$.
4. Solve $U\mathbf{z} = \mathbf{w}$.
5. Estimate $\|M^{-1}\|_\infty$ by $\|\mathbf{z}\|_\infty / \|\mathbf{r}\|_\infty$.

Step 1 and 2 produce a large-norm solution $\mathbf{r}$ and estimate $\|\mathbf{r}\|_\infty / \|\mathbf{d}\|_\infty$. Step 3 and 4 produce the sharper estimate $\|\mathbf{z}\|_\infty / \|\mathbf{r}\|_\infty$.

Note that other estimations techniques are also available, see, e.g., [17].

4.3.4. Iterative Refinement

Assume the solution $\hat{\mathbf{u}}$ to Eq. (4.1) is obtained using the factorisation $PM \approx \widehat{L}\widehat{U}$ in finite precision arithmetic. The goal is to improve the accuracy of the solution $\mathbf{u}$.

Algorithm 4.7 Iterative Refinement [16]

```
$\mathbf{r} \leftarrow \mathbf{b} - M\hat{\mathbf{u}}$
Solve $L\mathbf{y} = P\mathbf{r}$
Solve $U\mathbf{z} = \mathbf{y}$
$\mathbf{u}_{\text{new}} \leftarrow \hat{\mathbf{u}} + \mathbf{z}$
```

Applying Algorithm 4.7 would result in

$$M\mathbf{u}_{\text{new}} = M\hat{\mathbf{u}} + M\mathbf{z} = \mathbf{b} - \mathbf{r} + \mathbf{r} = \mathbf{b} \tag{4.79}$$

in exact arithmetic. In finite precision, however, things are a little more complicated. Assume for simplicity that $P = I_n$ and

$$M = \widehat{L}\widehat{U} - E \tag{4.80}$$

Now the identity

$$M\mathbf{u} = \widehat{L}\widehat{U}\mathbf{u} - E\mathbf{u} = \mathbf{b} \tag{4.81}$$

motivates the fixed-point iteration

$$\widehat{L}\widehat{U}\mathbf{u}_{\text{new}} - E\hat{\mathbf{u}} = \mathbf{b} \tag{4.82}$$

which can be rewritten as

$$\widehat{L}\widehat{U}\mathbf{u}_{\text{new}} = (\widehat{L}\widehat{U} - M)\hat{\mathbf{u}} + \mathbf{b} \tag{4.83}$$

$$\iff \mathbf{u}_{\text{new}} = \hat{\mathbf{u}} + (\widehat{L}\widehat{U})^{-1}[\mathbf{b} - M\hat{\mathbf{u}}] \tag{4.84}$$

Subtracting the equation above from the equation

$$\mathbf{u} = \mathbf{u} + (\widehat{L}\widehat{U})^{-1}[\mathbf{b} - M\mathbf{u}] \tag{4.85}$$

results in

$$\mathbf{u} - \mathbf{u}_{\text{new}} = [I - (\widehat{L}\widehat{U})^{-1}M](\mathbf{u} - \hat{\mathbf{u}}) \tag{4.86}$$

Taking norms shows that this process is expected to yield improvement if $\|I - (\widehat{L}\widehat{U})^{-1}M\| < 1$. In finite precision, iterative refinement usually stops yielding improvement after a few iterations.

4.3.5. Matrix Scaling

Improving accuracy is also possible by scaling the system of equations. Instead of solving Eq. (4.1), the equation

$$(D_1^{-1}MD_2)\mathbf{v} = D_1^{-1}\mathbf{b} \tag{4.87}$$

is solved and subsequently $\mathbf{u}$ is obtained by $\mathbf{u} = D_2\mathbf{v}$. The relative error of $\mathbf{v}$ is about $\varepsilon_0\kappa_\infty(D_1^{-1}MD_2)$, so choosing these matrices well should result in a more accurate solution [16]. Scaling costs $\mathcal{O}(n^2)$ flops.

4.4. Rank-Revealing Decompositions

A **rank-revealing decomposition**, as the name suggests, is a matrix decomposition which tells something about the rank of the matrix. These decompositions can be used to determine whether the matrix is well-conditioned or not and also to solve a system of linear equations.

4.4.1. Singular Value Decomposition

The most interesting decomposition is the **singular value decomposition (SVD)**, which is given as follows [16].

Theorem 4.20. *Let $M \in \mathbb{R}^{m\times n}$ of rank $r \le \min(m,n)$. There exists orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that*

$$M = U\Sigma V^\top, \qquad \Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times n} \tag{4.88}$$

where $\Sigma_r = \mathrm{diag}(\sigma_1, \sigma_2, \dots, \sigma_r) \in \mathbb{R}^{r\times r}$ and

$$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0 \tag{4.89}$$

The values σ_i are called the **singular values** of M. If $r < \min(m,n)$, then 0 is a singular value as well. The columns of U and V are called the **left** and **right singular vectors**, respectively. This theorem immediately shows that, to compute the rank of the matrix, one can look at the singular values in the SVD. As singular values can be very small, determining the rank of a matrix via the SVD in finite precision can be problematic due to rounding errors. It does, however, provide insight into the conditioning of the matrix.

