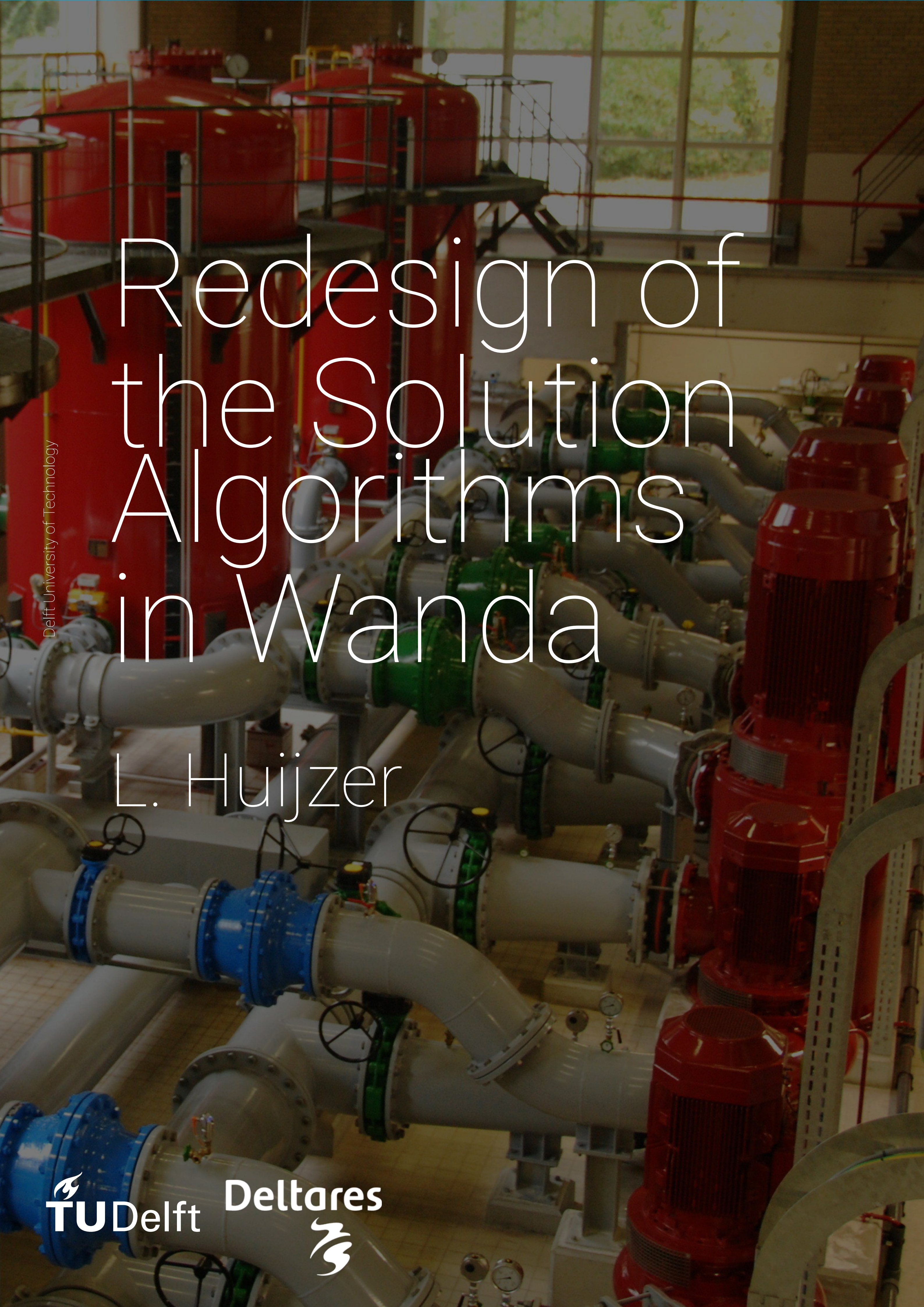


Redesign of the Solution Algorithms in Wanda

L. Huijzer

Delft University of Technology

Redesign of the Solution Algorithms in Wanda

by

L. Huijzer

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 28, 2018 at 11:00 AM.

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

TUDelft Deltares

Abstract

The Wanda software package developed by Deltares can used for simulating 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. For both types of simulations a system consisting of both linear and non-linear equations needs to be solved for the main unknown quantities, flow rate and head. 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 which requires a paid license. 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. The solution method therefore requires a redesign which should improve its robustness and maintainability. On the other hand, no ground should be yielded in terms of solution accuracy and performance.

The singularity of the matrices are caused by quantities being underdetermined either due to user error in network design or phase changes such as a closing valve. In this report, a graph-theoretic approach is taken to detect these structural singularities in the form of determining a maximum size matching in a graph representing the system of equations. This approach gives information about which quantity is undetermined where in the pipeline system. As an alternative, condition number estimation is implemented. Furthermore, the IMSL library is replaced by LAPACK, which is a lightweight and versatile alternative. The open source nature of LAPACK and its permissive license ensure its maintainability. Since the matrices are banded, the band version of the LAPACK algorithms can be used. The new solution method is compared to IMSL and evaluated in terms of robustness, accuracy and performance.

The graph-theoretic method resolves the robustness issues of the IMSL-based method, while showing great performance. The information it gives is used to either correct the matrix and continue the simulation or output an appropriate error message, ensuring a user-friendly experience. Condition number estimation is too slow while also not being useful for further matrix correction purposes and is therefore disregarded. To improve the accuracy of LAPACK, iterative refinement is used. The maximum relative error measured over all the test cases was about 2.5%, resulting from an ill-conditioned case. Overall, the accuracy compared to IMSL is good, so that users will not be able to notice large difference in solution quality between the two solution methods. To improve the performance of LAPACK, Reverse Cuthill-Mckee (RCM) is applied to reduce the bandwidth of the matrices. Using several test cases, the LAPACK performance is shown to be similar to that of IMSL when using RCM. In that respect, the transition from IMSL to LAPACK should also be flawless. A vendor-optimised LAPACK implementation did not yield any significant performance gains. It can be concluded that the new solution method is an improvement in terms of robustness and maintainability, while showing similar solution accuracy and performance.

Future work can focus on improving the performance by calculating the RCM permutation only once every time step, as well as keeping the Jacobian constant during each time step. As matrix bandwidth determines the performance of the matrix solver, better methods to reduce the matrix bandwidth could also yield significant improvements. This could be achieved by either ordering the components in the pipeline system a priori, or simply using matrix bandwidth reduction techniques. Optimised LAPACK libraries as well as a different numerical library such as MUMPS could improve the performance even further.

Acknowledgements

He has made everything beautiful in its time. He has also set eternity in the human heart;
yet no one can fathom what God has done from beginning to end.

– Ecclesiastes 3:11 (NIV)

I would like to show my gratitude to my supervisors, Martin and Sam. Not only have they supported me greatly with their invaluable advice and ideas, but also through their encouragement and enthusiasm. Furthermore, my gratitude goes out to the people at Deltares who were of great help during my time there. I would also like to thank my parents for their love and support, for believing in me, and for giving me the opportunity to study. Last, but not least, I would like to thank the Lord, for where would I be without You?

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
T	Temperature	$°C$
v	Velocity (average)	ms^{-1}
W	Mass flow	kgs^{-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 availability 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 a drinking water network. In order to prevent these kinds 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.

Wanda is a software package developed by Deltares that allows for the design, control and optimisation of user-built pipeline systems. Both steady state flow and transient flow simulations can be performed. The first type of simulation is often used during the initial design phase of a pipeline system. Transient flow simulations are mostly used for investigating the effect of various scenarios such as a pump trip on the performance and safety of a pipeline system. There is, however, one problem: there are pipeline systems for which the matrix solver either crashes or gets stuck in an infinite loop due to the matrix being singular. This happens when systems are built which do not contain enough information about their physical properties for a unique solution to exist. The current matrix solver is a part of a proprietary software library, the IMSL numerical library, and hence does not lend itself to troubleshooting. It also requires a paid license. Therefore, the solution method requires a redesign. A move to an open source numerical library with a permissive license is desirable. Furthermore, a great addition to Wanda would be a method to detect faulty pipeline systems before trying to run the simulations. Hence improvements can be made in terms of robustness and maintainability. On the other hand, yielding (much) ground in terms of accuracy and performance is not an option.

1.1. Research Goal and Approach

The goal of this report is to answer the main research question, which is posed as follows.

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

Numerous open source alternatives to IMSL are available. MUMPS [7] and LAPACK [9] are good examples. In this report LAPACK is considered as an alternative, since the matrix dimensions almost never exceed 10000 × 10000 and hence LAPACK is expected to perform well. It also offers a reliable and versatile solution. To tackle the robustness issue the focus is on detecting faulty pipeline systems by detecting structurally singular matrices using a graph-theoretic approach [24]. The new solution method is then compared to the original solution method and evaluated in terms of robustness, performance and accuracy.

1.2. Report Outline

Chapter 2 starts out 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. The redesign requirements, research questions and research approach will be posed in Chapter 4. The numerical methods involved in solving systems of linear equations will be explained in Chapter 5. Chapter 6 expounds on graph-theoretic methods to decrease the computational costs of solving systems of linear equations using LAPACK. Chapter 7 introduces the mathematical theory to detect faulty pipeline systems. Chapter 8 presents the results obtained using the new solution method and evaluates the new approach. Finally, Chapter 9 gives the final conclusions and recommendations for future work.

1.3. 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.,

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

- 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^{th} row
$M(:,j)$	denotes the j^{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^{th} and j^{th} rows

- Examples are given in boxes.

Example 1.1. This is an example.

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 [50].

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.

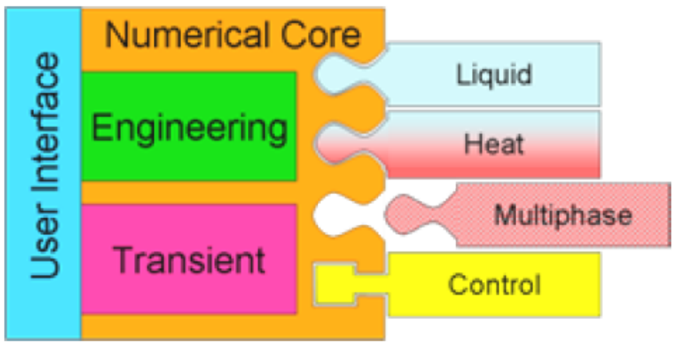


Figure 2.1: Wanda architecture.

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.

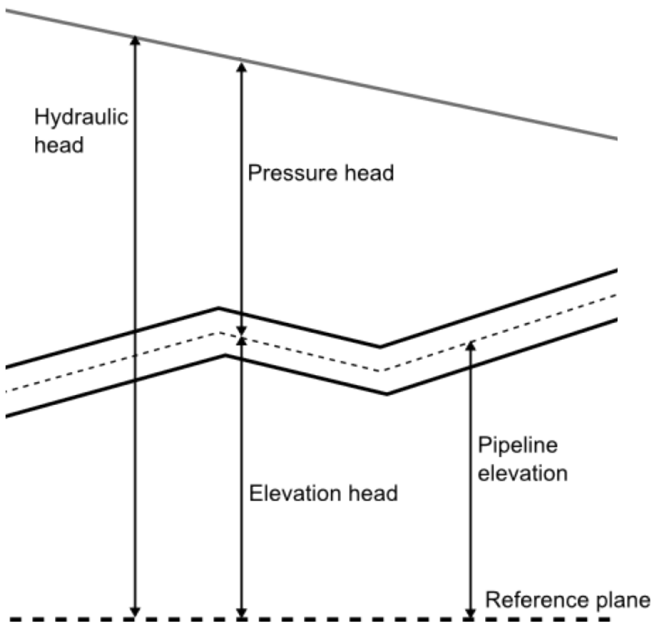


Figure 2.2: Hydraulic head.

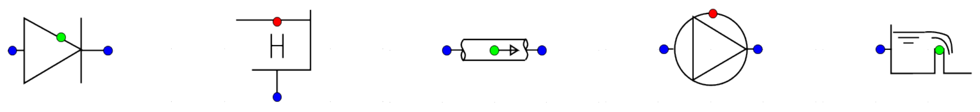
As shown in Fig. 2.2, H is the sum of pressure head $(p/\rho g)$ and elevation head (z), as measured relative to a fixed reference plane.

These two quantities are the main quantities of interest in the Wanda model. In the heat module, the relevant quantities are mass flow W $[kgs^{-1}]$, pressure p $[kgm^{-1}s^{-2}]$ and temperature T $[°C]$.

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

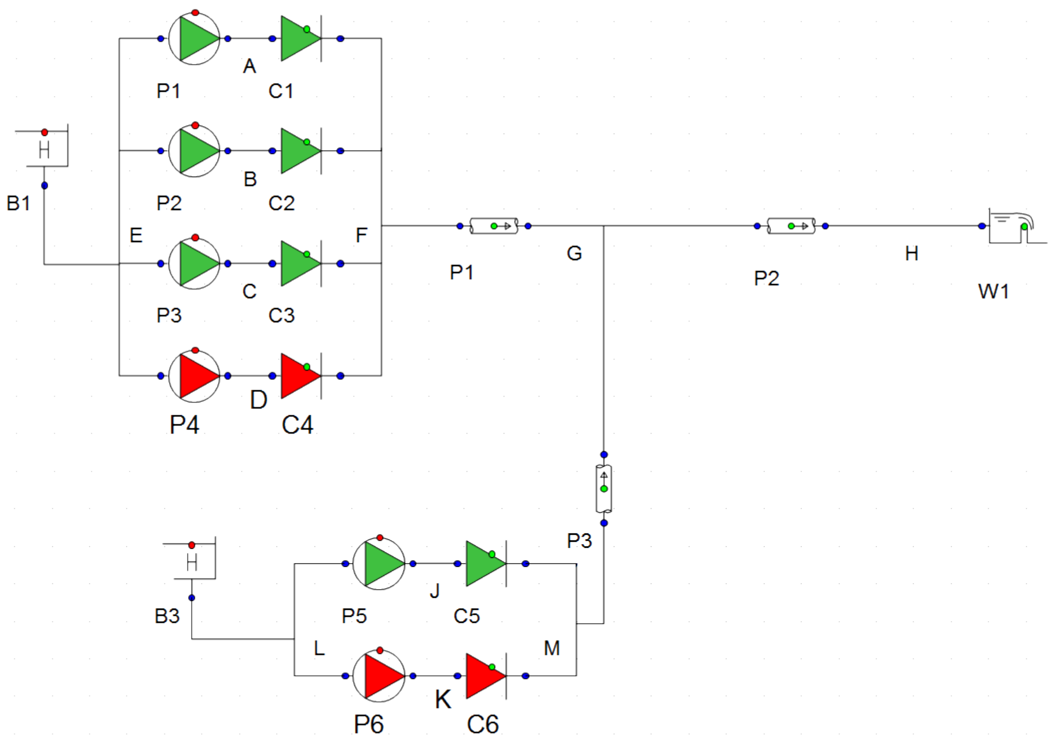


Figure 2.4: Typical sewage system.

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.

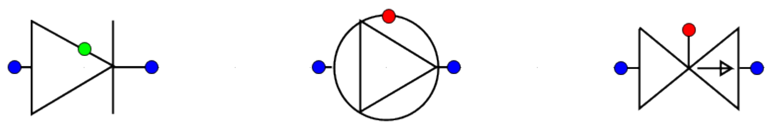


Figure 2.5: Fall type components: Check valve, pump and valve.

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 [50]. 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.

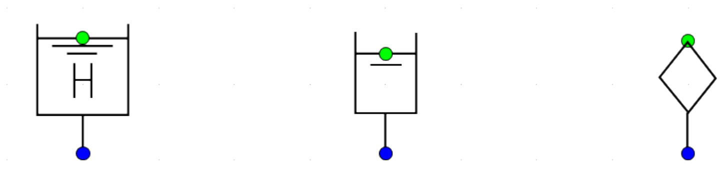


Figure 2.6: Supplier type components: Reservoir, surge tower and vent.

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.

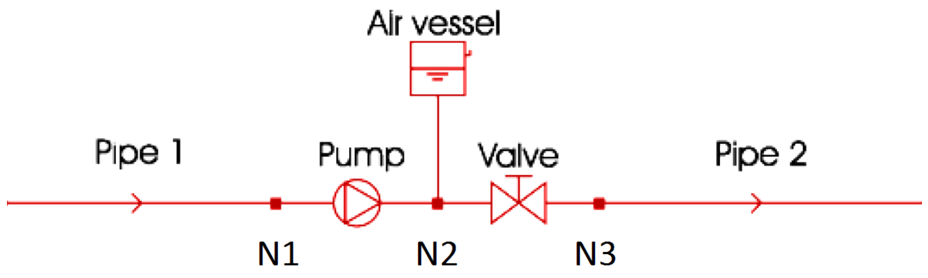


Figure 2.7: H-node example.

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

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

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.

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

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.

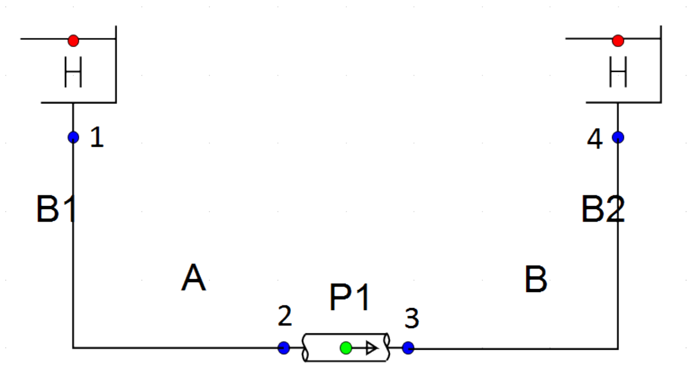


Figure 2.8: Small pipeline system.

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

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

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 [47] and [50].

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 [50].

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 $\Delta t = \Delta x / c$.

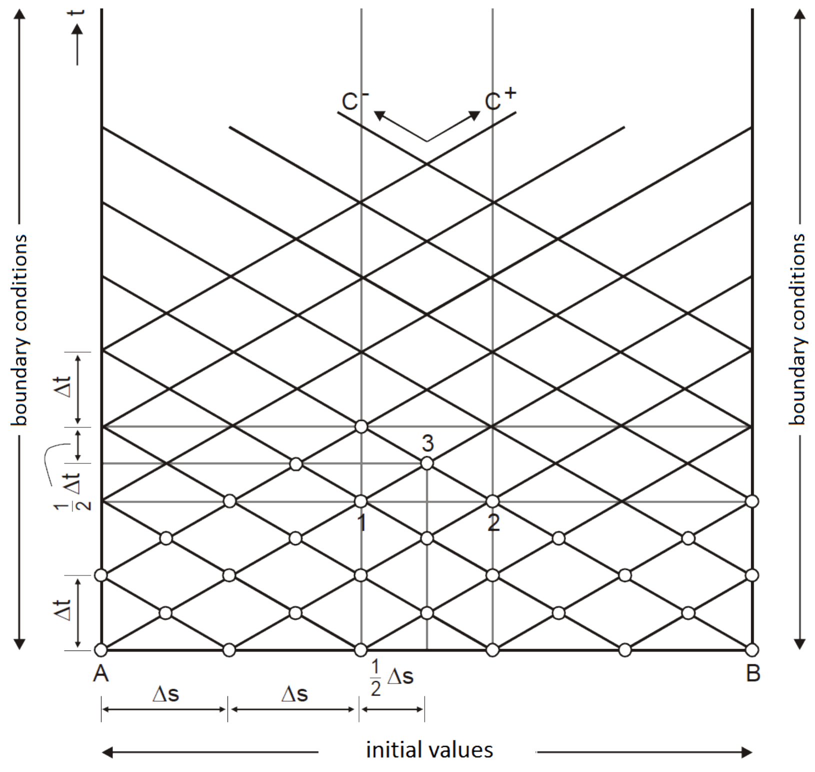


Figure 2.9: The x-t plane for constant c.

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 [50].

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

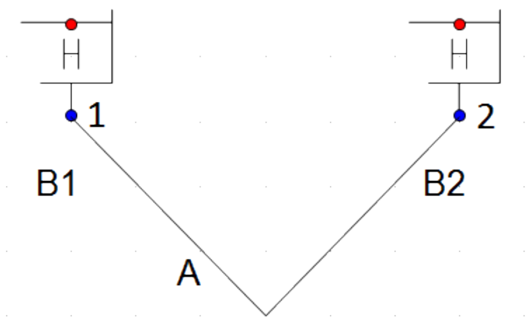


Figure 3.1: System leading to singular matrix due to user error.

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.

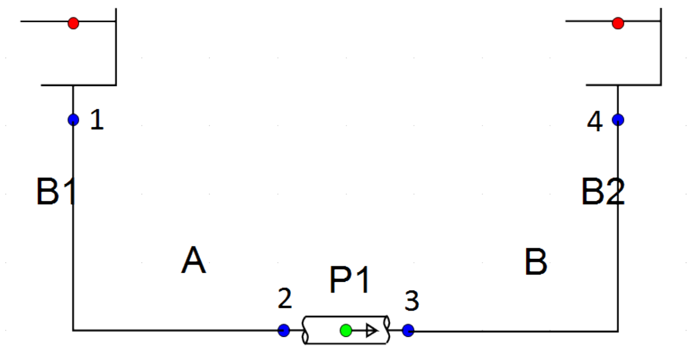


Figure 3.2: Small pipeline system.

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.