Let $\sigma_{\max} = \sigma_1$ and $\sigma_{\min} = \sigma_r$ if $r = \min(m,n)$ or $\sigma_{\min} = 0$ otherwise. From Theorem 4.20 and Eq. (4.13) it follows that $\|M\|_2 = \sigma_{\max}$ for $M \in \mathbb{R}^{m\times n}$. Now, if $M \in \mathbb{R}^{n\times n}$ such that $\mathrm{rank}(M) = n$, then $\|M^{-1}\|_2 = 1/\sigma_{\min}$. This shows that

$$\kappa_2(M) = \|M\|_2\|M^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}}, \tag{4.90}$$

which illustrates that the SVD contains information on the conditioning of a matrix. For a singular matrix $\kappa_2(M) = \infty$.

Example 4.21. Consider again Example 4.11. The matrix

$$M = \begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \tag{4.91}$$

can be decomposed as

$$M = U\Sigma V^\top \approx \begin{bmatrix} -0.005 & 0.999 \\ -0.999 & -0.005 \end{bmatrix} \begin{bmatrix} 1.414 & 0 \\ 0 & 0.007 \end{bmatrix} \begin{bmatrix} -0.707 & -0.707 \\ -0.707 & 0.707 \end{bmatrix} \tag{4.92}$$

showing that the matrix has full rank, however, $\sigma_{\max}/\sigma_{\min} \approx 202$ which shows that the matrix may be considered ill-conditioned when working in low-precision arithmetic (e.g. $t = 4$).

Calculating the SVD requires an expensive $\mathcal{O}(n^3)$ operations, although variations exists where only part of the SVD is calculated [16].

4.4.2. QR-Factorisation

Another rank-revealing decomposition is the QR-factorisation, where Q is an orthogonal matrix and R an upper triangular matrix [16]. For $M \in \mathbb{R}^{m\times n}$ with $r = \text{rank}(M)$, the Householder QR-factorisation procedure with partial pivoting yields

$$MP = QR, \qquad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \tag{4.93}$$

where $P \in \mathbb{R}^{n\times n}$ is a permutation matrix and $R_{11} \in \mathbb{R}^{r\times r}$.

Example 4.22. The QR-factorisation of M as in Example 4.21 is given as

$$M = QR = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 0 & -0.01 \end{bmatrix} \tag{4.94}$$

In this example, pivoting was not applied.

4.5. Other Performance Optimisations

Here a few performance optimisation techniques are introduced, namely, fill-in reduction strategies and alternatives for the Newton-Raphson method.

4.5.1. Fill-in Reduction Strategies

Section 4.3.2 showed that pivoting strategies can be used to reduce the fill-in during the LU-factorisation process. Other possibilities for reducing fill-in are adjusting the internal component numbering technique in Wanda and matrix reordering schemes.

Component Ordering

Section 2.3.1 shows that the physical connection points of the components and the H-nodes are numbered through a breadth-first search starting from the component first added to the pipeline system. It is possible that a different algorithm and/or starting point for ordering the components results in a matrix with lower bandwidth and hence less fill-in during the LU-decomposition. The advantage of targeting this part of the solution method is that the algorithm only needs to be performed once before running the solution method.

Matrix Reordering Schemes

Another possibility is to look into the feasibility of reordering the matrix after each phase change. This is only worthwhile if a method can be used with small computational costs yielding a significant fill-in reduction. Reordering schemes such as the Cuthill-McKee algorithm can be used [14].

4.5.2. Newton-Raphson Alternatives

As shown in Eq. (2.13), the Newton-Raphson method for solving a system of equations requires the computation of the Jacobian at each iteration. This is an expensive computation. Picard iteration and Quasi-Newton methods are considered as alternatives. The basic idea behind both types of methods is that they require less work per iteration at the cost of more iterations for convergence.

Quasi-Newton Methods

Quasi-Newton methods replace the Jacobian in Eq. (2.13) with an approximation. Examples of these type of methods are Broyden's method [10] and the Symmetric Rank-One method [11].

Picard Iteration

At each time step, the system of non-linear equations produced by Wanda can be written as $\mathbf{F}(\mathbf{u}) = 0$, where $\mathbf{u} = [Q_1\ H_1\ \dots\ Q_n\ H_n]^\top$. It can be rewritten as

$$M\mathbf{u} + G(\mathbf{u}) = \mathbf{b}, \tag{4.95}$$

where $M\mathbf{u}$ denotes the linear part, $G(\mathbf{u})$ the non-linear part and $\mathbf{b}$ a constant vector. This motivates the fixed-point iteration (or Picard iteration) [18]

$$M\mathbf{u}^{(k+1)} = \mathbf{b} - G(\mathbf{u}^{(k)}) \tag{4.96}$$

4.6. Numerical Libraries

The previous sections describe the mathematical theory of solution methods for systems of linear equations. For Wanda an actual implementation of these methods is required. Numerous libraries are available which provide optimised implementations. Two main libraries are considered here. Both libraries are Fortran-based, as the solution method in Wanda is also written in Fortran.

4.6.1. LAPACK

The **Linear Algebra Package (LAPACK)** is one of the most prominent numerical libraries. The library is written in Fortran and is primarily intended for solving equations involving large, dense matrices. One disadvantage of LAPACK is that it cannot directly handle matrices in coordinate format. This format, which is currently used in Wanda, must be converted to band matrix storage. LAPACK relies on the **Basic Linear Algebra Subroutines (BLAS)** implementation on the system, which handle the operations such as matrix-vector multiplication, vector addition, etc. This makes it worthwhile to also investigate which BLAS implementation has the best performance: ATLAS, BLAS or OpenBLAS.

Mathematical operation	LAPACK routine
Matrix norm	`DLANGB`
Condition number	`DGBCON`
LU factorisation step	`DGBTRF`
LU solve step	`DGBTRS`
Iterative refinement	`DGBRFS`
Singular value decomposition	`DGESVD`
QR-factorisation	`DGEQRF`

Table 4.1: LAPACK routines.

Table 4.1 shows the mathematical operations and their corresponding LAPACK computational routines [9]. The routine names start with 'D' which stands for double precision, as computations in Wanda are done in double precision. Note that LAPACK often uses block versions of algorithms which are rich in matrix-matrix multiplications [16]. This leads to better performance on computers. The `DGBTRF` routine uses a block version of the LU-decomposition with partial pivoting, as explained in Section 4.3.2. The condition number estimation technique used in the `DGBCON` routine is the one explained in Section 4.3.3.

If the performance of LAPACK turns out to be insufficient, two derivations of LAPACK can be considered which aim for better performance using the available hardware.

PLASMA

The **Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA)** library is a LAPACK implementation which aims to effectively use multi-core processors for performance improvements [2]. Since almost every computer, especially computers which are meant to run simulation software such as Wanda, contains a multi-core processor, PLASMA may yield better performance.

MAGMA

The **Matrix Algebra on GPU and Multi-core Architectures (MAGMA)** library aims to use both multi-core processors and GPUs for achieving a performance improvement [1]. Virtually all modern computers contain a GPU, hence also involving the GPU in the computations may yield even better performance.

4.6.2. MUMPS

The **Multifrontal Massively Parallel sparse direct Solver (MUMPS)** library is a numerical library written in Fortran which is intended for solving equations involving large, sparse matrices [5]. Since it also uses the coordinate format for storing matrices, it does not require any adjustments to the matrix storage. If LAPACK does not offer the required performance, MUMPS should be able to offer better performance.

MUMPS is purely built to solve systems of equations using Gaussian elimination. For asymmetric matrices, as is the case in Wanda, this amounts to using the LU-decomposition. The library also has routines available for operations such as matrix scaling, condition number estimation and iterative refinement.

4.7. Structural Singularity

The methods in previous sections relied on detecting underdetermined systems of equations by considering the matrix resulting from linearisation. Another interesting approach is to look at the network topology and the resulting system of equations to determine whether or not a solution exists. These equations can be either linear or non-linear. Consider the following definition [22].

Definition 4.23. Let $M \in \mathbb{R}^{n\times n}$. M is called **structurally singular** if every $N \in \mathbb{R}^{n\times n}$, with $N_{ij} = 0$ whenever $M_{ij} = 0$, is singular.

This definition motivates the use of graph theory for determining whether a given system of equations is undetermined.

Example 4.24. Consider again the system of Section 3.1.1.

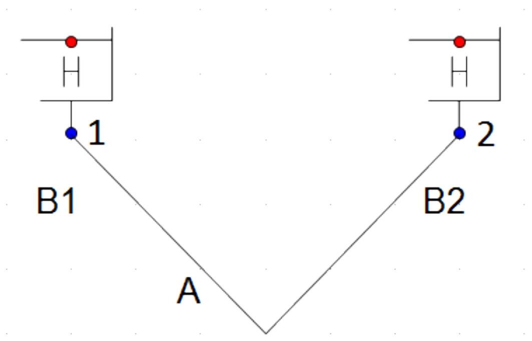


Figure 4.3: Steady state singularity.

This system is described by the following system of equations.

$$H_1 = c_1 \tag{4.97}$$
$$H_1 = H_A \tag{4.98}$$
$$Q_A + Q_1 + Q_2 = 0 \tag{4.99}$$
$$Q_A = 0 \tag{4.100}$$
$$H_2 = H_A \tag{4.101}$$
$$H_2 = c_2 \tag{4.102}$$

Let the matrix $M \in \mathbb{R}^{6\times 6}$ be defined by

$$M_{ij} = \begin{cases} 1, & \text{if variable } j \text{ is present in equation } i \\ 0, & \text{otherwise} \end{cases} \tag{4.103}$$

Note that the matrix is defined in such a way that it can also represent non-linear systems of equations. For this particular system the matrix is given as

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.104}$$

when using the variable ordering $(H_1, Q_1, H_A, Q_A, H_2, Q_2)$. This matrix is structurally singular; no matter what values are substituted in for the non-zeros of M, it will remain singular. After all, column 2 and 6 are always scalar multiples of each other. In other words, the singularity of M is determined by its sparsity pattern. When this is the case, the underlying system of equations is also called structurally singular. A different approach to detecting if a system is undetermined is by determining whether it is structurally singular. One way to do this is by considering a suitable graph representation of the system and look for a criterion when the underlying system is structurally singular.