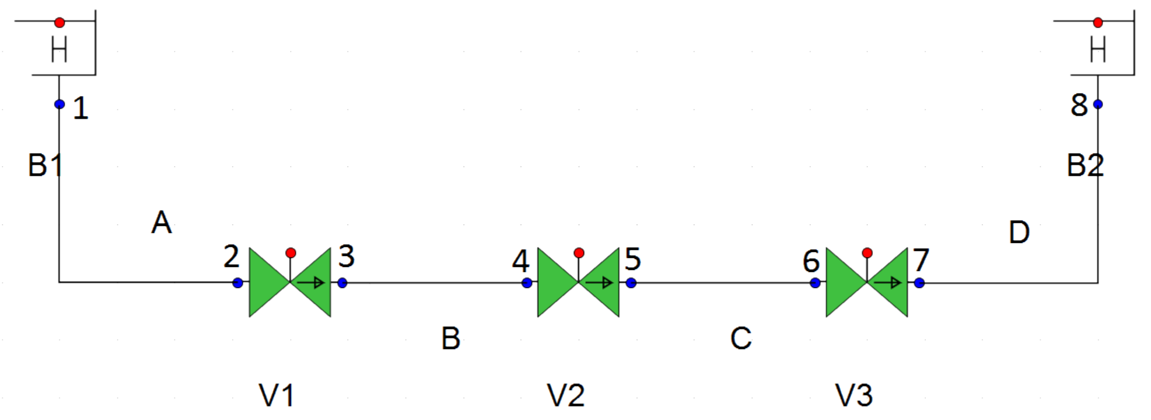


Figure 3.3: System leading to singular matrix due to phase transitions.

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

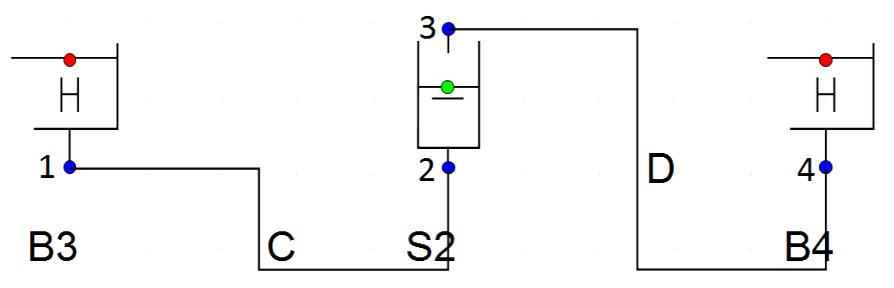


Figure 3.4: Shaft test case.

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 partially filled the upstream and downstream H are decoupled and $c_1 = c_3$ is required for a solution to exist. If S_2 is submerged, 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 at $t = 0$ 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. These test cases are larger than any other cases used in practice, hence these cases serve as excellent benchmarks for measuring the Wanda run time.

Name	H-components	n	nnz	b_l	b_u
Filter	654	3516	8487	203	204
Noord-Holland 1	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 5.1 and their significance is explained in Section 5.3.2.

Filter

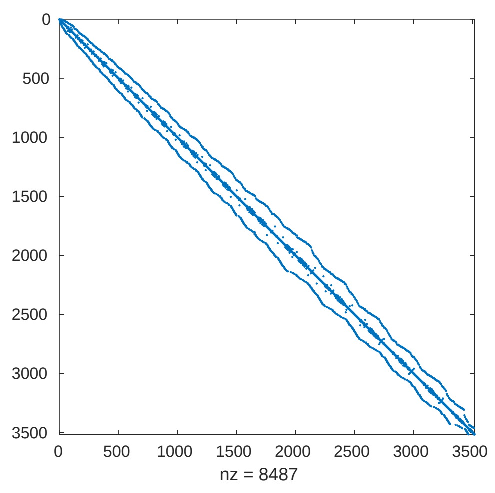
This case, which is not aimed at modelling a realistic situation, consists of one boundary condition prescribing H and 654 resistance components configured in a honeycomb grid. A visualisation of the network is given by Fig. A.3. Its sparsity pattern is visualised in Fig. 3.5a. The transient flow simulation spans 1000 seconds with a 1 second time step. No phase changes occur during the simulation.

Noord-Holland 1

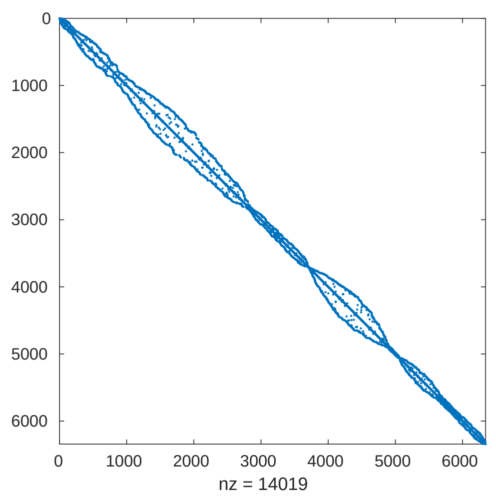
The Noord-Holland 1 (NH1) test case is a large test case representing a part of the drinking water network in Noord-Holland. A visualisation is given in Fig. A.4. The sparsity pattern of the steady state matrix is given in Fig. 3.5b. The transient flow simulation spans 200 seconds with a time step of 0.2 seconds and includes phase changes.

Noord-Holland 2

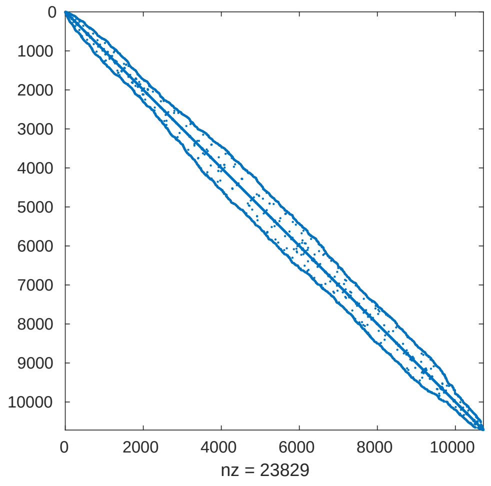
The Noord-Holland 2 (NH2) case also represents part of the drinking water network in Noord-Holland. A visualisation is given in Fig. A.5. The sparsity pattern of the steady state matrix is given in Fig. 3.5c. The simulation of transient flow spans 200 seconds with a time step of 0.1 seconds and includes phase changes.



(a) Filter



(b) Noord-Holland 1

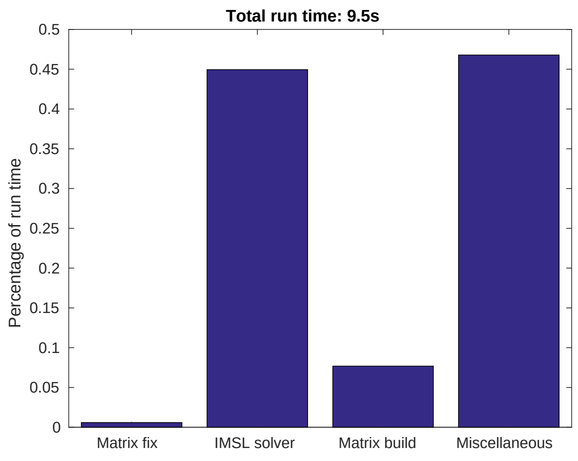


(c) Noord-Holland 2

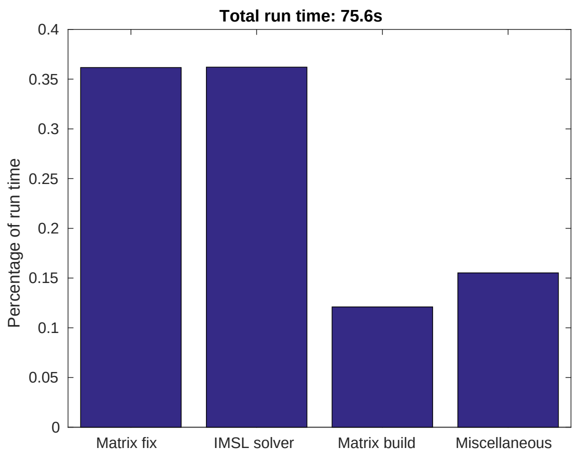
Figure 3.5: Sparsity pattern large test cases

3.5. Wanda Profiling

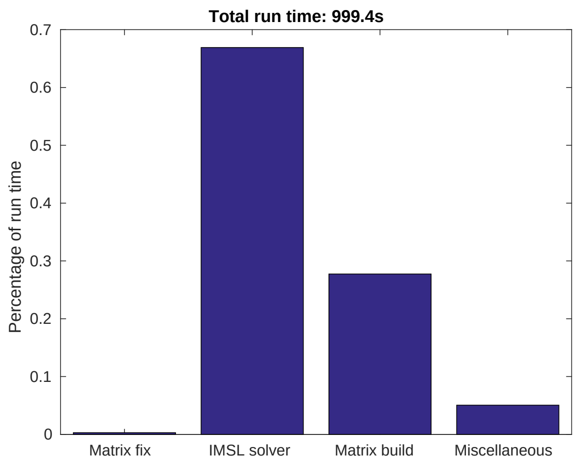
Profiling results of the current solution method of Wanda for each of the large test cases are given here. The results are obtained by simulating the transient flow scenario of each large test case, and taking the average time over five runs using the Fortran `GETTIM` routine for measuring elapsed clock time. The transient flow scenario details can be found in the previous section. During the simulation the results of each second of simulation time will be written to an output file. This way of testing reflects the average Wanda usage.



(a) Filter



(b) Noord-Holland 1



(c) Noord-Holland 2

Figure 3.6: Elapsed run time results for each test case.

Fig. 3.6 shows the percentage of the total run time divided into four categories.

- ’Matrix fix’ is the routine that searches for nodes where H is undetermined after phase changes have occurred and, if applicable, tries to fix this by prescribing H on an H-node. For the NH1 case, this routine takes a significant amount of time, as its simulation includes lots of phase changes. This indicates that there exist cases involving so many phase changes that it may be worth optimising this algorithm, although the absolute time is still limited. For the Filter and NH2 cases, this routine is insignificant. For the Filter problem this is a direct consequence of having no phase changes during the simulation.

- The ’IMSL solver’ is the part which solves the systems of linear equations in the Newton-Raphson iterations. For the filter case this part of the solution method takes about 45% of the time. Due to the numerous phase changes the ’matrix fix’ takes about the same amount of time as the matrix solver for the NH1 case. Less phase changes would result in a larger chunk of time spent in the matrix solver. For the NH2 case the IMSL solver takes about two-thirds of the total run time. This is rather significant.

- The ’matrix build’ routine is the routine which gathers all the relevant information from the components and builds the actual matrix. In the current version of Wanda, the matrix is built from scratch every iteration.

- The ’miscellaneous’ category includes all the other routines. Most of these routines are computational routines that are not IMSL routines and other routines that deal with the logistics of carrying out the simulation.

Overall, the results show that a small increase in computation time, if necessary to increase robustness, would not be such a big problem since the total run times are still limited. Especially since these large test case are bigger than seemingly any other Wanda case used in practice; the average user is not likely to be bothered with a small increase in run time. A large increase is, however, undesirable.

4 Redesign Requirements

This chapter states the redesign requirements, describes the scope of the research in the form of research questions and proposes an approach to solving these questions.

4.1. 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 accuracy and performance.

In other words, a redesign of the solution method should be sufficiently robust, accurate, fast and maintainable.

4.2. Main Research Question

The main research question can be stated as follows.

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

The focus is on improving the part of the solution method that solves the systems of linear 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 such as LAPACK 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. Furthermore, the solution method should be sufficiently accurate.

4.3. Detailed Research Questions

The main research question shows that the research should focus on four things: maintainability, robustness, accuracy and performance. To zoom in on the main research question and set the scope for the research, more detailed questions will be given here for the robustness and efficiency criteria. The accuracy and maintainability requirements are viewed as constraints, hence they do not require their own research questions.

4.3.1. 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?*

4.3.2. Efficiency

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

5. *Which numerical library (or LAPACK implementation) offers the best performance?*

As explained at the end of Section 2.3.1, components in a pipeline systems are ordered using breadth-first search which determines the matrix structure. Exploring alternative methods to order the components could therefore be beneficial.

6. *Can fill-in be reduced by using another algorithm to order the components in the pipeline system?*
7. *What fill-in reduction techniques can be used to improve the matrix solver performance?*
8. *Are there more efficient alternatives to the Newton-Raphson method for solving the system of non-linear equations which yield sufficiently accurate solutions?*

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 large modifications in the Wanda code, while others require little effort. It follows naturally that some questions have a higher priority than others.

4.4. Research Approach

The first step is researching the mathematical theory behind the methods used in the LAPACK library. Condition number estimation will be considered for singular matrix detection. Iterative refinement is considered as a method to increase the solution accuracy. Chapter 5 covers the topics described here.

The second step will be to investigate whether it is possible to detect undetermined systems a priori, that is, before trying to solve them. A graph-theoretic approach is taken to detect underdetermined variables in the system of equations. This information can then also be used to either correct the system of equations or output an informative error message. This is approach is considered in Chapter 7.

The third step is to implement LAPACK and improve its performance. Since the matrices in Wanda are banded, performance improvements can be made by reducing the matrix bandwidth. More specifically, the Reverse Cuthill-Mckee heuristic will be considered within the context of asymmetric matrices. This is described in Chapter 6.

Finally, all the methods covered in this report will be brought together and implemented in Wanda. Each part of the solution method is compared to its alternatives and the final solution method is then evaluated in terms of maintainability, robustness, accuracy and performance in Chapter 8.

5 Numerical Methods

Last chapter illustrated the problem with the current solution method. The goal of this chapter to introduce the numerical methods used in LAPACK, both for detecting singular matrices as solving systems of linear equations. The content of this chapter is for a significant part based on Golub and Van Loan [25].

5.1. 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

$$M\mathbf{u} = \mathbf{b}. \tag{5.1}$$

First some preliminary concepts are introduced.

5.1.1. 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 5.1. The IMSL[a] numerical library uses the **coordinate format** for storing matrices [3, 5]. For example, the matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 3 \\ 0 & 4 & 5 \end{bmatrix} \tag{5.2}$$

is stored as

i	1	2	2	3	3
j	2	1	3	2	3
value	1	2	3	4	5

(5.3)

including the dimension n and nnz(M), which are equal to 3 and 5, respectively. For this example the storage is ordered by row and column, but for the IMSL solver this is not necessary.

[a]See Section 3.3.

A **band matrix** is a special type of sparse matrix. The upper and lower bandwidth of a matrix is defined as follows [25].

Definition 5.2. 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 5.3. The matrix

$$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 0 & 7 \end{bmatrix} \tag{5.4}$$

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

$$\begin{aligned} i &\mapsto b_u + i - j + 1, \quad \max(1, j - b_u) \le i \le \min(1, j + b_l) \\ j &\mapsto j, \qquad\qquad\qquad\qquad\qquad 1 \le j \le n \end{aligned} \tag{5.5}$$

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 5.4. The matrix

$$\begin{bmatrix} M_{11} & M_{12} & 0 & 0 & 0 \\ M_{21} & M_{22} & M_{23} & 0 & 0 \\ M_{31} & M_{32} & M_{33} & M_{34} & 0 \\ 0 & M_{42} & M_{43} & M_{44} & M_{45} \\ 0 & 0 & M_{53} & M_{54} & M_{55} \end{bmatrix} \tag{5.6}$$

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

$$\begin{bmatrix} 0 & M_{12} & M_{23} & M_{34} & M_{45} \\ M_{11} & M_{22} & M_{33} & M_{44} & M_{55} \\ M_{21} & M_{32} & M_{43} & M_{54} & 0 \\ M_{31} & M_{42} & M_{53} & 0 & 0 \end{bmatrix} \tag{5.7}$$

5.1.2. 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 [25, 48].

Definition 5.5. 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.

$$\|\mathbf{u}\|_1 = \sum_{i=1}^{n} |u_i| \tag{5.8}$$

$$\|\mathbf{u}\|_2 = \left(\sum_{i=1}^{n} |u_i|^2\right)^{1/2} \tag{5.9}$$

$$\|\mathbf{u}\|_\infty = \max_{1\le i\le n} |u_i| \tag{5.10}$$

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

Definition 5.6. 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 5.4.

The matrix norm has the **submultiplicative** property

$$\|M_1 M_2\|_p \le \|M_1\|_p \|M_2\|_p \tag{5.11}$$

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. (5.8) to (5.10) for $M \in \mathbb{R}^{m\times n}$ can be computed using the following formulas.

$$\|M\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |M_{ij}| \tag{5.12}$$

$$\|M\|_2 = \sqrt{\lambda_{\max}(M^\top M)} \tag{5.13}$$

$$\|M\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |M_{ij}| \tag{5.14}$$

Here $\lambda_{\max}(\cdot)$ denotes the largest eigenvalue of a matrix. Eqs. (5.12) and (5.14) represent the maximum absolute column and row sums, respectively.

5.1.3. Rounding Errors

Analytically, it is possible to solve Eq. (5.1) 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

$$\pm 0.d_1 d_2 \dots d_t \cdot \beta^e, \tag{5.15}$$

where $d_1 > 0$ and $0 \le d_i < \beta$. Here $0.d_1 d_2 \dots d_t$, β and e are called the mantissa, base and exponent respectively [49]. For numerical calculations Wanda mostly uses double precision numbers, which means that $t = 53$, $\beta = 2$ and $-1024 \le e \le 1023$. **Rounding errors** occur when a real number x is rounded to the nearest floating point number $fl(x)$. Now

$$fl(x) = x(1+\varepsilon), \tag{5.16}$$

where

$$|\varepsilon| \le \frac{1}{2}\beta^{1-t} = \varepsilon_0 \tag{5.17}$$

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

$$|\varepsilon| \le \frac{1}{2}\beta^{1-53} \approx 10^{-16}, \tag{5.18}$$

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.

5.1.4. Quantifying Solution Errors

Vector norms can be used to quantify the error between a solution $\mathbf{u}$ to Eq. (5.1) and its (numerical) approximation $\hat{\mathbf{u}}$. The **absolute error** is given as

$$\|\mathbf{u} - \hat{\mathbf{u}}\| \tag{5.19}$$

and the **relative error** is given as

$$\frac{\|\mathbf{u} - \hat{\mathbf{u}}\|}{\|\mathbf{u}\|} \tag{5.20}$$

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

$$\|\mathbf{b} - M\hat{\mathbf{u}}\| \qquad \text{and} \qquad \frac{\|\mathbf{b} - M\hat{\mathbf{u}}\|}{\|\mathbf{b}\|}, \tag{5.21}$$

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

5.2. 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 5.7. Consider the matrix $a \cdot I_n \in \mathbb{R}^{n\times n}$ where I_n denotes the identity matrix [25]. 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 distinguishing 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 5.8. Consider the following equation.

$$M\mathbf{u} = \begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathbf{b} \tag{5.22}$$

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

$$M\mathbf{u}' = \begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} 0.01 \\ 1 \end{bmatrix} = \mathbf{b} + \Delta\mathbf{b} \tag{5.23}$$

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.

5.2.1. 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 $\mathbf{u}$ [48]. First assume only $\mathbf{b}$ is affected by a perturbation of the form

$$\mathbf{b} \to \mathbf{b} + \Delta\mathbf{b}, \tag{5.24}$$

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

$$M(\mathbf{u} + \Delta\mathbf{u}) = \mathbf{b} + \Delta\mathbf{b}, \tag{5.25}$$

hence by linearity

$$M\Delta\mathbf{u} = \Delta\mathbf{b}. \tag{5.26}$$

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

$$\|\Delta\mathbf{u}\|_p = \|M^{-1}\Delta\mathbf{b}\|_p \leq \|M^{-1}\|_p\|\Delta\mathbf{b}\|_p \tag{5.27}$$

From Eq. (5.25) it also follows by linearity that

$$\|\mathbf{b}\|_p = \|M\mathbf{u}\|_p \leq \|M\|_p\|\mathbf{u}\|_p, \tag{5.28}$$

which implies that

$$\frac{1}{\|\mathbf{u}\|_p} \leq \|M\|_p\frac{1}{\|\mathbf{b}\|_p}. \tag{5.29}$$

By combining Eqs. (5.27) and (5.29) the following bound is obtained.

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \|M\|_p\|M^{-1}\|_p\frac{\|\Delta\mathbf{b}\|_p}{\|\mathbf{b}\|_p}\| \leq \delta\|M\|_p\|M^{-1}\|_p \tag{5.30}$$

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 $\mathbf{u}$.

5.2.2. Definition and Properties

Definition 5.9. Let $M \in \mathbb{R}^{n\times n}$. The **condition number** of M using the p-norm is defined as [25]

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

Note that by submultiplicativity

$$\kappa_p(M) = \|M\|_p\|M^{-1}\|_p \geq \|MM^{-1}\|_p = \|I_n\|_p = 1 \tag{5.31}$$

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**) [25]. 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 $M \in \mathbb{R}^{n\times n}$

$$c_1\kappa_{p_1}(M) \leq \kappa_{p_2}(M) \leq c_2\kappa_{p_2}(M). \tag{5.32}$$

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

Example 5.10. Consider again Example 5.7. Calculating the condition number of $a^{-1}I_n$ shows that

$$\kappa_\infty(M) = \|a\cdot I_n\|_\infty\|(a\cdot I_n)^{-1}\|_\infty = a^{-1}\|I_n\|_\infty a\|I_n\|_\infty = 1, \tag{5.33}$$

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

Example 5.11. Consider Example 5.8. M and M^{-1} are given as

$$\begin{bmatrix} 0 & 0.01 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -100 & 1 \\ 100 & 0 \end{bmatrix}, \tag{5.34}$$

respectively. Now

$$\kappa_\infty(M) = \|M\|_\infty\|M^{-1}\|_\infty = 2\cdot 101 = 202 \tag{5.35}$$

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

$$\frac{\|\Delta\mathbf{u}\|_\infty}{\|\mathbf{u}\|_\infty} \leq 202\cdot 0.01 = 2.02, \tag{5.36}$$

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

5.2.3. Matrix and Right-Hand Side Perturbation

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

$$\mathbf{b} \to \mathbf{b} + \Delta\mathbf{b} \tag{5.37}$$

$$M \to M + \Delta M \tag{5.38}$$

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

$$(M + \Delta M)\mathbf{v} = \mathbf{b} + \Delta\mathbf{b}. \tag{5.39}$$

Assuming that $\delta\kappa_p(M) < 1$ (which prevents $M + \Delta M$ from becoming singular), it can be shown that [25]

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \frac{2\delta}{1 - \delta\kappa_p(M)}\kappa_p(M) \tag{5.40}$$

Example 5.12. 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

$$\frac{\|\Delta\mathbf{u}\|_p}{\|\mathbf{u}\|_p} \leq \frac{2 \cdot 0.5 \cdot 10^{-6}}{1 - 0.5 \cdot 10^{-6}10^6}10^6 = 2 \tag{5.41}$$

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

5.2.4. 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 [25]. 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.