For example, let $G = (U \cup V, E)$ where U and V denote the equations and variables, respectively, and $uv \in E$ if equation u contains variable v [20].

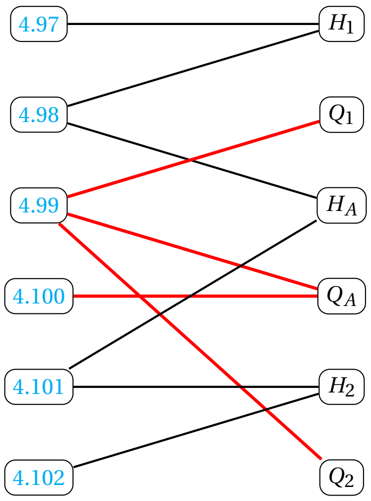


Figure 4.4: Bipartite graph representation.

Fig. 4.5 shows two disconnected components. The red component is underdetermined as it only contains two equations and three variables. The black component contains more equations than variables and does not have a solution if $c_1 \neq c_2$.

Another possibility is to let $D = (V, A)$ be a digraph with $V = \{1,2,3,4,5,6\}$ and $uv \in A$ if equation u contains variable v (using the same equation and variable ordering as in M) [15].

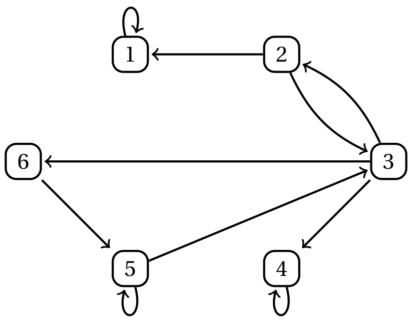


Figure 4.5: Digraph representation.

For this graph representation it is not immediately clear how the system of equations being structurally singular is visible in the graph structure.

These kind of methods could potentially be used to detect faulty systems before trying to solve them. Another potentially interesting approach is by looking at the physical model itself to prevent undetermined systems from occurring.

5 Thesis Proposal

This chapter summarises the problem description, describes the scope of the research in the form of research questions and proposes an approach to solving these questions.

5.1. Problem Summary

The current solution method applies the Newton-Raphson to linearise the system of equations. The resulting matrices are solved by the `LSLXG` routine of the proprietary IMSL numerical library which requires a paid license. The problem with this matrix solver is that it sometimes either crashes or gets stuck in an infinite loop when dealing with singular matrices. The proprietary nature of this library makes troubleshooting difficult. Both the robustness and maintainability of the current matrix solver leave much to be desired. For these reasons, Deltares wants to move away from this library to a robust and easily maintainable solution method, while not giving in on performance.

5.2. Main Research Question

The main research question can be stated as follows.

How can the robustness and maintainability of the current solution method in Wanda be improved, without giving in on efficiency?

The focus is on improving the part of the solution method that solves the matrix-vector equations as this would require a minimal change in the Wanda code. A robust solution method should always either find a solution or return a clear error message. This could be achieved by first checking for singularities before trying to solve the system of equations. Public domain libraries can be used to satisfy the maintainability requirement. As a constraint, the solution method should be efficient, that is, at least as fast time-wise as the current solution method. Improvements are always welcome, but the emphasis is on robustness.

5.3. Detailed Research Questions

The main research question shows that the research should focus on two things: robustness and efficiency. To zoom in on the main research question and set the scope for the research, more detailed questions will be given here.

5.3.1. Robustness

The following questions focus on robustness.

1. *Is it possible to detect undetermined systems via a graph representation and is it feasible to implement this in Wanda?*
2. *Is it possible to prevent undetermined systems from occurring by adjusting the physical model?*
3. *Is it possible to reliably detect singular matrices using condition number estimation techniques?*

4. *Can rank-revealing decompositions be used to both efficiently solve matrix-vector equations and detect singular matrices?*

5.3.2. Efficiency

The following questions focus on efficiency, that is, the speed of the solution method.

5. *Which numerical library offers the best performance (LAPACK or derivatives, or MUMPS)?*
6. *Can fill-in be reduced by using another algorithm to order the components in the pipeline system?*
7. *Is it feasible to apply a reordering scheme to the matrix to reduce fill-in?*
8. *Can fill-in be reduced by using an appropriate pivoting strategy?*
9. *Are there more efficient alternatives to the Newton-Raphson method for solving the system of non-linear equations which yield sufficiently accurate solutions?*
10. *Is it possible to improve the performance of the algorithm which detects the undetermined variables?*

Some of these questions are more important than others. For example, some questions are expected to yield a bigger performance improvement than others. Some also require a big change in the Wanda code, while others require little effort. It follows naturally that some questions have a higher priority than others.

5.4. Research Approach

The first step will be to investigate whether it is possible to detect undetermined systems a priori, that is, before trying to solve them. If this were possible and it is also possible to feasibly implement this in Wanda, no other methods for detecting singular systems are required and hence further research could focus on improving the performance of the Wanda solution method.

The second step is to look for improvement in building the matrix. One approach is to look at alternative methods for ordering the components in the pipeline system (which determines the matrix ordering) in order to reduce the fill-in. If this is not possible, looking into the reordering of the matrix after it is built to reduce the fill-in is another possibility. This can be done by looking at different pivoting strategies such as Markowitz pivoting, but also matrix-reordering schemes such as Cuthill-McKee.

The third step is investigating what the best method is to detect singular matrices, either by using a condition number estimation or a rank-revealing decomposition such as the SVD or QR-decomposition. This step is only required when the first step proves infeasible, although condition number estimation can also be used for estimating solution accuracy. If the LU-decomposition of a matrix is used for solving the system of linear equations, iterative refinement can be used to improve the accuracy of the solution, if desirable.

The fourth step is to investigate which implementation of these methods yields the best results, both performance- and accuracy-wise. This step is heavily connected to the previous one as this also requires investigating which methods should be used for detecting singularities and solving the systems of linear equations. This step also includes tweaking parameters such that the best performance is achieved while producing sufficiently accurate solutions. LAPACK and its derivatives, PLASMA and MAGMA, will be considered as well as the MUMPS library.

The fifth step, if time permits, is investigating if any other performance improvements can be gained. Perhaps performance improvements could be gained by optimising the algorithm which detects the undetermined variables. Furthermore, other methods instead of Newton-Raphson for solving systems of non-linear equations can be investigated, for example, Quasi-Newton methods and Picard iteration.

Bibliography

[1] Matrix algebra on gpu and multi-core archictectures. URL http://icl.cs.utk.edu/magma/.

[2] Parallel linear algebra software for multicore architectures. URL https://bitbucket.org/icl/plasma.

[3] *IMSL Fortran Math Library*, 2014.

[4] *Hydrodynamica van Leidingsystemem*, 2015.

[5] *Multifrontal Massively Parallel Solver. User's Guide*, 2016. URL http://mumps.enseeiht.fr.

[6] *WANDA 4.5 User Manual*, 2017.

[7] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[8] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

[10] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593, 1965. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2003941.

[11] Richard H. Byrd, Humaid Fayez Khalfan, and Robert B. Schnabel. Analysis of a symmetric rank-one trust region method. *SIAM Journal on Optimization*, 6(4):1025–1039, 1996. doi: 10.1137/S1052623493252985. URL https://doi.org/10.1137/S1052623493252985.

[12] S. Camiz and S. Stefani. *Matrices And Graphs*. World Scientific Publishing Company, 1996. ISBN 9789814530088.

[13] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson. An Estimate for the Condition Number of a Matrix. *SIAM Journal on Numerical Analysis*, 16:368–375, April 1979. doi: 10.1137/0716029.

[14] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM. doi: 10.1145/800195.805928. URL http://doi.acm.org/10.1145/800195.805928.

[15] Iain S Duff, Albert M Erisman, and John K Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Inc., New York, NY, USA, 1986. ISBN 0-198-53408-6.

[16] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996. ISBN 9780801854149.

[17] Nicholas J. Highham. A survey of condition number estimation for triangular matrices. *SIAM Rev.*, 29 (4):575–596, December 1987. ISSN 0036-1445. doi: 10.1137/1029112. URL http://dx.doi.org/10.1137/1029112.

[18] C. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995. doi: 10.1137/1.9781611970944. URL http://epubs.siam.org/doi/abs/10.1137/1.9781611970944.

[19] Harry M. Markowitz. The elimination form of the inverse and its application to linear programming. *Manage. Sci.*, 3(3):255–269, April 1957. ISSN 0025-1909. doi: 10.1287/mnsc.3.3.255. URL http://dx.doi.org/10.1287/mnsc.3.3.255.

[20] Kazuo Murota. *Matrices and Matroids for Systems Analysis*. Springer Publishing Company, Incorporated, 1st edition, 2009. ISBN 3642039936, 9783642039935.

[21] Farid N. Najm. *Circuit Simulation*. Wiley-IEEE Press, 2010. ISBN 0470538716, 9780470538715.

[22] Arnold Neumaier. Scaling and structural condition numbers. *Linear Algebra and its Applications*, 263, 09 1996.

[23] H.E.A. van den Akker and R.F. Mudde. *Fysische Transportverschijnselen I*. VSSD, 2005.

[24] J. van Kan, A. Segal, and F.J. Vermolen. *Numerical Methods in Scientific Computing*. Delft Academic Press, 2014.