5.2.5. 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.

5.3. *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. (5.1). Using Gaussian elimination M is factored into

$$M = LU, \tag{5.42}$$

where $L, U \in \mathbb{R}^{n\times n}$ are lower and upper triangular matrices, respectively. Solving Eq. (5.1) now amounts to solving the forward step

$$L\mathbf{y} = \mathbf{b} \tag{5.43}$$

and backward step

$$U\mathbf{u} = \mathbf{y} \tag{5.44}$$

such that

$$M\mathbf{u} = LU\mathbf{u} = L\mathbf{y} = \mathbf{b}. \tag{5.45}$$

In Section 5.3.3 it will be shown how the LU-decomposition can be used to estimate the condition number of a matrix.

Example 5.13. Consider the following system of linear equations.

$$u_1 + 2u_2 = 1 \tag{5.46}$$
$$3u_1 + 4u_2 = 1 \tag{5.47}$$

Subtracting 3 times Eq. (5.46) from Eq. (5.47) results in the triangular system

$$u_1 + 2u_2 = 1 \tag{5.48}$$
$$-2u_2 = -2 \tag{5.49}$$

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

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} = LU \tag{5.50}$$

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 $G_1, \ldots, G_{n-1}$ to M, such that it is reduced to row echelon form, i.e. $U = G_{n-1}G_{n-2}\ldots G_1 M$. 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.

$$M^{(k)} = G_k G_{k-1} \ldots G_1 M, \qquad k = 1, \ldots, n-1 \tag{5.51}$$

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

Definition 5.14. The kth Gauss-vector $\alpha_k \in \mathbb{R}^n$ is defined as [48]

$$\boldsymbol{\alpha}^{(k)} = [\underbrace{0, \ldots, 0}_{k}, \alpha^{(k)}_{k+1}, \ldots, \alpha^{(k)}_{n}], \tag{5.52}$$

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

The kth **Gaussian transformation** G_k can now be defined as

$$G_k = I - \boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top \tag{5.53}$$

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 [25]. 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

$$U = G_{n-1}G_{n-2}\ldots G_1 M. \tag{5.54}$$

It is not hard to show that

$$G_k^{-1} = I + \boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top \tag{5.55}$$

and

$$(G_{n-1}G_{n-2}\dots G_1)^{-1} = G_1^{-1}G_2^{-1}\dots G_{n-1}^{-1} = \prod_{k=1}^{n-1}\left(I + \boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top\right) = I + \sum_{k=1}^{n-1}\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top \tag{5.56}$$

Finally, if L is defined as

$$L = I + \sum_{k=1}^{n-1}\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^\top, \tag{5.57}$$

then $M = LU$ [25]. At each iteration k, the matrix $M^{(k)}$ can be partitioned into

$$M^{(k)} = \begin{bmatrix} N_{11}^{(k)} & N_{12}^{(k)} \\ 0 & N_{22}^{(k)} \end{bmatrix}, \tag{5.58}$$

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

5.3.1. 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 [25].

Algorithm 5.1 *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 $2b_lb_un$ flops. After factorisation, the solution to Eq. (5.1) can be obtained by solving the forward and backward steps as in Eqs. (5.43) and (5.44). The following versions of the forward and backward steps overwrite the right-hand side $\mathbf{b}$ with the solution of the substitution steps.

Algorithm 5.2 *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 5.3 *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.

5.3.2. 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 5.15. LU factorisation cannot be applied to the matrix

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{5.59}$$

as it has a zero pivot, while the matrix is well-conditioned [48].

Example 5.16. Consider the following equation in $\beta = 10$, $t = 3$ floating point arithmetic [25].

$$\begin{bmatrix} 0.001 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{u} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \tag{5.60}$$

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

$$\widehat{L} = \begin{bmatrix} 1 & 0 \\ 1000 & 1 \end{bmatrix}, \quad \widehat{U} = \begin{bmatrix} 0.001 & 1 \\ 0 & -1000 \end{bmatrix} \tag{5.61}$$

and

$$\widehat{L}\widehat{U} = \begin{bmatrix} 0.001 & 1 \\ 1 & 0 \end{bmatrix} \tag{5.62}$$

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 [25]) that the computed $\widehat{L}$ and $\widehat{U}$ satisfy

$$\widehat{L}\widehat{U} = M + \Delta M, \tag{5.63}$$

where

$$\|\Delta M\| \le n\varepsilon_0 \|\widehat{L}\| \|\widehat{U}\| \tag{5.64}$$

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 5.14 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 $M^{(k-1)}$,

$$P_k M^{(k-1)} Q_k, \tag{5.65}$$

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. (5.58)) [25]. The matrices P_k and Q_k represent the row and column interchange necessary to achieve this, respectively. In other words, Algorithm 5.1 becomes [25]

Algorithm 5.4 *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 5.17. Let $M \in \mathbb{R}^{n\times n}$. The **growth factor** γ of the Gaussian elimination of M is defined as

$$\gamma = \frac{\max\{\sigma, \sigma_1, \dots, \sigma_{n-1}\}}{\sigma}, \tag{5.66}$$

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

$$\widehat{L}\widehat{U} = M + \Delta M, \tag{5.67}$$

where

$$\|\Delta M\|_\infty \leq 6 n^3 \varepsilon_0 \gamma \|M\|_\infty \tag{5.68}$$

and ε_0 denotes the machine precision [25]. In practice, γ is usually of order 10.

Complete pivoting results in an *LU*-factorisation of the form

$$PAQ = LU \tag{5.69}$$

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 [25]. 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 5.18. Consider the discretisation matrix of the Laplacian in 2D [48].

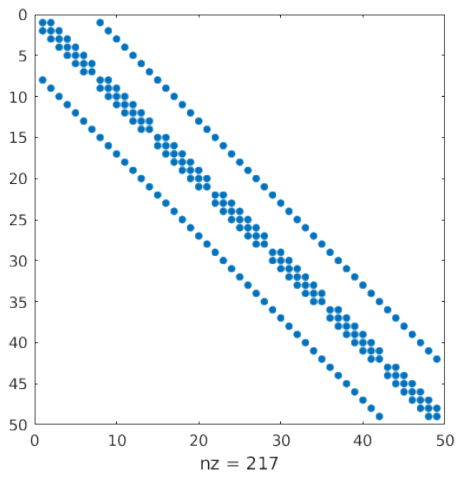


Figure 5.1: Sparsity pattern 2D Laplacian discretisation matrix

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.

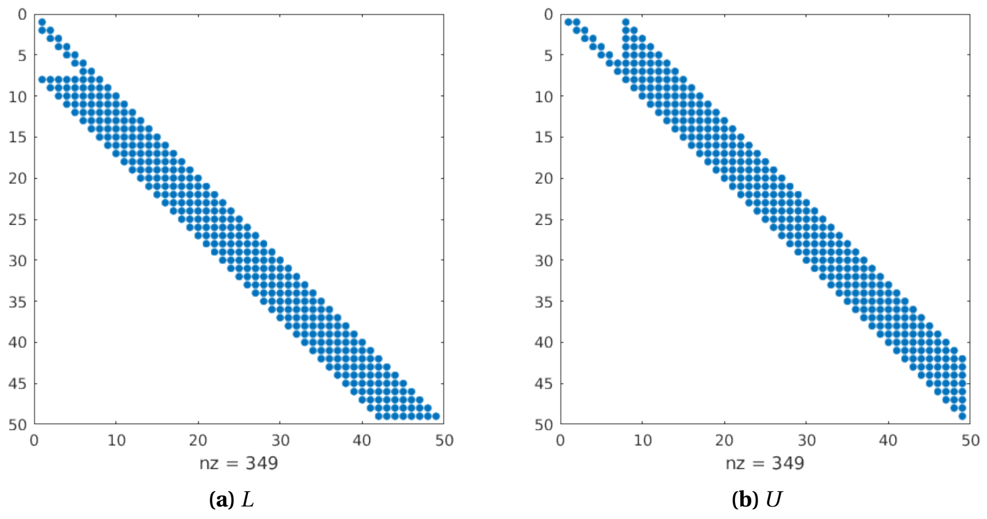


Figure 5.2: Sparsity pattern *LU*-decomposition 2D Laplacian discretisation

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. (5.58), and let $c_j^{(k)}$ denote the number of non-zero elements of column j of the same matrix. Now compute

$$\chi_{ij}^{(k)} = (r_i^{(k)} - 1)(c_j^{(k)} - 1) \tag{5.70}$$

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

$$\left|N_{22}^{(k)}\right|_{ij} \geq \delta \left|N_{22}^{(k)}\right|_{ab} \tag{5.71}$$

for all $k \leq a, b \leq n$ are considered [39].

5.3.3. Computing the Condition Number

As mentioned in Section 5.2.5, the definition of the condition number requires $\|M^{-1}\|_p$ to be known. Computing the inverse is expensive; it requires $\mathcal{O}(n^3)$ flops. 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 [16]. This method is based on the observation that

$$M\mathbf{u} = \mathbf{b} \qquad \Longrightarrow \qquad \|M^{-1}\|_\infty \geq \|\mathbf{u}\|_\infty / \|\mathbf{b}\|_\infty \tag{5.72}$$

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 [25].

Algorithm 5.5 Column Version Backward Substitution

```
$p(1:n) \leftarrow 0$
**for** $k = n \rightarrow 1$ **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. (5.72) 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 5.6 is a version of Algorithm 5.5 that implements this heuristic [25].

Algorithm 5.6 Triangular Condition Estimation

```
$p(1:n) \leftarrow 0$
**for** $k = n \rightarrow 1$ **do**
    Choose $d(k) \leftarrow 1$
    $y(k)^+ \leftarrow [d(k) - p(k)]/T(k,k)$
    $p(k)^+ \leftarrow p(1:k-1) + T(1:k-1,k)y(k)^+$
    Choose $d(k) \leftarrow -1$
    $y(k)^- \leftarrow [d(k) - p(k)]/T(k,k)$
    $p(k)^- \leftarrow p(1:k-1) + T(1:k-1,k)y(k)^-$
    **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 [16, 28]. This can be done using the following steps.

1. Apply the lower triangular version of Algorithm 5.6 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 [16].

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 5.6 can only be applied to triangular matrices. This motivates the following procedure.

1. Apply the lower triangular version of Algorithm 5.6 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., [28].

5.3.4. Iterative Refinement

Assume the solution $\mathbf{u}^{(k)}$ to Eq. (5.1) is obtained using the factorisation

$$PM = \hat{L}\hat{U} - \Delta M \tag{5.73}$$

in finite precision arithmetic. The goal is to improve the accuracy of the solution $\mathbf{u}^{(k)}$.

Algorithm 5.7 Iterative Refinement [25]

```
for $k = 1, \dots, k_{\max}$ do
    $\mathbf{r}^{(k)} \leftarrow \mathbf{b} - M\mathbf{u}^{(k)}$
    Solve $L\mathbf{y} = P\mathbf{r}^{(k)}$
    Solve $U\mathbf{z}^{(k)} = \mathbf{y}$
    $\mathbf{u}^{(k+1)} \leftarrow \mathbf{u}^{(k)} + \mathbf{z}^{(k)}$
end for
```

Assume, for simplicity, $P = I_n$. In exact arithmetic

$$M\mathbf{u}^{(k+1)} = M\mathbf{u}^{(k)} + M\mathbf{z}^{(k)} = \mathbf{b} - \mathbf{r}^{(k)} + \mathbf{r}^{(k)} = \mathbf{b} \tag{5.74}$$

However, in finite precision, things are a little more complicated. Applying Algorithm 5.7 results in

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \mathbf{z}^{(k)} \tag{5.75}$$

$$= \mathbf{u}^{(k)} + (\widehat{L}\widehat{U})^{-1}\mathbf{r}^{(k)} \tag{5.76}$$

$$= \mathbf{u}^{(k)} + (\widehat{L}\widehat{U})^{-1}[\mathbf{b} - M\mathbf{u}^{(k)}] \tag{5.77}$$

Additionally, the exact solution **u** satisfies

$$\mathbf{u} = \mathbf{u} + (\widehat{L}\widehat{U})^{-1}[\mathbf{b} - M\mathbf{u}] \tag{5.78}$$

Subtracting Eq. (5.77) from the identity above yields

$$\mathbf{u} - \mathbf{u}^{(k+1)} = [I - (\widehat{L}\widehat{U})^{-1}M](\mathbf{u} - \mathbf{u}^{(k)}) \tag{5.79}$$

So whether $\mathbf{u}^{(k)} \to \mathbf{u}$ depends on how well $\widehat{L}\widehat{U}$ approximates M. More precisely, if

$$\|I - (\widehat{L}\widehat{U})^{-1}M\| < 1 \tag{5.80}$$

then iterative refinement converges. In finite precision, iterative refinement usually stops yielding improvement after a few iterations.

5.4. The LAPACK Library

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. LAPACK will be considered here. 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 use an optimised BLAS implementation.

Mathematical operation	LAPACK routine
Matrix norm	`DLANGB`
Condition number	`DGBCON`
LU factorisation step	`DGBTRF`
LU solve step	`DGBTRS`
Iterative refinement	`DGBRFS`

Table 5.1: LAPACK routines.

Table 5.1 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 [25]. This usually leads to better performance on computers. The `DGBTRF` routine uses a block version of the *LU*-decomposition with partial pivoting, as explained in Section 5.3.2. The condition number estimation technique used in the `DGBCON` routine is the one explained in Section 5.3.3.

Bandwidth Minimisation

Since LAPACK uses the bandwidth format for matrix storage, the flops count of matrix operations depends heavily on the bandwidth. Therefore, bandwidth minimisation is of the utmost importance. For this reason a bandwidth minimisation heuristic is considered in combination with LAPACK. More on this will follow in Chapter 6.

6 Bandwidth Minimisation

In Section 5.3.1 it was mentioned that the LU-decomposition of a banded matrix M costs $\mathcal{O}(b_l b_u n)$ flops and the solve step involves $\mathcal{O}(n(b_l + b_u))$ flops. This immediately suggests a way of minimising the computational costs by minimising the bandwidth of the matrix M. In this chapter techniques for reordering the matrix M to minimise bandwidth are discussed.

6.1. Graphs and Matrices

For simplicity, in this section upper and lower bandwidth of matrices are summarised by the bandwidth.

Definition 6.1. Let $M \in \mathbb{R}^{n\times n}$ and let b_l and b_u denote the lower and upper bandwidth of M, respectively. The **bandwidth** of M is defined as

$$b(M) = \max\{b_l, b_u\}$$

The bandwidth minimisation problem for matrices can be formulated as follows.

$$\textit{Find a permutation matrix } P \textit{ which satisfies } P = \underset{Q}{\operatorname{argmin}}\, b(QMQ^\top). \tag{6.1}$$

In total there are $n!$ possible permutation matrices[1], so even for modest values of n checking all options is infeasible. Unfortunately, the bandwidth minimisation problem turns out to be NP-hard [26]. The good news is that there exist heuristics which tend to produce good results in practice. The bandwidth minimisation problem for matrices is usually considered from a graph-theoretic point-of-view. For this reason, some elementary concepts and definitions are required.

Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix and let $G = (V, E)$ be an undirected graph with $V = \{v_1, v_2, \ldots, v_n\}$ and $v_i v_j \in E$ if $M_{ij} \neq 0$ and $i \neq j$ [10]. In other words, ignoring the diagonal entries, the matrix M is the adjacency matrix of G. Let $f : V \to \{1, 2, \ldots, n\}$ be a **labelling** of the vertices of G.

Definition 6.2. The **bandwidth** of G under labelling f is defined by

$$b_f(G) = \max_{v_i v_j \in E} |f(v_i) - f(v_j)|$$

The bandwidth minimisation problem for graphs is given as follows.

$$\textit{Find a labelling } f \textit{ of } G \textit{ which satisfies } f = \underset{g}{\operatorname{argmin}}\, b_g(G). \tag{6.2}$$

The graph and matrix bandwidth minimisation problems are equivalent. In fact, the permutation matrix P and labelling f both correspond to the same permutation σ in S_n. An example will be given to illustrate that both formulations are equivalent.

[1] Permutation matrices can be defined in terms of the permutation group S_n which contains $n!$ permutations (see Section 7.1).

Example 6.3. Consider the following matrix

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{6.3}$$

with $b(M) = 5$. The corresponding graph G, with the (identity) labelling $f(v_i) = i$ shown in red, is given by

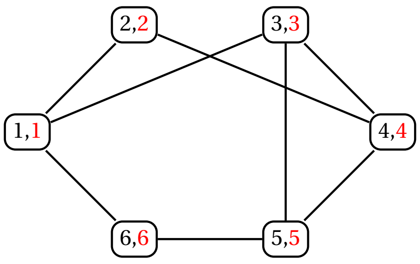


Figure 6.1: Labelled graph.

Indeed, $b_f(G) = 5 = b(M)$. The permutation $\sigma = (1\ 2\ 5\ 3\ 4\ 6)$, corresponding to the permutation matrix defined by $P_{\sigma(i)i} = 1$ and 0 elsewhere, results in

$$PMP^\top = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} M \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^\top = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{6.4}$$

where $b(PMP^\top) = 3$. The labelling g corresponding to σ is defined by $g(v_i) = \sigma(i)$ and results in the graph

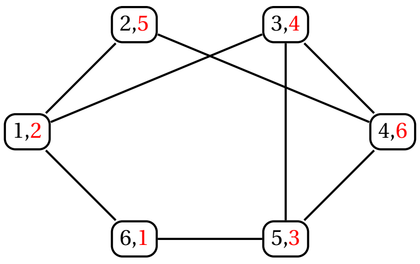


Figure 6.2: Relabelled graph

which also has $b_g(G) = 3$.

6.1.1. Reverse Cuthill-McKee

Since the bandwidth minimisation problem is NP-hard, it is necessary to resort to heuristics for trying to minimise the bandwidth within polynomial time. The most widely used heuristic is the Reverse Cuthill-McKee (RCM) [18, 19]. The version of the algorithm discussed here is the version as described by George and Liu [22]. This version includes an adjusted Gibbs-Poole-Stockmeyer method [23] of finding a node from which to start the labelling process. The GPS version of the RCM algorithm offers a good balance between computation time and bandwidth reduction and hence is a good candidate for usage in Wanda [34]. Other algorithms do not offer a larger bandwidth reduction for similar computation times (see, e.g., [33]). The basic idea, within the context of an undirected graph $G = (V, E)$, of the algorithm is to find a suitable starting node v and label the nodes in order of which they are visited during a breadth-first search starting from v.

First an algorithm is required to find a suitable starting node. Some notation is required. For $u, v \in V$ and $W \subseteq V$ let $\deg(v)$ denote the degree of v, $\mathrm{Adj}(W)$ the set of nodes adjacent to the nodes in W, and $d(u, v)$ the number of edges in the shortest path between u and v (if such a path exists).

Definition 6.4. For $v \in V$ the **eccentricity** $\varepsilon(v)$ is defined by

$$\varepsilon(v) = \max_{w \in V} d(v, w)$$

Definition 6.5. The **diameter** $\delta(G)$ of a graph $G = (V, E)$ is defined by

$$\delta(G) = \max_{v \in V} \varepsilon(v)$$

Definition 6.6. A vertex $v \in V$ is called a **peripheral node** if $\varepsilon(v) = \delta(G)$.

Definition 6.7. A vertex $v \in V$ is called a **pseudo-peripheral node** if for all $w \in V$ such that $d(v, w) = \varepsilon(w)$ it holds that $\varepsilon(v) = \varepsilon(w)$.

A pseudo-peripheral node is a node for which the eccentricity is (usually) close to the diameter, i.e., it can be seen as an approximation of a peripheral node. Finding peripheral nodes is expensive, as it would require the computation of the shortest paths between every pair of nodes in a graph. A pseudo-peripheral node offers a relatively cheap approximation. It has been observed that using pseudo-peripheral nodes as starting nodes for the breadth-first search often results in a lower bandwidth [22]. The next definitions introduces a concept that is used for finding a pseudo-peripheral node.

Definition 6.8. For a graph G and $v \in V$ the **level structure rooted at** v is defined as the partitioning

$$\mathscr{L}(v) = \{L_1(v), L_2(v), \ldots, L_k(v)\}$$

of V which satisfies

1. $L_1(v) = \{v\}$ and $L_2(v) = \mathrm{Adj}(L_1(v))$
2. $L_i(v) = \mathrm{Adj}(L_{i-1}(v)) \backslash L_{i-2}$ for $i = 3, 4, \ldots, k$

The **length** of $\mathscr{L}(v)$ is given as $\varepsilon(v)$ and the **width** is given as $\max\{|L_i(v)| : L_i(v) \in \mathscr{L}(v)\}$.