[25] A.E.P. Veldman and A. Velická. Stromingsleer, 2007.

[26] C. Vuik and D.J.P. Lahaye. Scientific computing, 2015.

[27] C. Vuik, F.J. Vermolen, M.B. van Gijzen, and M.J. Vuik. *Numerical Methods for Ordinary Differential Equations*. Delft Academic Press, 2015.

[28] E.B. Wylie and V.L. Streeter. *Fluid transients*. Number v. 1977 in Advanced book program. McGraw-Hill International Book Co., 1978.

A

Wanda

A.1. Conservation Laws

A.1.1. Conservation of Momentum

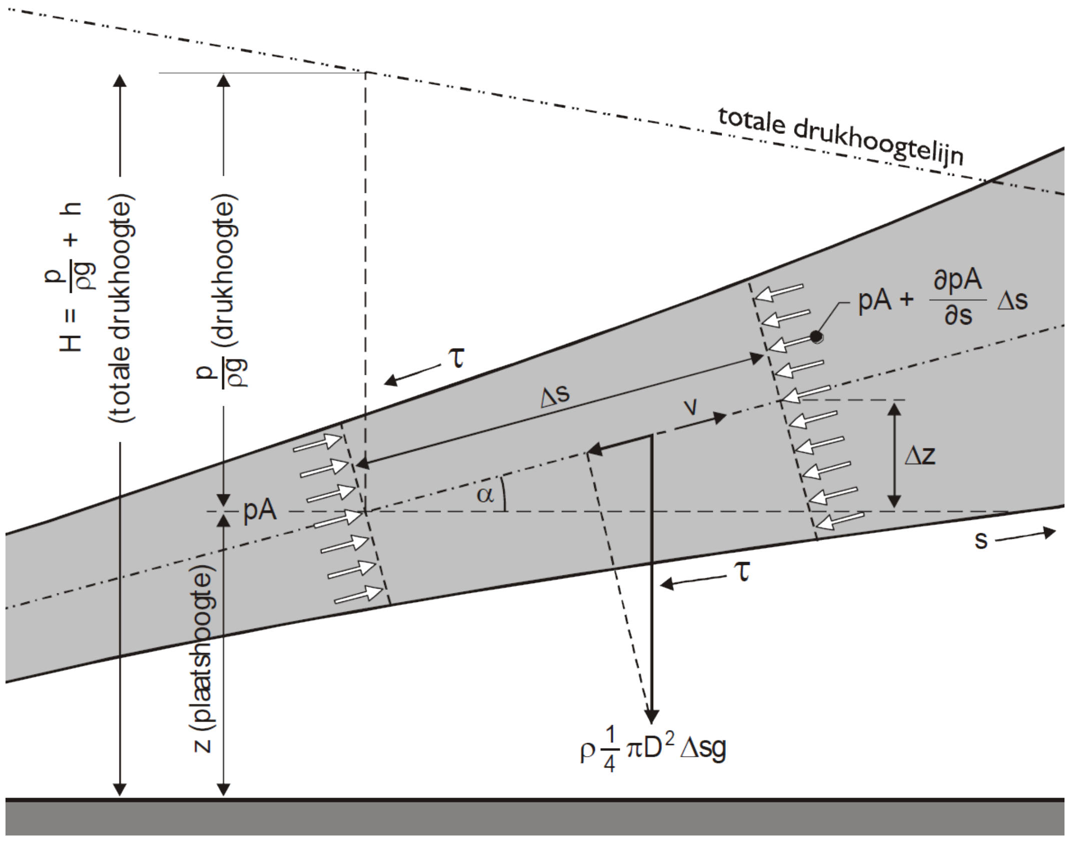

totale drukhoogtelijn
$H = \frac{p}{\rho g} + h$ (totale drukhoogte)
$\frac{p}{\rho g}$ (drukhoogte)
z (plaatshoogte)
pA
τ
Δs
v
α
Δz
$pA + \frac{\partial pA}{\partial s}\Delta s$
τ
s
$\rho \frac{1}{4}\pi D^2 \Delta sg$

Figure A.1: Conservation of momentum.

Note that here Δx is used instead of Δs. From Fig. A.1 the law of conservation of momentum can be derived, for Δx small.

$$\underbrace{pA + \left(pA + \frac{\partial(\rho A)}{\partial x}\Delta x\right)}_{\text{Pressure on ends}} - \underbrace{\rho g \Delta x \left(\frac{A + A + \frac{\partial A}{\partial x}\Delta x}{2}\right)\sin\alpha}_{\text{Force due to weight}} - \underbrace{\tau O \Delta x}_{\text{Friction}} + \underbrace{\left(p + \frac{1}{2}\frac{\partial p}{\partial x}\Delta x\right)\frac{\partial A}{\partial x}\Delta x}_{\text{Pressure on sides}}$$

$$\underbrace{= \rho \Delta x \left(\frac{A + A + \frac{\partial A}{\partial x} \Delta x}{2} \right) \frac{\mathrm{d}v}{\mathrm{d}t}}_{\text{Element mass acceleration}} \tag{A.1}$$

Here O denotes the circumference of the pipe. The friction term describes friction due to the pipe wall.