Example 6.9. The level structure rooted at v_6 in Example 6.3 is given by $\mathscr{L}(v_6) = \{L_1(v_6), L_2(v_6), L_3(v_6)\}$ where $L_1(v_6) = \{V_6\}$, $L_2(v_6) = \{v_1, v_5\}$ and $L_3(v_6) = \{v_2, v_3, v_4\}$. Both the depth and width of $\mathscr{L}(v_6)$ are equal to 3.

A level structure rooted at a pseudo-peripheral node with low degree tends to result in a structure with low width and hence high depth. The following algorithm describes a process for finding a good pseudo-peripheral node.

Algorithm 6.1 Finding a Pseudo-Peripheral Node

```
Initialise root vertex $r$
Generate level structure $\mathscr{L}(r) = \{L_1(r), L_2(r), \ldots, L_{k_r}(r)\}$
Choose node $v \in L_{k_r}(r)$ of minimum degree
**while** $\varepsilon(v) > \varepsilon(r)$ **do**
    $r \leftarrow v$
    Generate level structure $\mathscr{L}(r) = \{L_1(r), L_2(r), \ldots, L_{k_r}(r)\}$
    Choose node $v \in L_{k_r}(r)$ of minimum degree
**end while**
```

The algorithm above stops when the eccentricity stops growing and returns the latest root node r. Its running time is $\mathscr{O}(|E|)$ [22].

Example 6.10. Running Algorithm 6.1 with root v_6 returns the pseudo-peripheral node v_6. From the level structure in Example 6.9 the algorithm picks node v_2 and generates the level structure rooted at this node. But $\varepsilon(v_2) = \varepsilon(v_6) = 3$, hence the algorithm terminates and returns v_6. It is not hard to check that v_6 is a pseudo-peripheral node. In fact, all nodes in the graph are pseudo-peripheral nodes.

Note that the graph representation of a matrix need not be connected, hence ordering all nodes requires ordering each connected component. Now all the ingredients for describing the RCM algorithm are present.

Algorithm 6.2 Reverse Cuthill-McKee

```
for each connected component $G_C$ do
    Find a pseudo-peripheral node $v$ in $G_C$
    Initialise $L_1(v) \leftarrow \{v\}$ and $i \leftarrow 2$.
    while there exist unlabelled nodes do
        Construct level set $L_i(v)$
        Label all $u \in L_i(v)$ in increasing order of degree
        $i \leftarrow i + 1$
    end while
    Reverse the labelling
end for
```

In essence, RCM tries to minimise the bandwidth by making locally optimal choices. Assuming linear insertion is used for sorting the nodes in each level set in order of degree and a pseudo-peripheral starting node is provided, the running time of the algorithm is $\mathcal{O}(\eta|E|)$ where η is the maximum degree of any node [22].

Example 6.11. Running Algorithm 6.2 with pseudo-peripheral node v_6 on the graph in Example 6.3 finds the level structure as given in Example 6.9. At each level set, RCM labels the vertices in the set in increasing order of degree. After all nodes have been visited and labelled, it reverses the order found. This results in the following labelling.

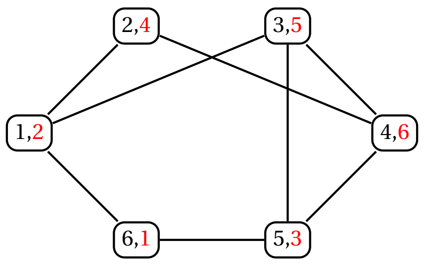


Figure 6.3: Labelled graph.

This labelling corresponds to the permutation $\sigma = (1\ 2\ 4\ 6)(3\ 5)$ and yields the permuted matrix

$$PMP^\top = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{6.5}$$

where $b(PMP^\top) = 3$.

6.1.2. Asymmetric Matrices

The previous sections only considered bandwidth within the context of symmetric matrices. The Wanda matrices, however, are asymmetric. There are three basic approaches to tackle this problem [42]. The main idea is to use M to obtain a symmetric matrix which is in turn used to obtain the permutation.

1. Use $M+M^\top$. The inequality $b(PMP^\top) \le b(PMP^\top + PM^\top P^\top) = b(P(M+M^\top)P^\top)$ shows that the bandwidth of the permuted matrix M cannot be worse than that of the symmetric matrix which is used to obtain the permutation.

2. Use the matrix
$$\hat{M} = \begin{bmatrix} 0 & M \\ M^\top & 0 \end{bmatrix} \tag{6.6}$$
which is the adjacency matrix of a bipartite graph which has a node for each row and each column of M. If $M_{ij} \ne 0$, then row node i is connected to column node j. Applying RCM to $\hat{M}$ results in the permuted matrix $P\hat{M}P^\top$ of the form
$$P\hat{M}P^\top = \begin{bmatrix} 0 & M_{11} & & & & & \\ M_{11}^\top & 0 & M_{21}^\top & & & & \\ & M_{21} & 0 & M_{22} & & & \\ & & M_{22}^\top & 0 & M_{23}^\top & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & M_{kl-1} & 0 & M_{kl} \\ & & & & & M_{kl}^\top & 0 \end{bmatrix} \tag{6.7}$$
where each M_{pq} is a submatrix of M corresponding to the row level set p and column level set q. Now permuting the rows of M by the ordered row level sets and the columns of M by the ordered column level sets yields the block bidiagonal form
$$M' = \begin{bmatrix} M_{11} & & & & \\ M_{21} & M_{12} & & & \\ & M_{32} & M_{33} & & \\ & & \ddots & \ddots & \\ & & & M_{kl-1} & M_{kl} \end{bmatrix} \tag{6.8}$$

3. Use the graph corresponding to $MM^\top$, where numerical cancellation in the multiplication is not taken into account. The resulting permutation is then used to order the rows of M and, subsequently, the columns are ordered according to their last non-zero entry.

The permutation matrix obtained from applying bandwidth minimisation to one of the matrices above usually also does a good job in minimising the bandwidth of the original matrix M. The third option is the most expensive to compute.

Example 6.12. Consider the asymmetric matrix.
$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{6.9}$$
with $b(M) = 5$. Applying RCM to the matrix $\hat{M}$ results in the following graph labelling (given in red).

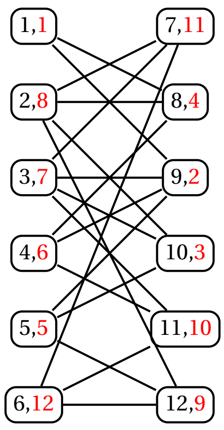


Figure 6.4: Labelled bipartite graph.

This corresponds with the (reversed) level sets

$$L_1(v_6) = \{v_1\},\ L_2(v_6) = \{v_8, v_9, v_{10}\},\ L_3(v_6) = \{v_2, v_3, v_4, v_5\},\ L_4(v_6) = \{v_7, v_{11}, v_{12}\},\ L_5(v_6) = \{v_6\}$$

The odd numbered levels sets are the row level sets and the even ones are the column level sets. The permuted matrix $\hat{M}$ of the form Eq. (6.7) is given as follows. Here the submatrices corresponding to the row and column level sets are highlighted in blue (cf. Eq. (6.7)).

$$P\hat{M}P^{\top} = \begin{bmatrix} 0&1&0&1&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&1&1&0&0&0&0&0\\ 0&0&0&0&1&0&1&1&0&0&0&0\\ 1&0&0&0&0&1&0&1&0&0&0&0\\ 0&1&1&0&0&0&0&0&1&0&0&0\\ 0&1&0&1&0&0&0&0&0&1&0&0\\ 0&1&1&0&0&0&0&0&0&1&1&0\\ 0&0&1&1&0&0&0&0&1&0&1&0\\ 0&0&0&0&1&0&0&1&0&0&0&1\\ 0&0&0&0&0&1&1&0&0&0&0&1\\ 0&0&0&0&0&0&1&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&1&1&0 \end{bmatrix} \tag{6.10}$$

The permutation on $\hat{M}$ is given by

$$\sigma = \begin{pmatrix} 1&2&3&4&5&6&7&8&9&10&11&12\\ 1&8&7&6&5&12&11&4&2&3&10&9 \end{pmatrix} \tag{6.11}$$

This notation should be interpreted as $\sigma(1) = 1$, $\sigma(2) = 8$, etc. From the definition of $\hat{M}$ and Eq. (6.7) it can be seen that for the row permutation the ordered row level sets and as the column permutation the ordered column level sets should be taken.

$$\sigma_{\text{row}} = (1)(6)(2\ 5)(3\ 4),\ \sigma_{\text{col}} = (11)(7\ 9\ 8\ 10\ 12) \tag{6.12}$$

By 'translating' σ_{col} back to a permutation on $1, 2, \ldots, 6$ and applying the permutations to the rows and columns of M, respectively, the resulting matrix M' of the form Eq. (6.8) and with $b(M') = 3$ is given by

$$M' = \begin{bmatrix} 1&0&1&0&0&0\\ 1&1&0&1&0&0\\ 1&0&1&0&1&0\\ 1&1&0&0&1&1\\ 0&1&1&1&0&1\\ 0&0&0&1&1&1 \end{bmatrix} \tag{6.13}$$

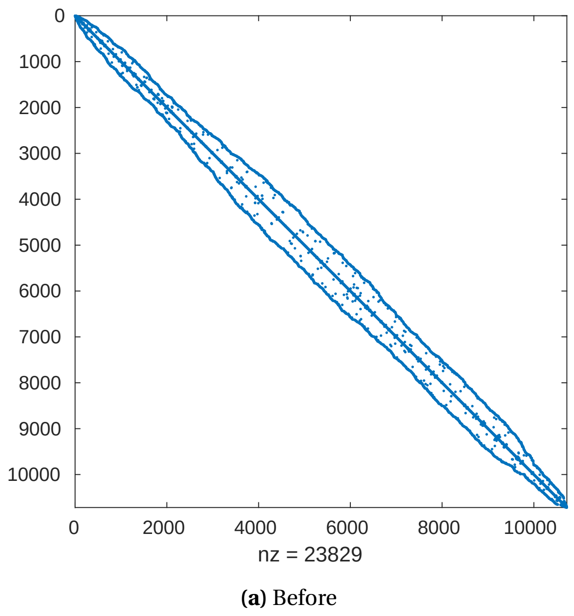
6.1.3. Results

The Fortran implementation used is the one provided by Burkardt [13]. Since the goal of bandwidth reduction is to minimise the operation count during the LU-factorisation, bandwidth reduction is mostly relevant for large matrices. For this reason, results are shown only for the large test cases. The results in Table 6.1 are obtained using the Matlab version of the RCM implementation provided by Burkardt.

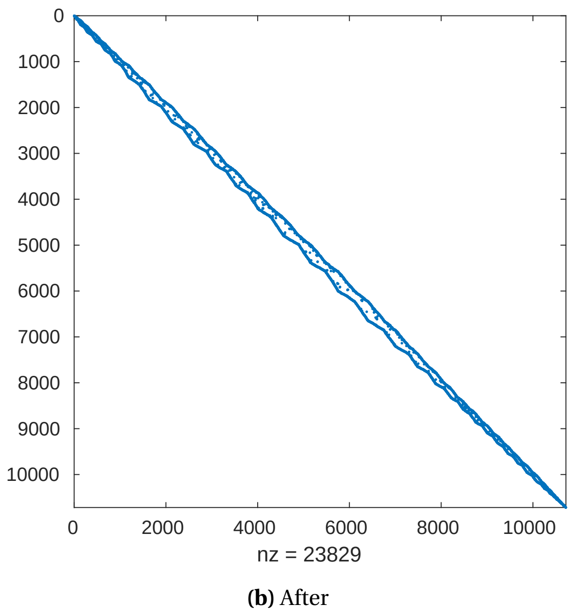
Test case	n	nnz	b	$M+M^\top$	$\hat{M}$	$MM^\top$
Filter	3516	8487	204	64	63	63
Noord-Holland 1	6342	14264	310	177	14	16
Noord-Holland 2	10718	23829	634	368	249	249

Table 6.1: RCM results.

Table 6.1 shows the results of applying RCM to the test cases using the strategies explained in Section 6.1.2. The original bandwidth is denoted by b. The results clearly indicate that applying RCM to $\hat{M}$ and $MM^\top$ yield the best results. A major downside of using the $MM^\top$ matrix is that it is expensive to obtain as it requires a matrix-matrix multiplication. Therefore, the best strategy is applying RCM to $\hat{M}$.



(a) Before



(b) After

Figure 6.5: Sparsity pattern NH2 before and after applying RCM.

As illustrated by Fig. 6.5, the permuted matrix not only tends to have a smaller bandwidth, but the upper bandwidth also tends to be smaller than the lower bandwidth. This is because minimising the bandwidth of $\hat{M}$, by definition of $\hat{M}$, gives a higher priority to minimising the upper bandwidth of M.

Case	b_l	b_u
Filter	63	50
Noord-Holland 1	14	10
Noord-Holland 2	249	180

Table 6.2: Upper and lower bandwidth after applying RCM.

Indeed, for all three test cases the lower bandwidth is larger than the upper bandwidth. Furthermore, applying RCM to the steady state matrix for each of the cases as presented in Section 8.3 shows that 53 out of 55 cases have a lower bandwidth which is larger than or equal to the upper bandwidth.

The reason why this is of importance is the following. Suppose M has a lower and upper bandwidth of b_l and b_u, respectively. Computing the LU-decomposition with row interchanges (partial pivoting) results in L keeping the b_l lower bandwidth and U having upper bandwidth $b_l + b_u$ [25]. So the total bandwidth equals $2b_l + b_u$. Now, if the LU-decomposition of $M^\top$ is computed, the total bandwidth is given by $b_l + 2b_u$. Since usually, though not necessarily, $b_l > b_u$, it is more efficient to solve the linear system using the LU-decomposition of $M^\top$. Table 6.2 shows that the difference can be significant. LAPACK has the built-in capability of solving the transposed system of equations using a given LU-decomposition, so their is no overhead for using the transposed matrix. This motivates the following procedure for solving $M\mathbf{u} = \mathbf{b}$.

1. Obtain P and Q by applying RCM to $\hat{M}$, and permute the transposed matrix $N = Q^\top M^\top P^\top$.
2. Obtain the LU-decomposition of N, i.e., $N = LU$ (ignoring the pivoting).
3. Solve $U^\top L^\top \mathbf{v} = P\mathbf{b}$.
4. Obtain the solution to $M\mathbf{u} = \mathbf{b}$ by setting $\mathbf{u} = Q\mathbf{v}$ (i.e. de-permute $\mathbf{v}$).

This tends to be less computationally expensive than using the LU-decomposition of M.

7

Structural Singularities

The methods in Chapter 5 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 [40].

Definition 7.1. 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.

The following example will illustrate the use of this concept.

Example 7.2. Consider again the system of Section 3.1.1.

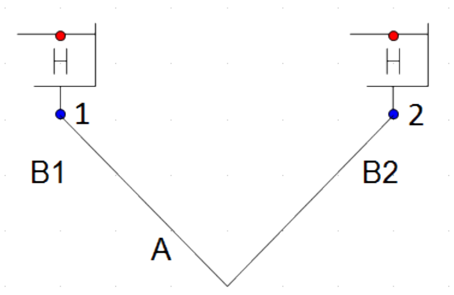


Figure 7.1: Steady state singularity.

This system is described by the following system of equations.

$$\begin{cases} H_1 &= c_1 \\ H_1 &= H_A \\ Q_A + Q_1 + Q_2 &= 0 \\ Q_A &= 0 \\ H_2 &= H_A \\ H_2 &= c_2 \end{cases} \tag{7.1}$$

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

$$M_{ij} = \begin{cases} 1, & \text{if variable } j \text{ is present in equation } i \\ 0, & \text{otherwise} \end{cases} \tag{7.2}$$

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

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.3}$$

when using the variable ordering $(Q_1, H_1, Q_A, H_A, Q_2, H_2)$. 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. So a different approach to detecting if a system is undetermined is by determining whether it is structurally singular.

Example 7.2 claims that the system of Section 3.1.1 is not only singular, but also structurally singular. Intuitively, this makes sense as there are only two equations available to solve the three Q_i's in the system. This chapter will describe a graph-theoretic characterisation of structurally singular matrices. Furthermore, using the test cases introduced in Section 3.4, it will be illustrated that the singular matrices produced by Wanda are in fact structurally singular. These examples are also used to illustrate that this characterisation can be used to detect where an additional prescription of H (or p or T) is required. Finally, a practical method to detect structural singularities will be discussed.

7.1. Graph-Theoretic Characterisation

For $M \in \mathbb{R}^{n\times n}$, a bipartite graph representation will be used, which is defined as follows. Let $G = (U \cup V, E)$ be a bipartite graph where $U = \{u_1, u_2, \ldots, u_n\}$ represents the rows of M, $V = \{v_1, v_2, \ldots, v_n\}$ the columns and $u_i v_j \in E$ if $M_{ij} \neq 0$. Now consider the following definition [17].

Definition 7.3. A **matching** in a graph $G = (V, E)$ is a set $F \subseteq E$ of edges such that every $v \in V$ is incident with at most one edge in F. A **perfect matching** is a matching $F \subseteq E$ such that every $v \in V$ is incident with exactly one edge in F.

Note that a perfect matching need not be unique. In fact, a matching of any size need not be unique.

The following theorem states the characterisation of structurally singular matrices based on the bipartite graph representation.

Theorem 7.4. *Let $M \in \mathbb{R}^{n\times n}$ and let $G = (V \cup U, E)$ be its bipartite graph representation. M is structurally singular if and only if there does not exist a perfect matching in G.*

Because of the insightful nature and beauty of the proof, a sketch will be given here. A more formal proof can be found in [37] or [38]. The proof makes use of the following definition of the determinant, where $M \in \mathbb{R}^{n\times n}$.

$$\det M = \sum_{\sigma \in S_n} \varepsilon(\sigma) \prod_{i=1}^{n} M_{\sigma(i)i} \tag{7.4}$$

Here S_n denotes the permutation group containing all permutations $\tau : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. For example, for $n \geq 3$ the permutation $(1\ 3\ 2) \in S_n$ represents the map $1 \mapsto 3$, $3 \mapsto 2$ and $2 \mapsto 1$. S_n contains $n!$ elements. Each element $\tau \in S_n$ can be non-uniquely written as

$$\tau = (k_1\ k_2)(k_3\ k_4)\ldots(k_{m-1}\, k_m), \tag{7.5}$$

the product of $m/2$ transpositions. If $m/2$ is even $\varepsilon(\tau) = 1$ and $\varepsilon(\tau) = -1$ if $m/2$ is odd. Although the transposition expansion of τ is not unique, the parity $m/2$ is.

Example 7.5. Consider the matrix

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \tag{7.6}$$

The determinant is defined as

$$\det M = \underbrace{M_{11}M_{22}M_{33}}_{(1)(2)(3)} - \underbrace{M_{11}M_{32}M_{32}}_{(1)(2\ 3)} - \underbrace{M_{12}M_{21}M_{33}}_{(1\ 2)(3)} + \underbrace{M_{12}M_{31}M_{23}}_{(1\ 2\ 3)} + \underbrace{M_{13}M_{21}M_{32}}_{(1\ 3\ 2)} - \underbrace{M_{13}M_{31}M_{22}}_{(1\ 3)(2)} \tag{7.7}$$

The graph G corresponding to the matrix above is given by

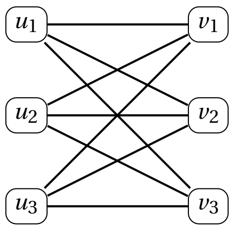


Figure 7.2: Bipartite graph representation.

From the example above it can be seen that each product corresponding to a $\tau \in S_n$ in the determinant formula represents a perfect matching in the graph G. The following observations can be made.

- In general, terms in the determinant formula do not cancel due to the uniqueness of each permutation.
- If $M_{ij} = 0$, then $u_i v_j \notin E$.

From these two observations it can be concluded that if $\det M = 0$ for any value assignment of the M_{ij}'s, then no perfect matching in G exists. And if no perfect matching in G exists, then $\det M \equiv 0$ for any value assignment of the M_{ij}'s. The theorem thus provides a powerful tool for the detection of structurally singular matrices. As the theorem does not rely on any finite precision numbers, there is no ambiguity involved such as in case of condition number estimation for detecting structurally singular matrices.

7.2. Application to Wanda

The question now is how this result can be applied to Wanda. Consider the system of (non-linear) equations

$$\begin{cases} u_1(v_1, v_2, \ldots, v_n) = 0 \\ u_2(v_1, v_2, \ldots, v_n) = 0 \\ \vdots \\ u_n(v_1, v_2, \ldots, v_n) = 0 \end{cases} \tag{7.8}$$

describing a pipeline system in Wanda. The Jacobian $J_{\mathbf{u}} \in \mathbb{R}^{n \times n}$ of this system is given by

$$[J_{\mathbf{u}}(\mathbf{v})]_{ij} = \frac{\partial u_i}{\partial v_j}(\mathbf{v}) \tag{7.9}$$

Observe that $[J_{\mathbf{u}}(\mathbf{v})]_{ij} = 0$ if equation u_i does not contain variable v_j, so the Jacobian provides information about the system of non-linear equations that are solved. It is used in the Newton-Raphson method in Wanda to solve the system of equations. Checking for structural singularity in the Jacobian then amounts to checking whether a necessary condition for the existence of a unique solution in the linearised system, and therefore also in the non-linear system, is met. The systems can, of course, still be numerically singular. It will be illustrated using examples that the singular matrices resulting from faulty systems in Wanda are in fact structurally singular. Similar applications are given in [24].

The matchings that will be given in the examples can be proved to be maximum size matchings using König's theorem [17].

Definition 7.6. Let $G = (V, E)$. $C \subseteq V$ is called a **cover** if every $e \in E$ has at least one end point in C.

Theorem 7.7 (König's theorem)**.** *For a bipartite graph G,*

$$\max\{|F| : F \textit{ a matching}\} = \min\{|C| : C \textit{ a cover}\}$$

Hence, given a matching F, to prove F is indeed maximum a cover C of the same size is required. Consider the bipartite graph $G = (U \cup V, E)$. Let F be a maximum size matching in G and let $W \subseteq U$ denote the set of unmatched vertices. Now let $X \subseteq U \cup V$ denote the set of vertices reachable from some $w \in W$ via an F-**alternating path**, that is, a path alternating between edges in F and edges not in F. Finally, let $Y = W \cup X$. Then

$$C = (U \setminus Y) \cup (V \cap Y) \tag{7.10}$$

is a minimum size cover [17]. This way of constructing C will be used to prove the optimality of F in the given examples.

Example 7.8. Consider the system as given in Example 7.2. The Jacobian of the system is given by

$$J_{\mathbf{u}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.11}$$

using the variable ordering $(Q_1, H_1, Q_A, H_A, Q_2, H_2)$. This results in the following graph G.

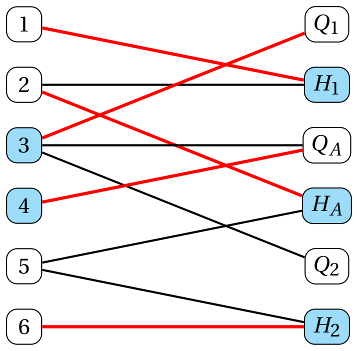


Figure 7.3: Bipartite graph representation.

Here a maximum matching F of size 5 is given in red. The cover C consisting of the blue vertices proves the optimality of F. It can be concluded that the Jacobian is structurally singular.

The example above suggests that the variable nodes which are not incident to an edge in the matching correspond to the undetermined ones. If either some Q_i or H_i is unmatched, it means there are not enough independent equations to determine either the Q_j's or H_j's. So not only does computing a maximum matching in G show whether or not the Jacobian is structurally singular, but it also shows where one should adjust the equations to make the matrix non-singular. Of course, the perfect matching in the graph above is not unique, but the unmatched variable is always found in the underdetermined part. In the particular example above, Q_2 is unmatched, showing that the problem resides with the Q_i's which are all connected.

Example 7.9. Consider again the system as given in Section 3.1.2

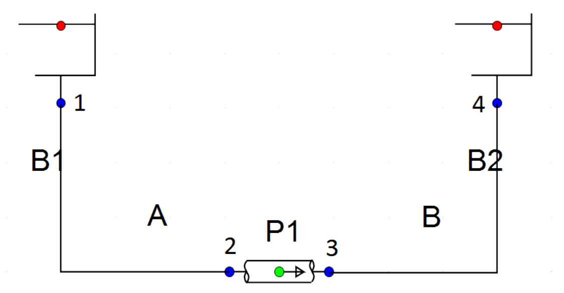


Figure 7.4: Small pipeline system.

which is described by the system of equations

$$
\begin{cases}
Q_1 &= c_1 \\
Q_A + Q_1 - Q_2 &= 0 \\
Q_A &= 0 \\
H_A &= H_1 \\
H_A &= H_2 \\
H_2 - H_3 &= \dfrac{\lambda L}{8A/O} \dfrac{Q_2|Q_2|}{A^2 g} \\
Q_2 &= Q_3 \\
H_B &= H_3 \\
H_B &= H_4 \\
Q_B &= 0 \\
Q_B + Q_3 + Q_4 &= 0 \\
Q_4 &= c_4
\end{cases}
\tag{7.12}
$$

The bipartite graph corresponding to the Jacobian of the system above is in Fig. 7.5a.

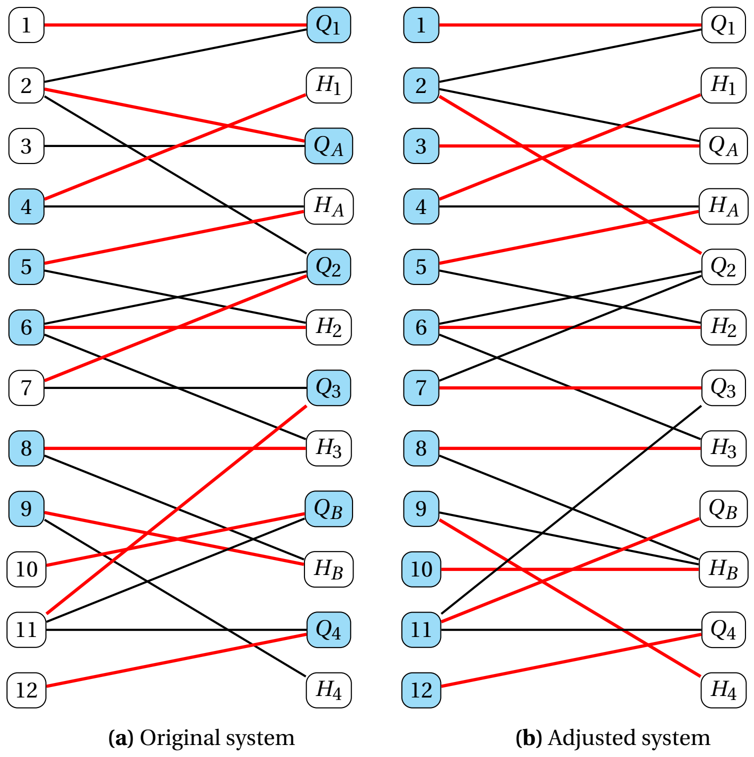


Figure 7.5: Q Boundary with Pipe System.

The red edges denote the edges of a maximum matching of size 11 in Fig. 7.5a. The vertex cover in blue is also of size 11, proving the optimality of F. Again it can be concluded that the Jacobian is structurally singular. In this system H_4 is undetermined. From H_4 the first H-node that can be reached via edges in E is B. Prescribing H_B will result in a matrix that is not structurally singular. Fig. 7.5b shows that the adjusted system does contain a perfect matching, hence the corresponding Jacobian is not structurally singular. It could still be singular, but structural singularities form the primary issue in Wanda.

Example 7.10. Consider again the system presented in Section 3.1.2.

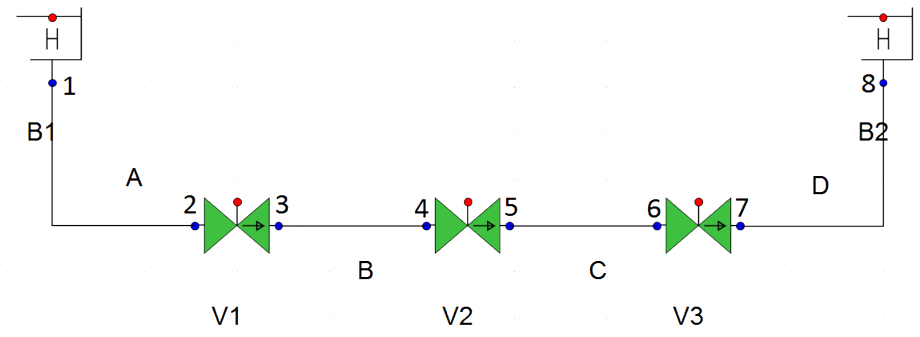


Figure 7.6: System leading to singular matrix due to phase transitions.

Assume all valves are open at $t = 0$ and at $t = 1$ valves V_1 and V_3 are closed. The system consists of 48 equations and unknowns. At $t = 1$, the graph corresponding to this situation consists of disconnected parts. The H_i's between the valves are disconnected from the H_i's outside the valves. Due to its size the graph will be omitted here. Using Matlab a matching of size $|F| = 23$ is obtained, which is of maximum size. H_6 is unmatched and hence undetermined. From H_6 the first H-node that can be reached is C. Prescribing H_C at $t = 1$ with the solution at $t = 0$ gives a fully determined system and the corresponding graph does contain a perfect matching.

7.3. The Hopcroft-Karp Algorithm

The matchings in the examples, except for the last one, in the previous section were obtained by inspection. Theorem 7.7 was used to prove the optimality of the matchings. In the Wanda setting an algorithm for computing a maximum size matching is required. Bipartite graphs, as a special case of general graphs, are favourable in the sense that their structure allows for (relatively) simple methods of computing maximum matchings. Two (rather old) Fortran implementations for calculating maximum size matchings were found: *GRAFPACK* [12] and *LAUPACK* [31]. Both packages are freely available and use $\mathcal{O}(|V|^3)$ algorithms [32, 41]. After some tests it became clear that both packages offer insufficient performance. For this reason, an alternative is required. The algorithm offering the best known theoretic performance is the Hopcroft-Karp algorithm with $\mathcal{O}(\sqrt{|V|}|E|)$ run time [17, 29]. Another advantage of this algorithm is its (relative) simplicity, which makes it an excellent candidate for an actual implementation in Fortran.

7.3.1. Basic Ideas

In order to explain how the algorithm works a few concepts and definitions are required. All definitions and theorems are obtained from [17].

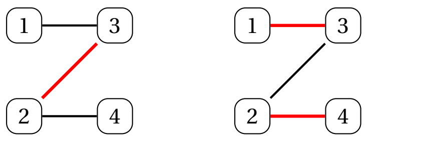
Definition 7.11. Let $G = (V, E)$ be a graph, $v \in V$ and F a matching in G. The vertex v is called F-**covered** if there exists an $e \in M$ such that $e = vw$. v is called F-**exposed** if no such edge exists.

Definition 7.12. Let G be a graph and F a matching in G. A path P is called F-**augmenting** if it is an F-alternating path and both its end nodes are distinct and F-exposed.

Theorem 7.13 (Berge)**.** *A matching F in a graph G is maximum if and only if there is no F-augmenting path.*

This theorem suggests a way of finding a maximum matching in a graph, namely, by iteratively finding augmenting paths and thereby iteratively expanding the matching, until no more augmenting path exists. Consider a matching F in a graph G and an F-augmenting path P with end nodes v, w. Then $M\Delta E(P)$, where Δ denotes the symmetrical difference, is a matching that covers all nodes covered by F, as well as v and w, i.e., $F\Delta E(P)$ is a larger size matching.

Example 7.14. Fig. 7.7a shows a small network with a matching of size 1 and an F-augmenting path. After augmentation, as depicted in Fig. 7.7b, the graph contains a matching of size 2.



(a) Original **(b)** Augmented

Figure 7.7: Augmenting path.

The question now is how to find F-augmenting paths. Given a matching F in a graph G, the first step is to look for an F-exposed node r. Starting from this node the idea is to built an F-**alternating tree**. Let $A, B \subseteq V$ such that each node in A is at the other end of an **odd**-length F-alternating path starting at r, and each node in B is at the other end of an **even**-length F-alternating path starting from r. These sets are built up iteratively. Now, if there exists an edge $vw \in E$ such that $v \in B$ and $w \notin A \cup B$ is F-exposed, then the F-alternating path from r to v plus the edge vw forms an F-augmenting path. Starting from $A = \varnothing$ and $B = \{r\}$ the rule

$$\text{If } vw \in E,\ v \in B,\ w \notin A \cup B,\ wz \in M, \text{ then add } w \text{ to } A \text{ and } z \text{ to } B.$$

produces such sets A and B. The set $A \cup B$ and the corresponding edges form a tree T with root r.

Example 7.15. From the graph depicted in Fig. 7.3 in Example 7.8 the following tree can be constructed starting from the F-exposed node 5.

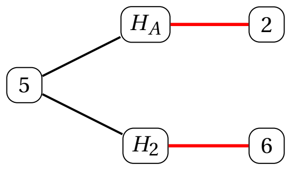


Figure 7.8: F-alternating tree.

No edge vw exists in the graph depicted in Fig. 7.3 such that $v \in B$ and $w \notin A \cup B$ is F-exposed, i.e., no augmenting path exists. Therefore, the matching is of maximum size.

7.3.2. The Algorithm

Hopcroft-Karp is based on the idea of building F-alternating trees from F-exposed nodes. From a bird's-eye view the pseudocode of the algorithm is given as follows.

Algorithm 7.1 Hopcroft-Karp

```
Initialise $F \leftarrow \varnothing$.
while $F$ is not a maximum matching do
    Find $\{P_1, \dots, P_k\}$, a maximal set of vertex-disjoint shortest $F$-augmenting paths.
    Set $F \leftarrow F\Delta(E(P_1) \cup \dots \cup E(P_k))$
end while
```

Let $G = (U \cup V, E)$ be a bipartite graph and F a matching in G. In Algorithm 7.1 it can be seen that at each phase of the while loop the Hopcroft-Karp algorithm comes up with a maximal set of vertex-disjoint shortest

F-augmenting paths. The algorithm uses a breadth-first search starting from each F-exposed $u \in U$ to build the F-alternating trees. It then uses a depth-first search to augment the matching along the augmenting path. It does this in such a way that in each phase this set of augmenting paths is maximal and the paths are vertex-disjoint.

The full pseudocode of the actual implementation in Fortran is given in Algorithms A.1 to A.3 and is based on the Python code from the *NetworkX* package [21, 27]. Instead of explicitly building the sets A and B, the actual implementation of Hopcroft-Karp uses a distance function to construct the trees. It includes an extra node, the null vertex, to which all $v \in V$ are connected. Note that the distance is only defined for the null vertex and for all $u \in U$. Starting from an F-exposed node r, the breadth-first search labels nodes $u \in U$ in the F-alternating tree with a distance equal to the number of even nodes between r and u, plus 1. If at some point the breadth-first search ends at some $v \in V$, the distance of the null-vertex is set equal to the distance of the vertex paired with v, plus 1. Nodes are only considered in the breadth-first search if their distance from the root is smaller than the current distance of the null vertex. After building the F-alternating trees, the depth-first search looks for shortest F-augmenting paths in the trees. If the distance of the null vertex is equal to the distance of the current path explored by the depth-first search plus 1, then the F-augmenting path found is a shortest one and hence the matching is augmented along this path. After augmentation the vertices in that path cannot be considered again in subsequent depth-first searches, because the distance of the node paired with a node $v \in V$ on the path now equals the distance of u. This ensures the vertex-disjointness of the augmenting paths. If no shortest augmented path can be found starting from a vertex u, the distance of u will be set to infinity in order to avoid considering that node again. The maximality of the set of vertex disjoint shortest F-augmenting paths is ensured by the fact that depth-first search will find a shortest augmenting path from each F-exposed $u \in U$, if it exists, and hence no more augmenting paths can be added.

It can be proven that the algorithm terminates after at most $2\sqrt{|V|}$ phases. Furthermore, each phase can be carried out in $\mathcal{O}(|E|)$ time. Therefore, the algorithm achieves $\mathcal{O}(\sqrt{|V|}|E|)$ running time [17].

7.3.3. Implementation

A Matlab implementation of the algorithm showed promising performance, hence the algorithm was implemented in Fortran. As mentioned in Example 5.1, the matrices in Wanda are stored in the coordinate format which includes the dimension n, the number of non-zero elements m and the indices and values of the non-zero elements. The nice thing about this is that the indices of the non-zero elements form the edge-list of the bipartite graph. The available data hence does not require transformation to represent the bipartite graph. The Hopcroft-Karp implementation takes as input n, m and the row and column indices of the non-zero elements denoted by I and J, respectively. In order to be able to efficiently determine which nodes are adjacent to a $u \in U$, the (unsorted) coordinate format I, J is transformed into the **compressed sparse row (CSR) format)** IA, JA using *SPARSKIT2* [43, 44].

Example 7.16. The array IA is of length $n+1$ and is defined by $IA(1) = 0$ and $IA(k) = IA(k-1) + \text{nnz}(M(k-1,:))$ for $2 \leq k \leq n+1$. The array JA is of size m and contains the column indices sorted by row.

The CSR format of the matrix in Example 5.1 is given by

$$IA = [0\ 1\ 3\ 5] \qquad \text{and} \qquad JA = [2\ 1\ 3\ 2\ 3] \tag{7.13}$$

For any node $1 \leq u \leq n$ in U, the array $JA((IA(u)+1) : IA(u+1))$ contains the nodes in V which are adjacent to u.

To speed up the algorithm, an initial matching is obtained using a Greedy type algorithm, i.e., an algorithm that makes a locally optimal choice at each stage. In this case, the algorithm simply checks for each $u \in U$ if it there exists a $uv \in E$ such that v is F-exposed and, if so, it adds uv to F.

Algorithm 7.2 Initial Greedy Matching

```
Initialise $F \leftarrow \varnothing$
for $u$ in $U$ do
    for $v$ adjacent to $u$ do
        if $v$ is $F$-exposed then
            $F \leftarrow F \cup \{uv\}$
            break
        end if
    end for
end for
```

7.3.4. Preliminary Results

The algorithm is able to detect the structural singularity of each of the test problems as given in Table 3.1. The large test cases, as given in Table 3.2, all passed the test, as expected.

8 Results

This chapter first gives an overview of the new solution method. Then comparisons are made between the original IMSL solver and several variants of the newly implemented LAPACK solver in terms of robustness, solution accuracy and run time. The new solution method will be evaluated based on these results.

8.1. New Solution Method

The new solution method involves two phases.

Before attempting to solve the system of linear equations, at the first iteration of the steady / transient flow simulation, or after a phase change occurring during the simulation, a check is done to detect whether the matrix is singular by estimating the condition number or by finding a maximum size matching in the corresponding graph. If the matrix is singular, try to fix the error. If no fix is possible, break off the simulation and return an appropriate error message.

The second phase is the actual solving of the system of linear equations. For each iteration, the system of linear equations $M\mathbf{u} = \mathbf{b}$ is solved using the following steps:

1. Generate row and column permutations P and Q to reduce the matrix bandwidth using the RCM algorithm on the bipartite matrix
$$\hat{M} = \begin{bmatrix} 0 & M \\ M^\top & 0 \end{bmatrix}.$$
Since often $b_u < b_l$ for PMQ and partial pivoting increases the bandwidth of the LU-decomposition by b_l it is more efficient to compute the LU-decomposition of the transposed matrix $N = Q^\top M^\top P^\top$.

2. Convert the matrix from coordinate format to band matrix format.

3. Obtain the LU-decomposition of N, i.e. $N = LU$ (ignoring the permutation matrix resulting from pivoting).

4. Permute the right-hand side vector $\mathbf{b}$ to $P\mathbf{b}$ and obtain the solution $\mathbf{v}$ to $U^\top L^\top \mathbf{v} = P\mathbf{b}$.

5. Apply iterative refinement to $\mathbf{v}$: Compute $\mathbf{r} = P\mathbf{b} - N^\top \mathbf{v}$; solve $U^\top L^\top \mathbf{w} = \mathbf{r}$ for $\mathbf{w}$; set $\mathbf{v}_{\text{new}} = \mathbf{v} + \mathbf{w}$. It is possible to do this step multiple times.

6. Obtain the solution $\mathbf{u}$ to $M\mathbf{u} = \mathbf{b}$ by setting $\mathbf{u} = Q\mathbf{v}_{\text{new}}$ (i.e. de-permute $\mathbf{v}_{\text{new}}$).

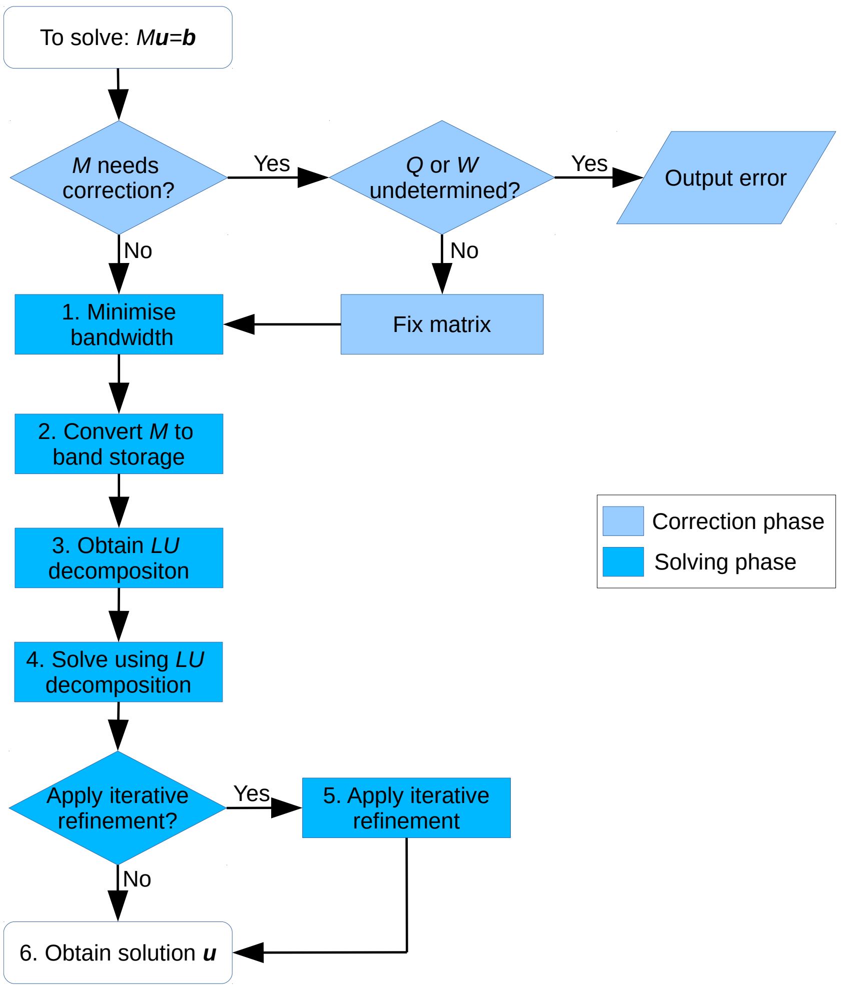


Figure 8.1: Solver steps flowchart

Fig. 8.1 is a visual representation of the steps given above. The correction phase is only executed at the beginning of a steady or unsteady simulation and after a phase change. Iterative refinement is given as an option here since the idea is to make the number of iterative refinement iterations an option in the accuracy window in Wanda. By default this will be set to one iteration.