A.1.2. Conservation of Mass

Now the conservation of mass will be derived. Note that here Δx can vary over time, since an element of mass can shrink in size due to pressure.

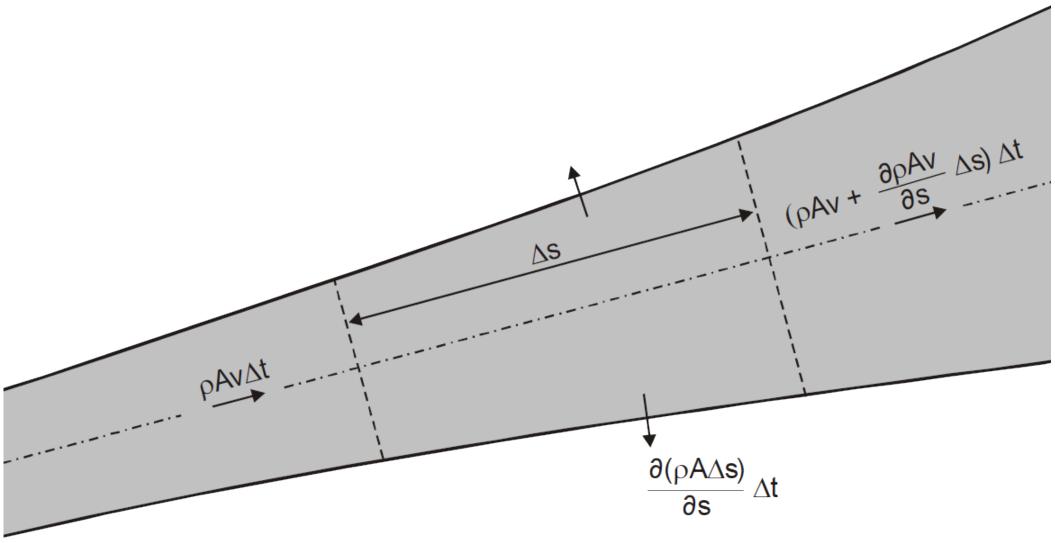


Figure A.2: Conservation of mass.

Fig. A.2 leads to the following equation (again using s instead of x to denote place).

$$\underbrace{\rho A v \Delta t}_{\text{Inflow}} = \underbrace{\left(\rho A v + \frac{\partial(\rho A v)}{\partial x} \Delta x \right) \Delta t}_{\text{Outflow}} + \underbrace{\frac{\partial(\rho A \Delta x)}{\partial t} \Delta t}_{\text{Net flow}} \tag{A.2}$$

I.e., inflow is equal to outflow plus the mass that is kept within the element.

A.2. Large Test Cases

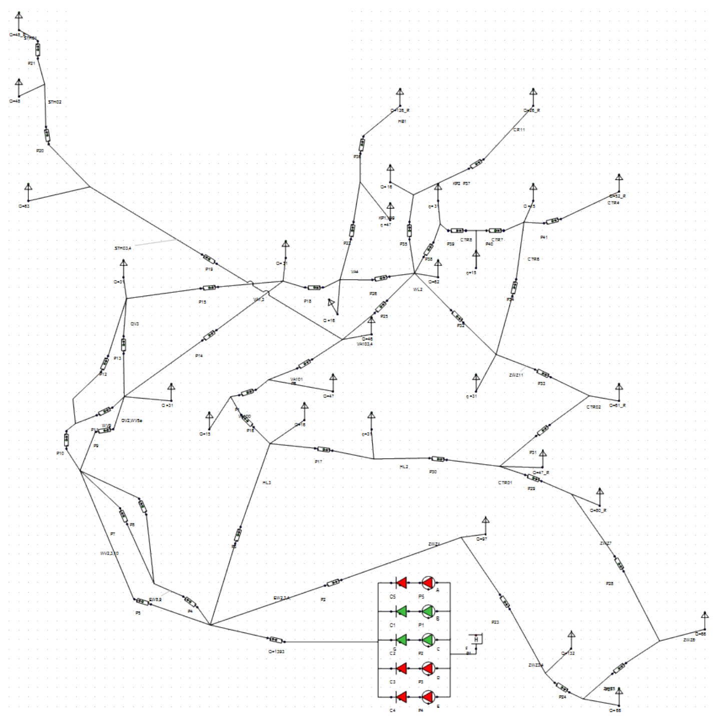


Figure A.3: Drinking water test case.