8.1.1. Singularity Detection

The *LU*-decomposition subroutine can detect singular matrices via pivots being exactly zero. The singularity is detected for each of the test cases. Since row operations, used in computing the *LU*-decompositions, do not preserve the structural singularity of a matrix, and numerically singular matrices do not necessarily have an exactly zero pivot, this does not seem to be a good strategy overall. Nevertheless, to see how the different methods of singularity detection compare in terms of performance, the internal detection is tested as well. The other two methods use condition number estimation and maximum size matchings, respectively.

8.1.2. Routines Used

The great thing about the new solution method is that only relatively few routines are required. This makes it a lightweight and versatile solution method, which should be easily maintainable.

For computing a maximum size matching self-written code is used (see Section 7.3.3). The LAPACK routines used are given in Table 5.1. There is also a LAPACK iterative refinement routine available, however, this routine will not be used. Implementing iterative refinement using `DGBTRS` and the BLAS routines `DGBMV` and `DAXPY` is more efficient.

Furthermore, for bandwidth reduction the RCM routines provided by Burkardt are used [13]. For the matrix conversion from coordinate format to CSR format an adjusted conversion routine based on a routine from SPARSKIT2, provided by Saad, is used [43, 44].

Both RCM and SPARSKIT2 are licensed under the GNU LGPL. LAPACK uses the Modified BSD License.

Heat Module Routines

The heat module still depends on the `DNEQNF` and `NEQNF` IMSL routines for computing the root of a system of non-linear equations. The IMSL manual mentions that these routines are based on the `HYBRD1` MINPACK routine [3]. For this reason these routines are replaced by the double and single precision version of `HYBRD1`, respectively [36]. After this modification, Wanda is free of IMSL routines.

8.2. Robustness

Robustness can be improved by using the maximum size matching for detecting which variables are undetermined, if any. The current solution method only has a routine in place that checks whether the quantities H in liquid and p and T in heat are undetermined. Not only can the new method do this much faster, but it can also detect singularities in the Q (liquid) and W (heat) quantities. This information can be used to output an error which shows the user where the problem resides in the network. With some physics and Wanda knowledge these problems can often be resolved by adding or removing some component in the system. This solves the issue explained in Chapter 3 where Q being undetermined resulted in Wanda either crashing or getting stuck in an infinite loop.

8.2.1. Matrix Correction

In Section 7.2 it was argued that unmatched variables correspond to undetermined variables. This creates an opportunity to replace the current matrix fixing routine by a more efficient and less complex one. At the start of the steady and unsteady simulation and after each phase change a matrix correction algorithm is required to correct the matrix, if necessary.

There seem to be two cases in which a matrix correction is required. First, in case of a structurally singular matrix, and secondly, in case of an isolated loop in the system.

Loops in Pipeline Systems

Not only is it necessary to fix the matrix if it is structurally singular, it is also necessary to prescribe the H (or p) if a pipeline system contains an isolated loop. That is, a loop that is not connected to any other components where H (or p) is known. This will be illustrated using the following example.

Example 8.1. Consider the following system.

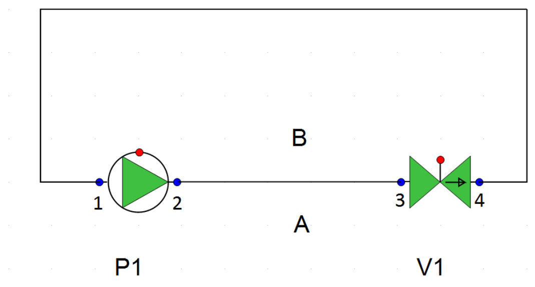


Figure 8.2: Pump loop.

This system is governed by the following equations.

$$\begin{cases} H_2 - H_1 &= c_1 Q_1 |Q_1| \\ Q_1 &= Q_2 \\ -Q_1 + Q_B + Q_4 &= 0 \\ Q_2 + Q_A - Q_3 &= 0 \\ H_1 &= H_B \\ H_2 &= H_A \\ Q_B &= 0 \\ Q_A &= 0 \\ H_3 - H_4 &= c_2 Q_3 |Q_3| \\ Q_3 &= Q_4 \\ H_3 &= H_A \\ H_4 &= H_B \end{cases} \tag{8.1}$$

The curious thing about this system is that the both the Q_i's and H_i's cannot be determined from the system. All the Q_i's are coupled, and all the H_i's are coupled, but they cannot be determined. In fact, the system is singular, though not structurally singular. From

$$-Q_1 + Q_B + Q_4 = 0 \text{ and } Q_B = 0 \text{ and } Q_4 = Q_3$$

it follows that

$$Q_1 = Q_3$$

Furthermore, from

$$Q_2 + Q_A - Q_3 = 0\text{and } Q_A = 0$$

it follows that $Q_2 = Q_3$. Combining this with the previous observation yields $Q_1 = Q_2$. This is exactly the second equation in the system, hence the system has an infinite number of solutions. To see that the system is not structurally singular, observe that if the first equation is replaced with $H_2 = c_1 Q_1 |Q_1|$ and the second equation is replaced with $Q_1 + Q_2 = 0$, the derivation above yields $H_i = Q_i = 0$ everywhere. If the valve were to be closed, the ninth equation would be replaced by $Q_3 = 0$. In that case, the corresponding matrix is structurally singular. The system is perfectly determined if $Q_A = 0$ or $Q_B = 0$ is replaced by $H_i = c_3$, i.e., by prescribing H on one of the H-nodes. This is also the solution that makes the most sense physically.

In an isolated loop, all the H_i's (or p_i's) are coupled, but none are prescribed. This illustrates the necessity of checking for each group of connected H_i's whether at least one of them is prescribed. The original matrix correction algorithm fixes the issue by simply prescribing H on one of the H-nodes. The same thing will be done in the new version. Using Breadth-First Search (BFS) the connected H_i's will be grouped and a check will be done whether each group includes and equation of the form $H_j = c_j$. This approach can be carried out in $\mathcal{O}(|E|)$ time [20].[1] Another advantage is that the BFS principles can also be used in other parts of the matrix correction algorithm.

Algorithm

First the Hopcroft-Karp algorithm is run to check for any structural singularities. This algorithm returns the matching size and an array which for each variable v contains the equation matched to it (if any). In broad terms, the algorithm can be described as follows. The actual implementation involves a number of subtleties, but in order to not make it unnecessarily complicated a broad overview is given.

[1] An alternative would be to use cycle detection the bipartite graph representation of the matrix using Depth-First Search, which has the same complexity.

Algorithm 8.1 Matrix Correction

```
if MSize < $n$ then
    for each variable $v$ do
        if $v$ is unmatched then
            if $v = Q$ or $v = W$ then
                return Error
            else if $v = T$ then
                Do BFS starting from $v$ for H-node
                Prescribe $v$ at H-node found
            end if
        end if
    end for
end if
for each variable $v$ do
    if ($v = H$ or $v = p$) and $v$ is not grouped then
        Do BFS starting from $v$ grouping the connected variables
        if $v$ is not prescribed in current group then
            Do BFS starting from $v$ for H-node
            Prescribe $v$ at H-node found
        end if
    end if
end for
```

Note that W, p and T denote the mass flow, pressure and temperature quantities used in the heat module, respectively. The checks for H and p will always be carried out, since these can be caused by numerical singularities such as in case of isolated loops. The checks for the other quantities can only be caused by structural singularities.

8.3. Accuracy

Next to robustness, the solution accuracy or solution quality is another very important factor. In this section the solutions produced by the LAPACK solver are compared to the solutions yielded by the IMSL solver. This also presents an opportunity to optimise the number of iterative refinement iterations.

8.3.1. Procedure

The accuracy test consists of running a test set consisting of cases from the liquid and heat module. Both the steady and transient simulations are carried out. The liquid set consists of 37 Wanda cases from the liquid module. The heat set consists of 18 cases from the heat module. More details on the test cases and their correspondence to the case names shown in the bar graphs can be found in Tables A.1 and A.2.

First the test cases are run using the original IMSL solver. Then the LAPACK version with various numbers of iterative refinement iterations is run on the test set. The solutions of each LAPACK variant will be compared to the IMSL solutions.

Error Definition

For running the test set a Python script is used in combination with the PyWanda module. The way it works is that, after running the test cases, the script loops over all nodes in a given Wanda test case and compares the solutions per node and per quantity. The solution for a node is given as a time series. Let $\mathbf{u}^{(i,j)} = [u_1^{(i,j)}\ u_2^{(i,j)}\ \dots\ u_k^{(i,j)}]^\top$ denote the IMSL solution for quantity j in node i and $\mathbf{v}^{(i,j)} = [v_1^{(i,j)}\ v_2^{(i,j)}\ \dots\ v_k^{(i,j)}]^\top$ the corresponding LAPACK solution. The number of time steps is denotes by k. The ∞-norm provides a good starting point for a comparison, because the main interest is the overall solution accuracy.

The absolute error is given as

$$\varepsilon_{\text{abs}} = \max_{(i,j)\in N\times Q} \|\mathbf{u}^{(i,j)} - \mathbf{v}^{(i,j)}\|_\infty \tag{8.2}$$

and the relative error as

$$\varepsilon_{\text{rel}} = \max_{(i,j)\in N\times Q} \frac{\|\mathbf{u}^{(i,j)} - \mathbf{v}^{(i,j)}\|_\infty}{\|\mathbf{u}^{(i,j)}\|_\infty} \tag{8.3}$$

where N denotes the set of nodes, and Q denotes the set of quantities. The relative error is taken relative to the solution **u**, as large deviations from the IMSL solution method are undesirable. The Wanda users should not be able to notice much difference between the IMSL and LAPACK solutions. That does, however, not mean that the IMSL solution is necessarily closer to the exact solution than the LAPACK solution.

8.3.2. Iterative Refinement

The LAPACK solver will be tested with different numbers of iterative refinement iterations. The solution error with respect to the IMSL solution will be measured using zero, one, two and three iterations of iterative refinement.

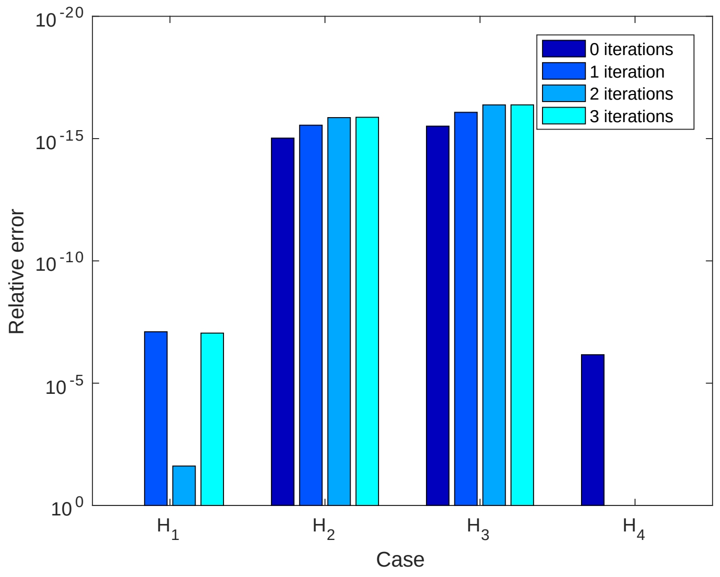


Figure 8.3: The error ε_{rel} for heat cases.

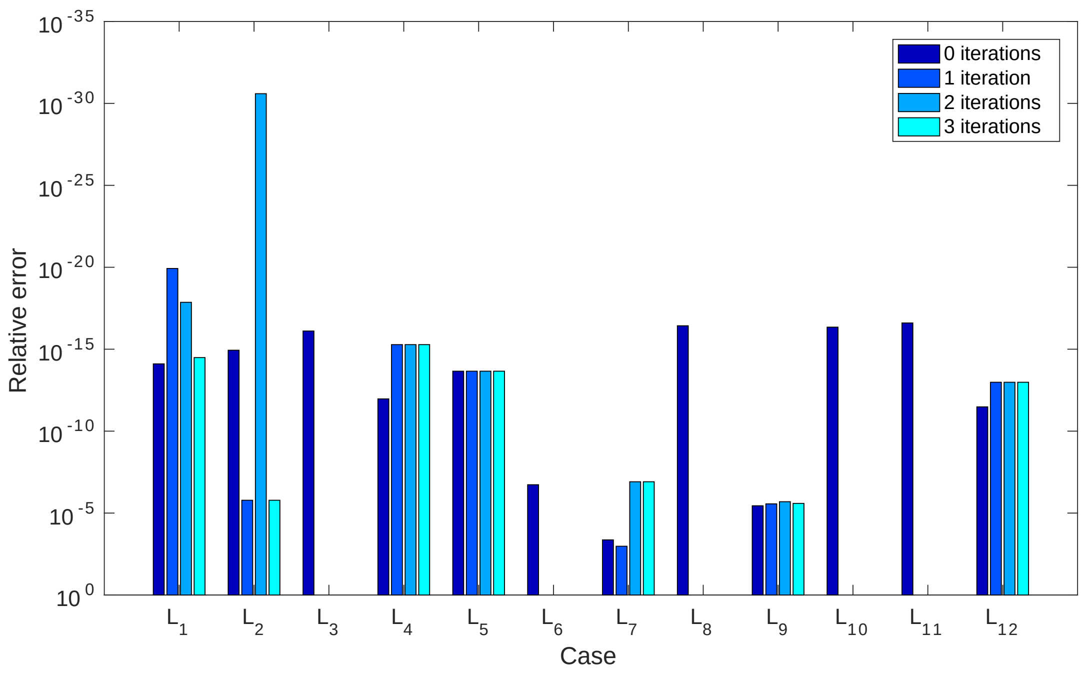
Fig. 8.3 shows the error ε_{rel} for the heat cases. Only the cases where the results are non-zero for at least one of the four different numbers of iterative refinement iterations are shown; 14 out of 18 cases have an exactly zero error for each variant. Note that the larger the bar, the lower the error, and hence the better the solution (compared to IMSL). Absent bars correspond to $\varepsilon_{\text{rel}} = 0$.

Case H_4 shows that iterative refinement can yield good improvements compared to no iterative refinement.[2] Case H_1 shows irregular behaviour. Without iterative refinement, the LAPACK routine detects a singular matrix; no solution is determined. The matrix is, however, not structurally singular. When using iterative refinement, no singularity is detected. It turns out that the matrices for this case are poorly conditioned. Throughout the transient simulation, the condition number estimation shows that $\kappa_\infty(M) \geq 10^{11}$ and at some point the estimation even shows that $\kappa_\infty(M) \geq 5 \cdot 10^{22}$. The bound in Section 5.3.4 shows that iterative refinement yields improvement if

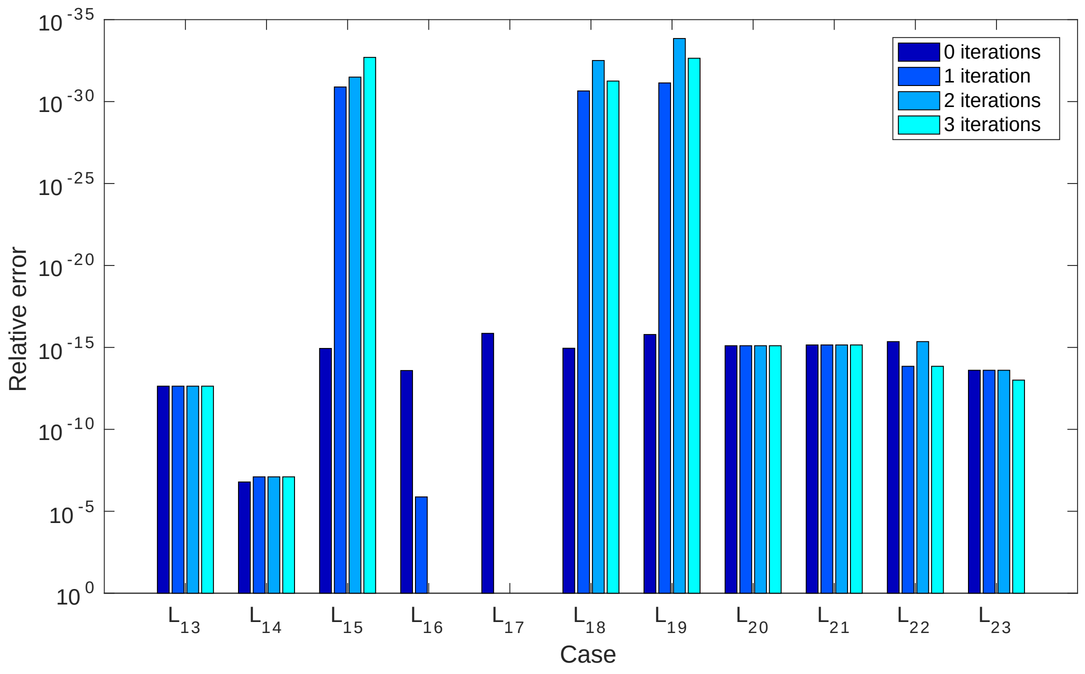
$$\|I - (\widehat{L}\widehat{U})^{-1}M\| < 1 \tag{8.4}$$

However, if M is badly conditioned, $\widehat{L}\widehat{U}$ tends to be a poor approximation of M, hence the condition above is unlikely to hold. This results in erratic behaviour, as illustrated in Fig. 8.3.

[2]The case names corresponding to the case symbols displayed here can be found in Table A.2.



(a) Part I



(b) Part II

Figure 8.4: The error ε_{rel} for liquid cases.

The results for the liquid cases are shown in Fig. 8.4. Only the cases for which the error is non-zero for at least one of the four different numbers of iterative refinement iterations are shown; 14 out of 37 cases have no error. Again, the larger the bar, the better the solution and absent bars correspond to $\varepsilon_{\text{rel}} = 0$.

The thing that stands out the most is that the cases L_1, L_2 and L_{16} show somewhat erratic behaviour.[3] Investigation shows that during the simulation the estimated $\kappa_\infty(M)$ do not exceed 10^6, 10^8 and 10^{10}, respectively.

[3]The case names corresponding to the case symbols displayed here can be found in Table A.1.

The L_1 is not considered ill-conditioned. The other two cases are not well-conditioned, but also not as ill-conditioned as case H_1. The erratic behaviour of these cases therefore cannot be explained by their conditioning. A possible explanation is that this is an artefact resulting from using the ∞-norm as opposed to, e.g., the Euclidean norm. Convergence may happen in the 2-norm, but not in the ∞-norm. Investigations shows that this is not the case; in 2-norm erratic behaviour is apparent as well. Another explanation may be that the solution is compared with the IMSL solution, which may not be close to the exact solution. I.e., convergence may occur with respect to the exact solution, but not with respect to the IMSL solution.

For most cases, doing more than one iteration of iterative refinement does not yield significant further improvements. There are exceptions, however, namely the cases L_2, L_7 and L_{16}. The good news is that for all three cases the error for one iteration is still within acceptable limits. L_7 has the largest error of about 0.1%, which is still an acceptable error in practice.

Number of iterations	0	1	2	3
Average error	$2 \cdot 10^{-5}$	$4 \cdot 10^{-5}$	$9 \cdot 10^{-4}$	$2 \cdot 10^{-7}$
Maximum error	$4 \cdot 10^{-4}$	10^{-3}	$2 \cdot 10^{-2}$	$3 \cdot 10^{-6}$
Error improvement	-	13	6	2

Table 8.1: Error statistics over all test cases per number of iterative refinement iterations.

Table 8.1 displays the number of cases where ε_{rel} using i iterations improves ε_{rel} using $i-1$ iterations by at least a factor 10. Furthermore, the average and maximum error for each number of iterative refinement iterations are shown. Both the heat and liquid cases are included in the results. The average and maximum error suggest that three iterations results in the most accurate solutions (at least compared to IMSL), however, the error improvement shows that there are only 2 cases for which three iterations yield significant improvements over two iterations. In practice, measurement errors in the input are likely to dominate numerical errors, so iterative refinement as a whole could be superfluous were it not that the H_1 case requires iterative refinement for a solution to be found. For this reason at least one iteration of iterative refinement will be used. A possibility is to adding an option for the user in Wanda to set the number of iterative refinement iterations with the default being one. Overall, the solution accuracy of LAPACK seems sufficient.

8.4. Run Time

The last factor of importance is the run time of the solution method. This will be evaluated in the current section.

8.4.1. Procedure

The benchmarks are performed on an *HP ProBook 6750B*. It contains an *Intel i5-3210M* dual core CPU running at 2.5 GHz. This laptop is representative for the average Wanda user.

The results are obtained using the `GETTIM` Fortran routine to measure the elapsed (clock) time. The resolution of this routine is 10ms. During the test runs only the results of the last time step will be written to an output file. Normally, the solution for every second of the simulation is written to an output file. For the purpose of this chapter, evaluating the performance of the new solution method, this does not matter. Only the relative running times of the different matrix solvers are relevant. For all tests, all three large test cases, as presented in Section 3.4.2, are run. More specifically, the run time of the transient flow scenarios are measured. As a starting solution, a steady state solution is computed using the matrix solver version that is being tested. The results shown are an average over five runs for the Filter and NH1 test cases, and three runs for the NH2 test case. Note that, unless explicitly mentioned, the current matrix correction algorithm is used for the tests.

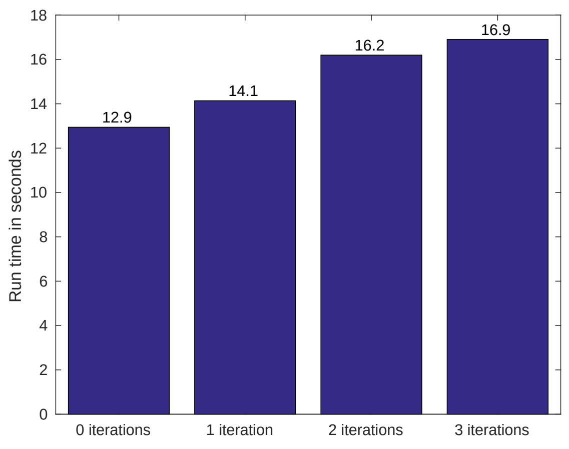
The following tests will be carried out.

- Comparison of the number of iterative refinement iterations.
- IMSL vs. LAPACK solution method

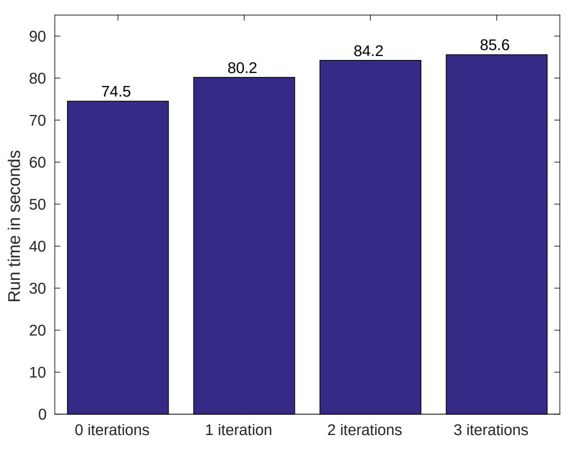
- LAPACK with internal singularity detection vs. condition number estimation vs. maximum size matching.
- Blocked vs. unblocked implementation of computing the LU-decomposition.
- Time consumption per step of the LAPACK solution method.
- Comparison of matrix correction algorithms.
- Reference LAPACK vs. vendor-optimised LAPACK

8.4.2. Iterative Refinement

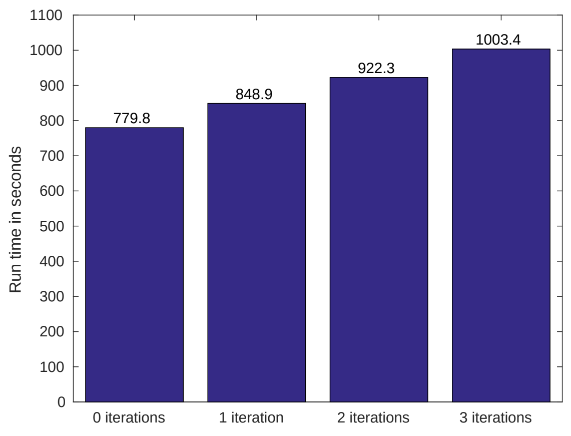
In this section the run times per number of iterative refinement iterations will be evaluated.



(a) Filter



(b) Noord-Holland 1



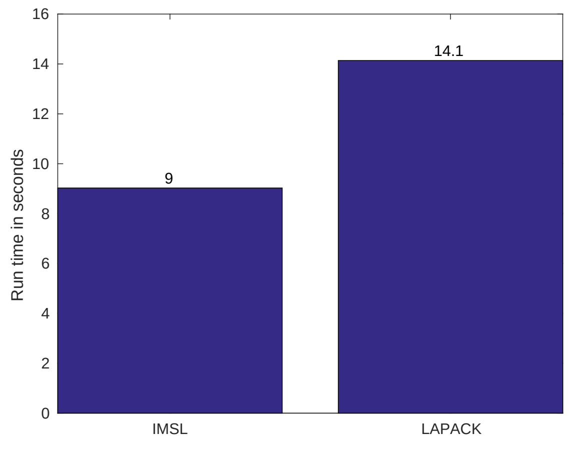
(c) Noord-Holland 2

Figure 8.5: Run times per number of iterative refinement iterations.

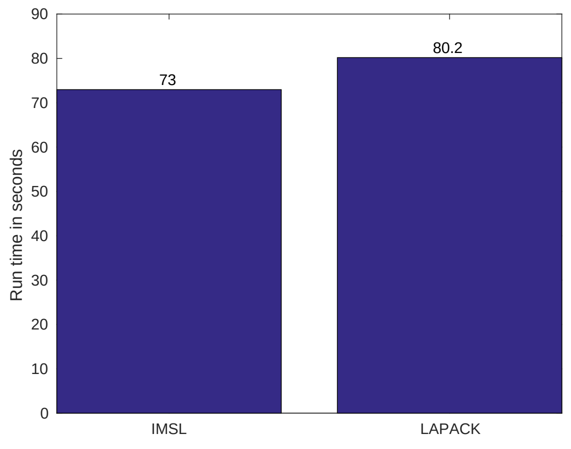
The results in Fig. 8.5 show that using one iteration of iterative refinement already adds a relatively big amount of time. The relative time additions are 9.3%, 7.7% and 8.9% for the three cases, respectively. As discussed in Section 8.3, iterative refinement is necessary since there are test cases that fail when not using it. The best practice is to add an option in Wanda for setting the number of iterations, with the default being one iteration. The absolute run times are still limited, and, as will be shown in the next section, are still comparable to the IMSL run times.

8.4.3. IMSL vs. LAPACK

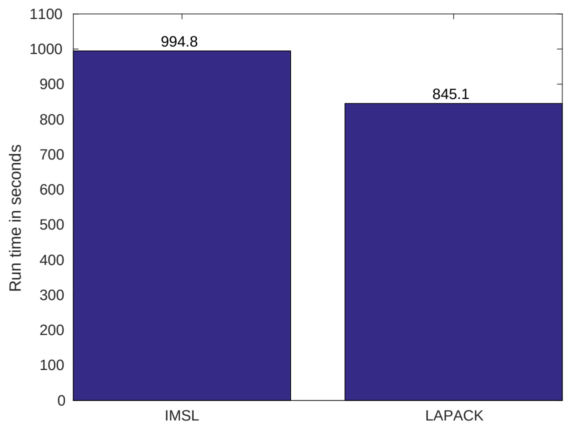
For the comparison between IMSL and LAPACK one iteration of iterative refinement is used for the LAPACK solution method.



(a) Filter



(b) Noord-Holland 1



(c) Noord-Holland 2

Figure 8.6: Run time comparison between IMSL and LAPACK.

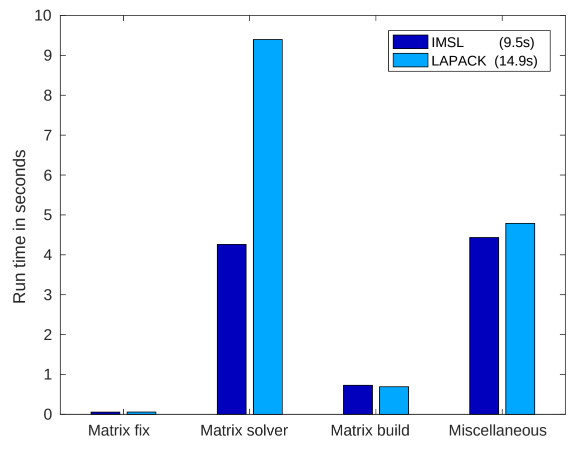
The time addition of the LAPACK run times relative to IMSL amount to 57%, 9.9% and -15%, respectively. The difference for Filter may seem very large, however, the absolute run times are still very low. For the largest case, LAPACK is even significantly faster.

Test Bench

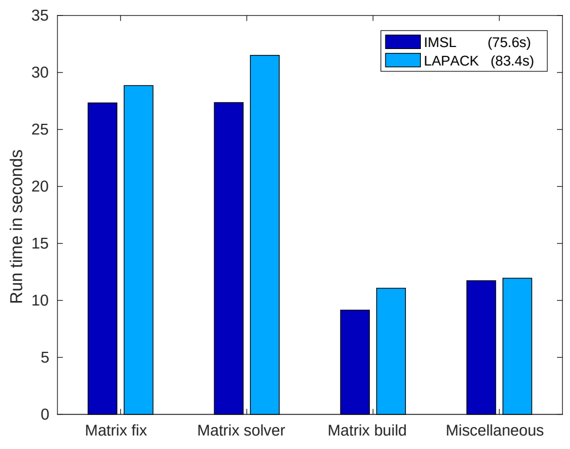
Running IMSL and LAPACK on the 55 test cases presented in Section 8.3.1 results in LAPACK being on average 3% faster than IMSL, while the worst result is about 3% slower. Most of these cases are relatively small test cases with matrices smaller than 1000×1000. These cases should resemble the average usage of Wanda more than the three large test cases above. The results indicate that the average user will not be affected by much change in computation times.

Profile

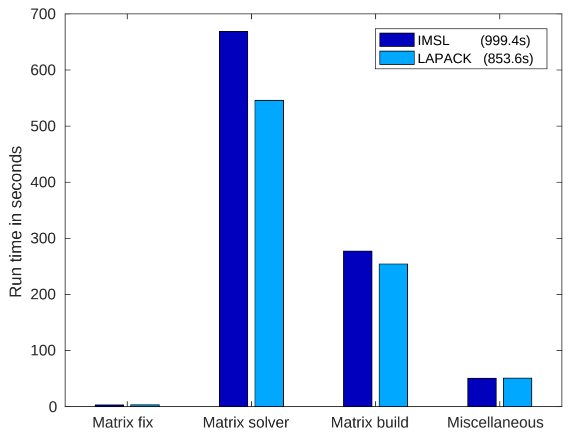
To get a better idea of how the run time of the LAPACK solution method itself compares to IMSL, the run time for the four categories as presented in Section 3.5 are measured and compared with IMSL. Note that, for this result, the solution of every time step is written to a file.



(a) Filter



(b) Noord-Holland 1



(c) Noord-Holland 2

Figure 8.7: Run time profile IMSL vs. LAPACK.

Fig. 8.7 shows the run time of the four categories for IMSL and LAPACK. For the Filter case the LAPACK solver takes significantly more time than IMSL for the actual solving of the system of linear equations. For NH1 LAPACK is also slightly slower. There is also a significant difference in the matrix build run time. This is because IMSL only requires 6276 iterations, while LAPACK requires 6742 iterations for the whole simulation. For NH2 the difference in run time is now in favour of LAPACK. Both methods require the same number of iterations: 92819. The time difference in matrix build may be explained by the allocation and deallocation of arrays used by IMSL. The cases for which LAPACK is slower the absolute time difference is still limited.

8.4.4. Singularity Detection

For this comparison, only the NH1 test case will be considered. The reason for this is that this case contains the most phase changes, hence a high number of singularity checks will be performed (after all, the singularity checks are done at the start of the simulation and after phase changes). Fig. 8.7 shows that the matrix fix algorithm run time is negligible for both other cases. Note that the original matrix correction algorithm is used in this test; only the singularity detection methods differ.

Fig. 8.8 shows the results. Singularity detection using maximum size matching only adds about 0.5 seconds to the run time. Condition number estimation is about 9 seconds slower than internal detection. Clearly, if additional singularity detection methods are required, maximum size matchings are the best choice performance-wise. The additional advantage is that the matching provides information about where the matrix is singular and where to fix it, as explained and implemented in Section 8.2.

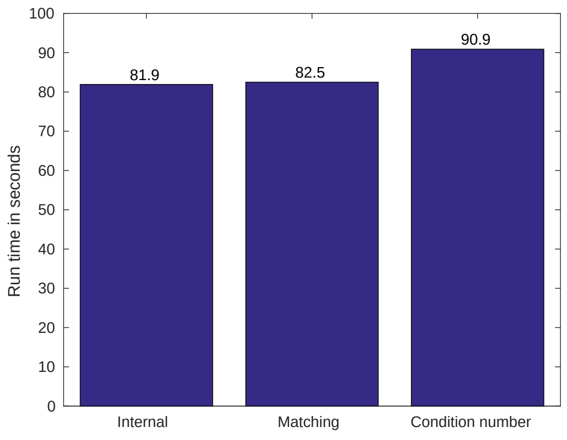


Figure 8.8: Total run time for each method of matrix structural singularity detection.

8.4.5. *LU*-Decomposition Computation

The LAPACK *LU*-decomposition subroutine `DGBTRF` checks whether the upper bandwidth of the input matrix is greater than 64. If so, the blocked version of the algorithm is used. Otherwise the unblocked routine `DGBTF2` is used. When using the block version, the majority of the flops come from matrix multiplications. These operations can usually be performed efficiently on actual computer hardware. As a result, a block version should be faster, especially for large matrices. The only test case with a bandwidth bigger than 64 (after permutation), is the NH2 test case. To see whether the blocked version yields a performance improvement a benchmark is performed.

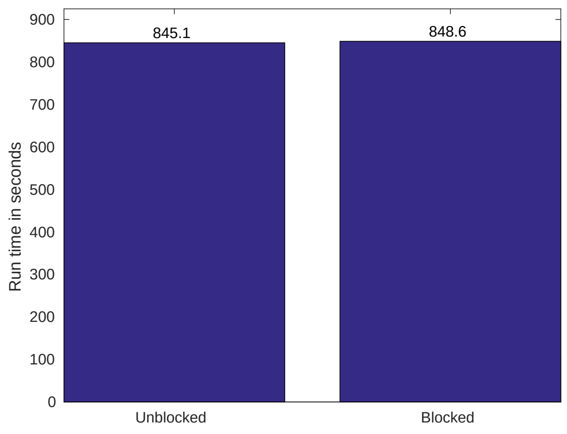


Figure 8.9: Run time for the blocked vs. unblocked *LU*-decomposition computation.

Fig. 8.9 shows no significant performance difference between the two methods, therefore the unblocked version will be used in the new solution method.

8.4.6. Solution Method Steps

In this section the run times for the first five steps presented in Section 8.1 will be measured. This will give a good overview of which steps are the most time-consuming within the LAPACK solution method. The times will be measured for both Noord-Holland cases. The Filter case will be neglected since the `GETTIM` routine will, due to its accuracy of 10ms, not yield accurate results. To summarise, the LAPACK solution method involves the following steps.

1. Obtaining the RCM permutations and their inverses.
2. Matrix coordinate format to band storage format conversion.
3. *LU* factorisation step.

4. *LU* solve step.

5. Iterative refinement (one iteration).

Misc. Miscellaneous operations not directly part of the other steps (allocating/deallocating memory, copying vectors and permuting the matrix and vectors).

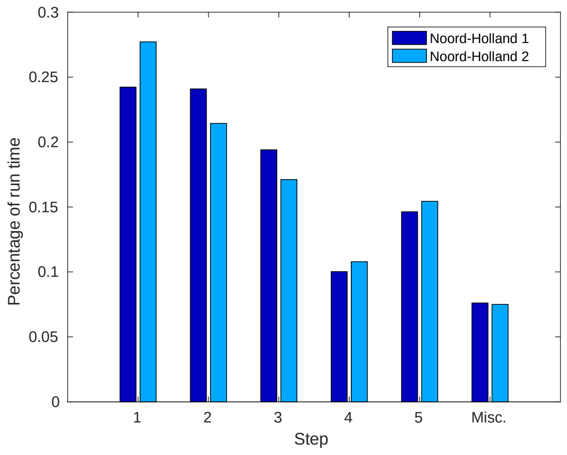


Figure 8.10: Percentage of run time per step of the LAPACK solution method.

Both the results for NH1 and NH2 are depicted in Fig. 8.10. Step 1, obtaining the RCM permutations, is the most time-consuming in both cases. This indicates that a more efficient RCM implementation would be beneficial. Step 2, converting the matrix to band format takes a significant amount of the time. This is mostly due to the fact that the band matrix first needs to be initialised to 0, which is responsible for the vast majority of the run time. Steps 3-5 and miscellaneous show no real surprises. The factorisation step consumes more time than the solve step, while iterative refinement sits between these two steps in terms of time consumption.

8.4.7. Matrix Correction Algorithms

A comparison is made between the original and new 'matrix fix' algorithms as presented in Section 8.2. Only the Noord-Holland 1 case is considered here, as in this case the algorithms are used often due to the high number of phase changes.

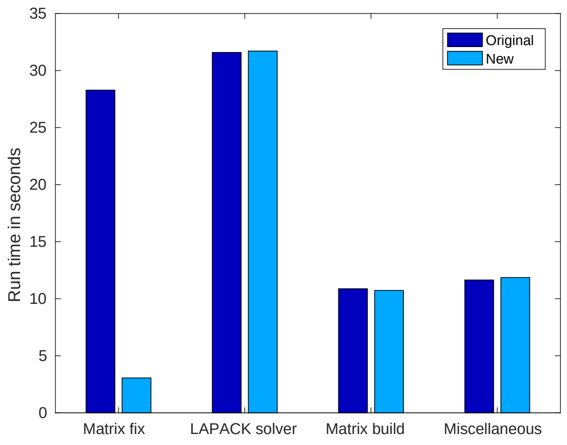
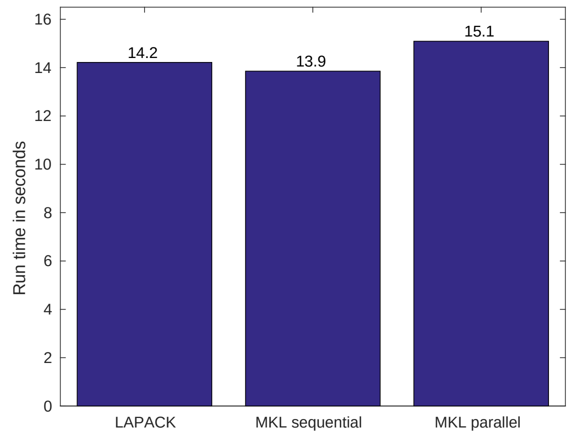


Figure 8.11: Matrix fix algorithms comparison.

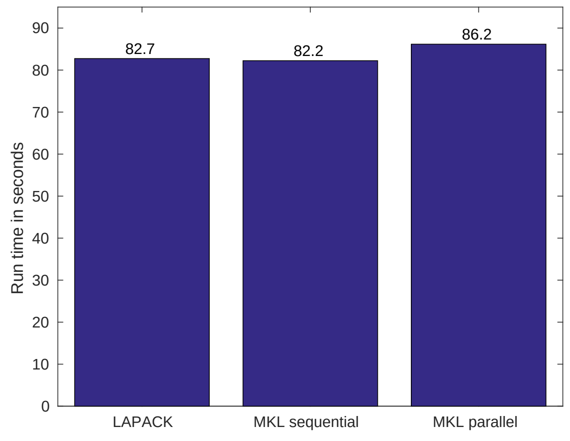
Fig. 8.11 shows a significant time reduction when using the new algorithm. In fact, the average total run time is reduced from 82.4 seconds to 57.3 seconds which is a time reduction of about 30%.

8.4.8. Reference LAPACK vs. Vendor-Optimised LAPACK

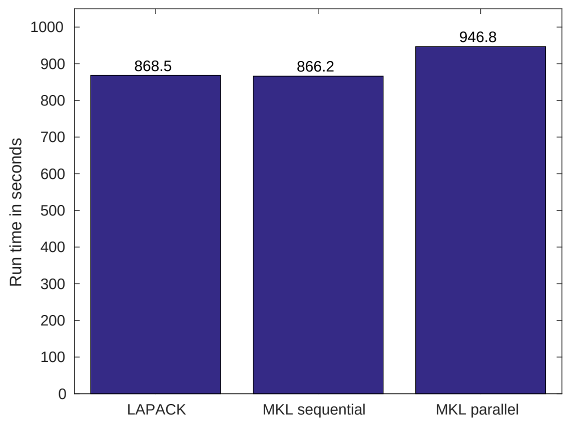
The LAPACK implementation currently used is the reference version of LAPACK including the required reference BLAS routines. It is advised to use an optimised BLAS/LAPACK implementation that is tuned for specific hardware as this usually results in better performance. Various optimised implementations are available such as PLASMA [2], MAGMA [1], OpenBLAS and MKL. To measure the performance benefit, the Intel Math Kernel Library (MKL) is compared to reference LAPACK. MKL is available in two versions: a sequential and parallel version. Both versions will be tested.



(a) Filter



(b) Noord-Holland 1



(c) Noord-Holland 2

Figure 8.12: Run time of reference LAPACK vs. MKL.

The results, as depicted in Fig. 8.12, show that the MKL sequential version is only marginally faster than reference LAPACK. Perhaps the typical problem size is too small to yield significant performance benefits. Additionally, since the reference LAPACK library is compiled with order 3 optimisation, compiler optimisations may already significantly boost the performance of reference LAPACK. In all three cases the parallel MKL is the slowest. Communication overhead likely dominates any performance benefits yielded by parallelising the computations, resulting in overall worse performance. The test shows that the performance of reference LAPACK is already sufficient in the context of Wanda. The additional benefit is that reference LAPACK is open source, while MKL is not. Optimised open source libraries do exist. Examples are PLASMA, MAGMA and OpenBLAS.

8.5. Evaluation

The results show that the LAPACK solver would be a good successor to IMSL. Taking both accuracy and run time into consideration, one iteration of iterative refinement seems to be the best trade-off between the aforementioned aspects. The run times using one iteration indicate that during extreme usage, but even more so during average usage, the user should not be able to notice large differences in computation time compared to IMSL.

To improve robustness, the current matrix correction algorithm is replaced by one that uses the maximum size matching property of a matrix. Using this method, any quantity being underdetermined can be detected and a correction can be applied. If no correction can be applied, a suitable error message is produced which indicates where in the pipeline system the problem lies. The correction algorithm is also significantly faster than the previous version.

In conclusion, the new solution method based around LAPACK offers a welcome improvement in terms of maintainability and robustness, while showing similar, and sometimes even improved, results in terms of accuracy and performance.

9 Conclusions and Recommendations

9.1. Conclusions

The main research question concerned how the maintainability and robustness of the solution method could be improved, without giving in on accuracy and efficiency. To this end, the LAPACK library together with a method for detecting singularities was implemented. The following conclusions can be drawn.