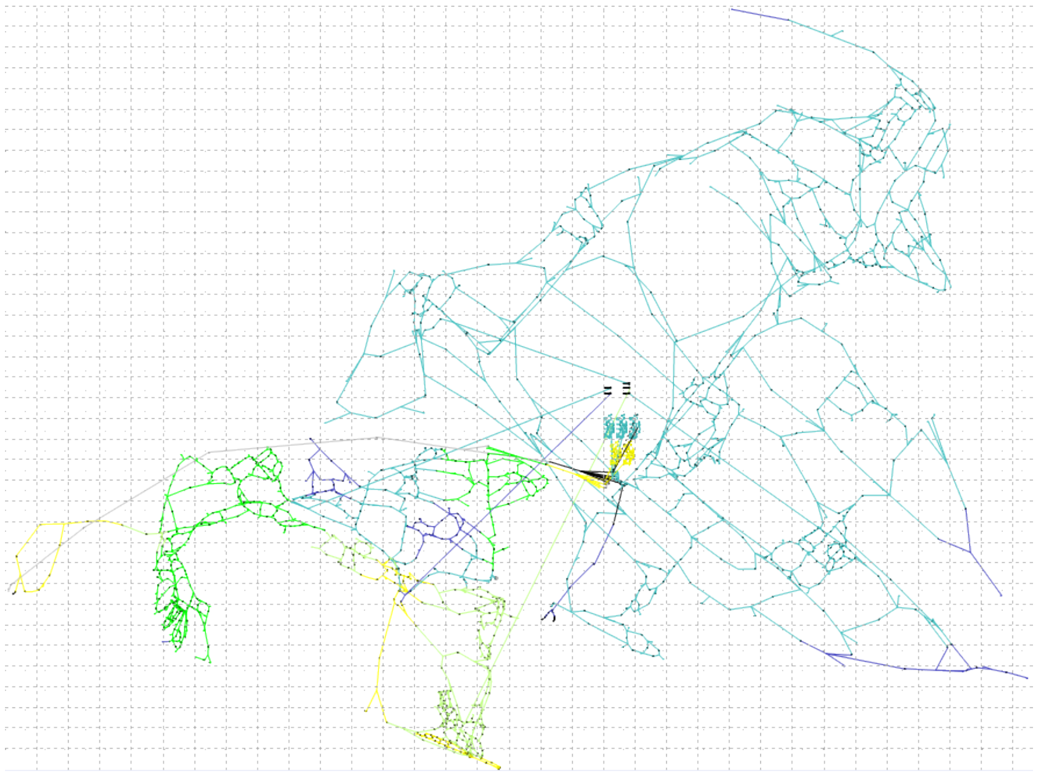


Figure A.4: Noord-Holland 1 test case.

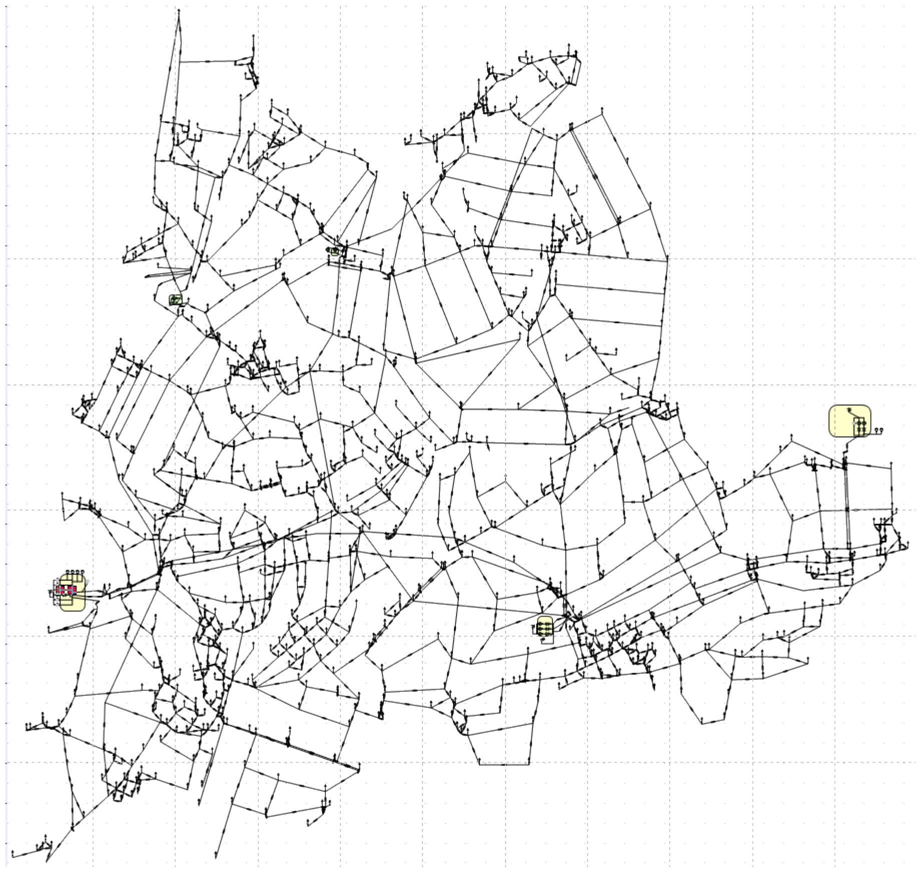


Figure A.5: Noord-Holland 2 test case.