Maintainability

The open source nature and permissive license of LAPACK together with the fact that only the routines that are used need to be supplied make it an excellent solver in terms of maintainability.

Robustness

The method of detecting structural singularities can in practice detect the underdetermination of any variable, except in isolated loops. Underdetermined variables in isolated loops were detected the presence of an equation prescribing the variable. Just as in the old solution method, a correction can be applied in case H, p or T are undetermined, but the new method is much faster. Additionally, if Q or W are undetermined, an error message is shown indicating the location of the problem in the pipeline system. This was not possible in the old method, therefore resulting in crashes or infinite loops in the IMSL routines. This new addition ensures a user-friendly Wanda experience. Condition number estimation was also considered. This method proved to be reliable, but too slow in practice while also not being usable for correcting the matrix, therefore this method is neglected.

Accuracy

Since LAPACK has been a standard numerical library for many years, it should be trustworthy in terms of solution accuracy. Tests were done using different numbers of iterative refinement iterations on a test set of 55 cases. The results showed that, compared to IMSL, there was only one case for which the results were somewhat concerning, as the relative error was about 2.5%. These results can be explained by the cases being ill-conditioned, in fact, without iterative refinement LAPACK detects a singular matrix while running that case. Overall, the numerical errors are likely to be dominated by input errors and the accuracy compared to IMSL is good, ensuring that users will not notice large differences in solution quality between IMSL and LAPACK. Optionally, the number of iterative refinement iterations can be set to sacrifice run time for performance or vice versa.

Efficiency

To improve the performance of LAPACK, the Reverse Cuthill-McKee (RCM) heuristic was applied to reduce the bandwidth of the matrices before trying to solve the systems. Test results on the three large test cases showed that, when applying RCM, the LAPACK solution method run time is similar to that of IMSL. This result is confirmed by comparing the performance on the test set consisting of 55 cases. By default, one iteration of iterative refinement will be used. Using maximum size matchings for detecting singularities proved much faster than using condition number estimation. Basing a new matrix correction algorithm on this method not only improves the robustness of Wanda, but comparing this new method to the old correction algorithm shows a significant performance boost. A big advantage of using LAPACK is the wide availability of vendor-optimised versions, despite the fact that a test of the optimised MKL library showed that no significant performance improvements were gained. This is possibly due to the small problem sizes in LAPACK and the aggressive compiler optimisations applied to the reference implementation.

Unexplored Methods

Some of the research questions in Chapter 4 indicate directions which remain unexplored. A short explanation will be given as to why this is.

A lot of undetermined systems can be corrected using an a posteriori fix, but preventing these issues altogether by adjusting the physical model does not seem to be a good approach since underdetermination is always either a result of phase changes or a mistake by the user. Both issues can be resolved, although fixing W or Q being undetermined requires some knowledge of the underlying physics.

Rank-revealing decompositions seemed an inferior approach to using singularity detection using maximum matchings and simply using the LU-decomposition. Especially since the structural singularity approach can be used to identify the problem in the network.

9.2. Recommendations

One possible improvement is providing all the matrix coefficients for each possible phase of the components and only setting the coefficients of the current component phase to non-zero. The LAPACK solver is not affected by adding these zero elements. Now phase changes during a time step would leave the matrix structure intact. When the RCM permutation is computed based on this information, it only needs to be computed once every time step. This could improve performance, although it could also negatively affect the matrix bandwidth. This method also allows for keeping the Jacobian, apart from changes in flow direction, constant within a time step. Adding these two improvements could yield a significant performance boost.

Optimised BLAS and/or LAPACK libraries could yield improvements. Using implementations such as ATLAS and OpenBLAS could possibly be beneficial. This would also not require large changes in the code. Similarly, the performance of entirely different numerical libraries such as MUMPS could be investigated.

Since matrix bandwidth is what really determines the performance of the matrix solver, a larger bandwidth reduction could potentially lead to a large performance improvement. The objective is to find a bandwidth minimisation heuristic that performs better than RCM and has a similar run time. For both Noord-Holland test cases computing the RCM permutations takes most of the solution method run time, hence a more efficient implementation of RCM, if it exists, could already yield significant improvements.

Alternatively, ordering the components in a pipeline system a priori could make a posteriori methods obsolete, so this also seems to be a potentially fruitful research direction.

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] L.W. Beineke and R.J. Wilson. *Selected topics in graph theory*. Number v. 3 in Selected Topics in Graph Theory. Academic Press, 1988. ISBN 9780120862030.

[11] 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.

[12] John Burkardt. Grafpack – graph computations, 2000. URL https://people.sc.fsu.edu/~jburkardt/f_src/grafpack/grafpack.html.

[13] John Burkardt. Reverse cuthill-mckee ordering, 2003. URL https://people.sc.fsu.edu/~jburkardt/f_src/rcm/rcm.html.

[14] 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.

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

[16] 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.

[17] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011. ISBN 9781118031391.

[18] 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.

[19] 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.

[20] Shimon Even. *Graph Algorithms*. Cambridge University Press, 2 edition, 2011. doi: 10.1017/CBO9781139015165.

[21] Jeffrey Finkelstein. Source code for bipartite matchings, 2015. URL https://networkx.github.io/documentation/latest/_modules/networkx/algorithms/bipartite/matching.html.

[22] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981. ISBN 0131652745.

[23] Norman E. Gibbs, William G. Poole, and Paul K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976. ISSN 00361429. URL http://www.jstor.org/stable/2156090.

[24] Manfred Gilli and Myriam Garbely. Matchings, covers, and jacobian matrices. *Journal of Economic Dynamics and Control*, 20(9):1541 – 1556, 1996. ISSN 0165-1889. doi: https://doi.org/10.1016/0165-1889(95)00910-8. URL http://www.sciencedirect.com/science/article/pii/0165188995009108.

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

[26] Ch H. Papadimitriou. The np-completeness of the bandwidth minimization problem. 16:263–270, 01 1976.

[27] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[28] 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.

[29] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi: 10.1137/0202019. URL https://doi.org/10.1137/0202019.

[30] 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.

[31] Hang Tong Lau. Laupack – graph routines, 2000. URL https://people.sc.fsu.edu/~jburkardt/f_src/laupack/laupack.html.

[32] H.T. Lau. *Algorithms on graphs*. A Petrocelli book. TAB Professional and Reference Books, 1989. ISBN 9780830634293.

[33] A. Lim, B. Rodrigues, and Fei Xiao. A centroid-based approach to solve the bandwidth minimization problem. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 6 pp.–, Jan 2004. doi: 10.1109/HICSS.2004.1265221.

[34] Andrew Lim, Brian Rodrigues, and Fei Xiao. Heuristics for matrix bandwidth reduction. 174:69–91, 10 2006.

[35] 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.

[36] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. *User Guide for MINPACK-1*, August 1980.

[37] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-47465-5, 9780521474658.

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

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

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

[41] A. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms for Computers and Calculators*. Academic Press, New York, 1978.

[42] J. K. Reid and J. A. Scott. Reducing the total bandwidth of a sparse unsymmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 28(3):805–821, 2006. doi: 10.1137/050629938. URL `https://doi.org/10.1137/050629938`.

[43] Youcef Saad. *SPARSKIT: a basic tool kit for sparse matrix computations - Version 2*, 1994.

[44] Yousef Saad. Sparsekit2 – sparse matrix utility package, 1989. URL `http://people.sc.fsu.edu/~jburkardt/f77_src/sparsekit2/sparsekit2.html`.

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

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

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

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

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

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

A
Appendix

A.1. Conservation Laws

A.1.1. Conservation of Momentum

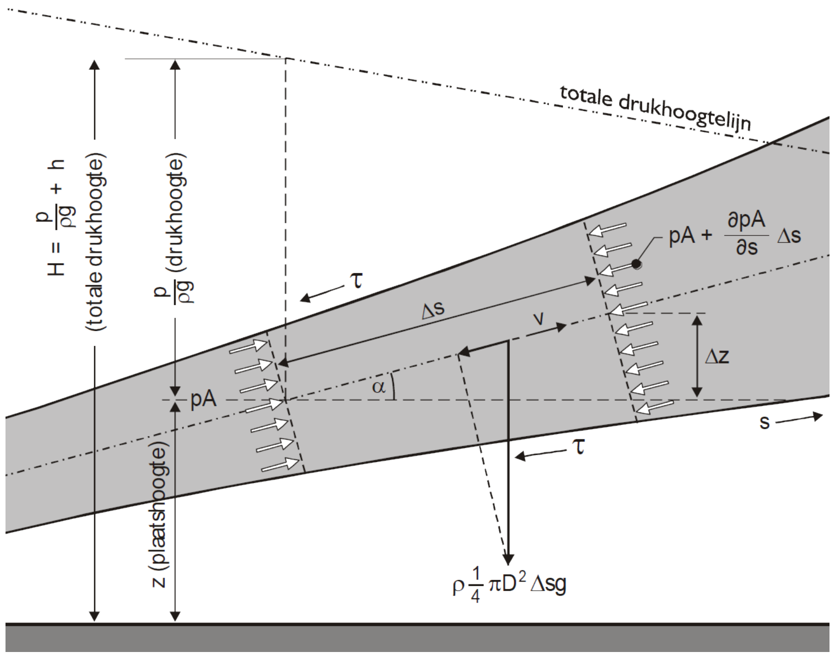

totale drukhoogtelijn
$H = \frac{p}{\rho g} + h$ (totale drukhoogte)
$\frac{p}{\rho g}$ (drukhoogte)
z (plaatshoogte)
τ
Δs
v
$pA + \frac{\partial pA}{\partial s}\Delta s$
pA
α
Δz
s
τ
$\rho \frac{1}{4}\pi D^2 \Delta s g$

Figure A.1: Conservation of momentum.

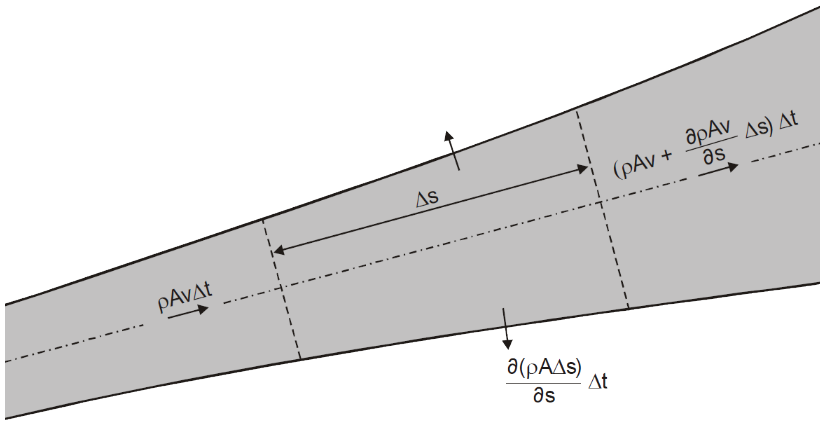
Note that in the figure Δs is used instead of Δx. From Fig. A.1 the law of conservation of momentum can be derived, for Δx small.

$$\underbrace{pA + \left(pA + \frac{\partial(\rho A)}{\partial x}\Delta x\right)}_{\text{Pressure on ends}} \underbrace{- \rho g \Delta x \left(\frac{A + A + \frac{\partial A}{\partial x}\Delta x}{2}\right)\sin\alpha}_{\text{Force due to weight}} - \underbrace{\tau O \Delta x}_{\text{Friction}} + \underbrace{\left(p + \frac{1}{2}\frac{\partial p}{\partial x}\Delta x\right)\frac{\partial A}{\partial x}\Delta x}_{\text{Pressure on sides}}$$
$$= \underbrace{\rho \Delta x \left(\frac{A + A + \frac{\partial A}{\partial x}\Delta x}{2}\right)\frac{\mathrm{d}v}{\mathrm{d}t}}_{\text{Element mass acceleration}} \tag{A.1}$$

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.



$\rho Av\Delta t$
Δs
$(\rho Av + \frac{\partial \rho Av}{\partial s}\Delta s)\,\Delta t$
$\frac{\partial(\rho A\Delta s)}{\partial s}\,\Delta t$

Figure A.2: Conservation of mass.

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

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

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

A.2. Large Test Cases

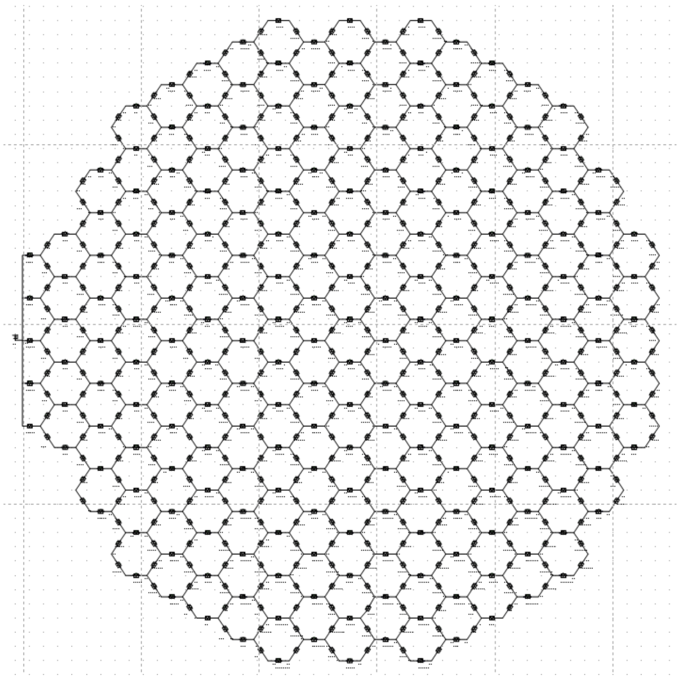


Figure A.3: Filter test case.

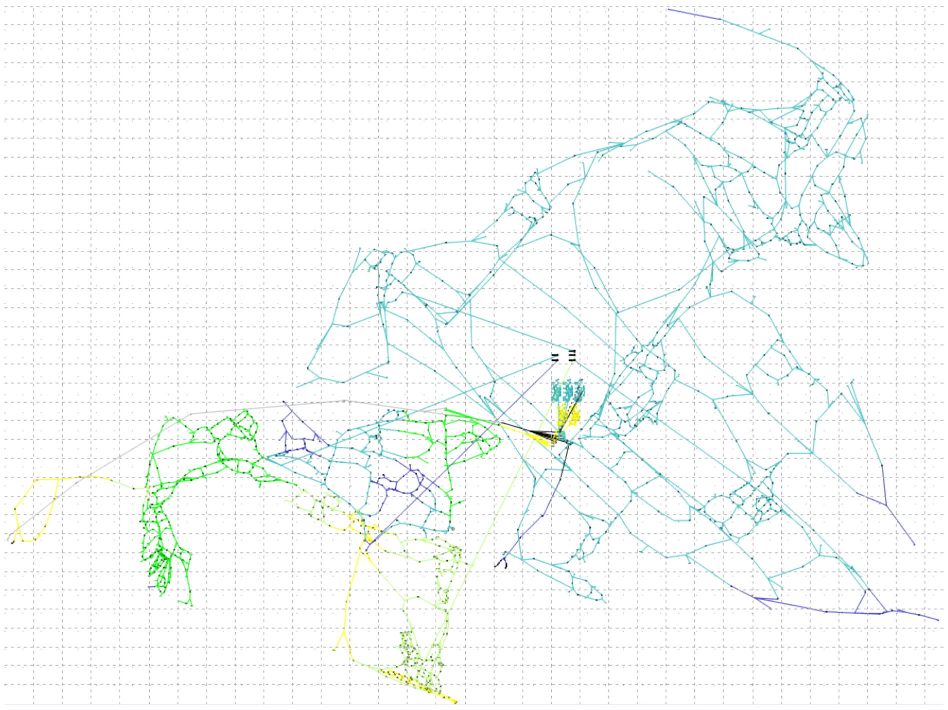


Figure A.4: Noord-Holland 1 test case.

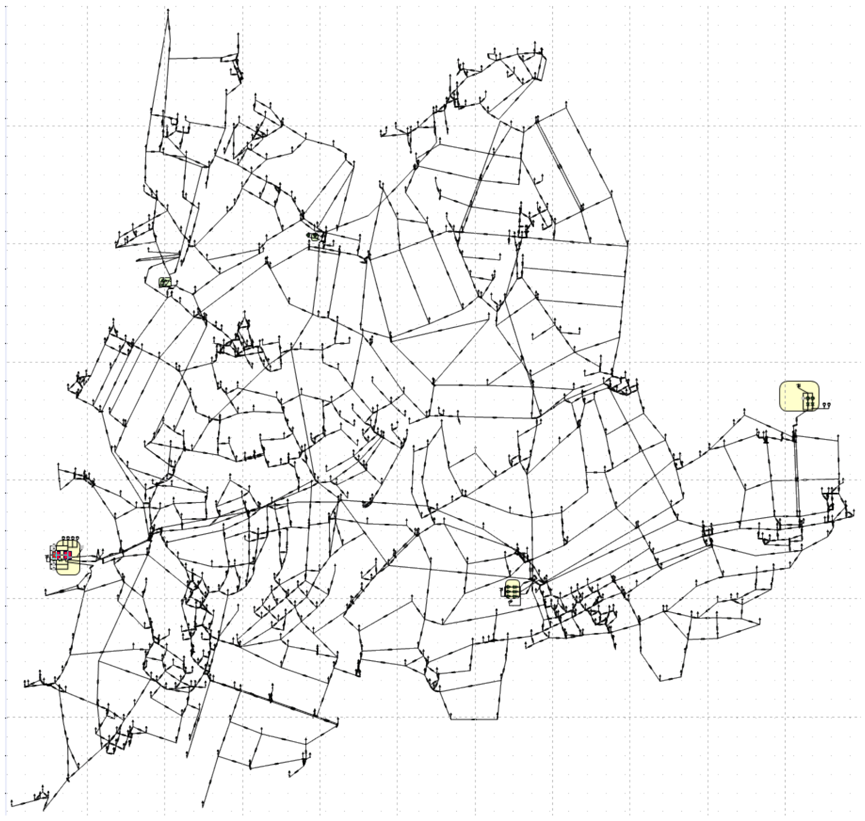


Figure A.5: Noord-Holland 2 test case.

A.3. Hopcroft-Karp Pseudocode

Algorithm A.1 Hopcroft-Karp

```
   Input: Graph $G$
   Output: Maximum matching PairU, MSize
 for $u$ in $U$ do
     PairU($u$) ← 0
 end for
 for $v$ in $V$ do
     PairV($v$) ← 0
 end for
 MSize ← 0
 while BFS() = true do
     for $u$ in $U$ do
         if PairU($u$) = 0 then
             if DFS(u) = true then
                 MSize ← MSize + 1
             end if
         end if
     end for
 end while
 return MSize, PairU
```

Algorithm A.2 Breadth-First Search (BFS)

```
   Input: -
   Output: True if augmenting path exists, false otherwise
 for $u$ in $U$ do
     if PairU($u$) = 0 then
         Dist($u$)←0
         Enqueue($u$)
     else
         Dist($u$)←$\infty$
     end if
 end for
 Dist(0) ← $\infty$                              ▷ Dist is initialised as Dist(0:n)
 while Empty(Q) = false do                         ▷ 0 denotes the null vertex
     $u$ ← Dequeue(Q)
     if Dist($u$)< Dist(0) then
         for $v$ adjacent to $u$ do
             if Dist(PairV($v$)) = $\infty$ then
                 Dist(PairV($v$)) = Dist($u$)+1
             end if
         end for
     end if
 end while
 return Dist(0) != $\infty$
```

Algorithm A.3 Depth-First Search (DFS)

```
   Input: A vertex $u$ in $U$
   Output: True if successfully augmented (or $u$=0), false otherwise
 if $u$ != 0 then
     for $v$ adjacent to $u$ do
         if Dist(PairV($v$)) = Dist($u$)+1 then
             if DFS(PairV($v$)) = true then
                 PairV($v$) $\leftarrow u$
                 PairU($u$) $\leftarrow v$
                 return true
             end if
         end if
     end for
     Dist($u$)$\leftarrow\infty$
     return false
 end if
 return true
```

A.4. Solution Accuracy Test Cases

Case name	Symbol
airvalve	L_1
Airvessels	L_2
boundaries	-
channel	-
Checkvalves	L_3
collector	-
InfPipesQH	-
pump_drivetypes_control	L_4
Resistances	-
SurgeTowers_combined	L_5
TJunct_brancL_allphases	L_6
Tstraight_FINAL	L_7
Tap_combined	-
Fcv	-
PdCV	L_8
PRV_combined	L_9
PuCV	L_{10}
valve	L_{11}
vent	L_{12}
venturi	-
weir_combined	L_{13}
YJUNC_COMB_ALLPHASES	L_{14}
YJUNC_DIV_ALLPHASES	-
cooling_water	L_{15}
LoadingArm_with_Control	L_{16}
Loading_System	L_{17}
Pump_trip_control	-
Sewage_transient	-
trip11_QSd	L_{18}
trip17_QSe	L_{19}
CAPWAT_241	L_{20}
CAPWAT_242B	L_{21}
CAPWAT_257	L_{22}
CAPWAT_258b	L_{23}
M129_multiple_200el_qs_oi1	-
Perugia99052608_QS	-
Perugia99052609_QS	-

Table A.1: Liquid cases.

Case name	Symbol
4wayheat	H_1
Airvessel_pipe	H_2
checkvalve_1	-
GASBOIL	-
HEATDEMDIS	-
HEATSUPdownT	-
heatsupqh	-
heatsupqh_wlim	-
heatexchanger_simple	-
pipe	-
pumpheat	-
heatresist	-
solarcollector	-
surge_tower_pipe	H_3
valve	H_4
District_heating_system_EPS	-
District_heating_system_RTS	-
slug	-

Table A.2: Heat cases.