DELFT UNIVERSITY OF TECHNOLOGY

AUTOMATING AC POWER FLOW SIMULATIONS
THE EUROPEAN ELECTRICITY GRID

Thesis MSc Applied Mathematics

AUTHOR:
Shravan Chipli

SUPERVISORS TU DELFT:
Prof.dr.ir. C. Vuik
M.E. Kootte

SUPERVISORS TENNET:
Martin Wevers
Jorrit Bos

AUGUST 2021

Foreword

This MSc graduation project is a collaboration between the group of Numerical Mathematics, TU Delft, and TenneT TSO B.V. TenneT is a European electricity transmission system operator (TSO) that manages the high-voltage grid in the Netherlands and large parts of Germany.

The power flow problem involves determining the voltages in every bus of the power system. The voltages are then used to compute the power flow in every branch of the power system network. This constitutes steady state power flow simulations which, for given generation and consumption data, give insight into the steady state behavior of the network. Hence, power flow simulations play a fundamental role in various sectors of a TSO such as operation and planning.

In this project, we develop a mathematical framework that automates AC power flow simulations by solving both converging and diverging power flow problems. On the basis of the mathematical framework, we develop an algorithm that can be used in TenneT to perform year-round AC power flow simulations with little manual intervention. We demonstrate the applicability of the algorithm by implementing it on artificial test networks, and two real-world transmission networks - the Dutch transmission network and a sub-European transmission network.

Contents

Chapter 1

Introduction

TenneT is responsible for safe and reliable transport of electric power in the Netherlands and large parts of Germany (fig. 1.1 shows the grid map of TenneT). One of the key challenges to ensure safety and reliability is to keep the voltages within safe limits across the transmission network. The rapid increase in addition of Renewable Energy Sources (RES) into electricity networks causes voltage fluctuations to occur more frequently. To manage such situations best in the future, clever planning of the infrastructure that is needed to keep the voltages within safe limits is necessary. Hence, the objective of transmission system planning is to plan the essential infrastructure to best prepare the transmission system for the future to ensure safety and reliability.

Year-round AC power flow simulations are required in TenneT to plan the future grid. An essential part of planning is to make modifications or additions to the existing grid and test it with varying hourly load and generation data. For example, modifications to the topology (structure) of the grid are made by adding a few lines to the network and such changes are studied with varying load and generation data. A consequence of this is that the power flow problem often fails to converge to a solution. In TenneT, convergence problems are currently being solved by manually making adjustments to the power flow problem. For example by making changes to the topology of the grid or by making adjustments to the load and generation data. This is a trial and error technique which does not guarantee an optimal solution. Moreover, it is impossible to manually solve convergence problems during year-round simulations which involve solving the power flow problem with hourly load and generation data of the next few decades. For this reason, AC power flow simulations are currently being done at TenneT for only a few critical hours. Hence, in order to perform AC year-round simulations, a computer program that can automatically solve convergence problems without manual intervention is necessary. This problem statement forms the basis for this project and we thus develop an algorithm which can automatically fix convergence problems during year-round AC power flow simulations.

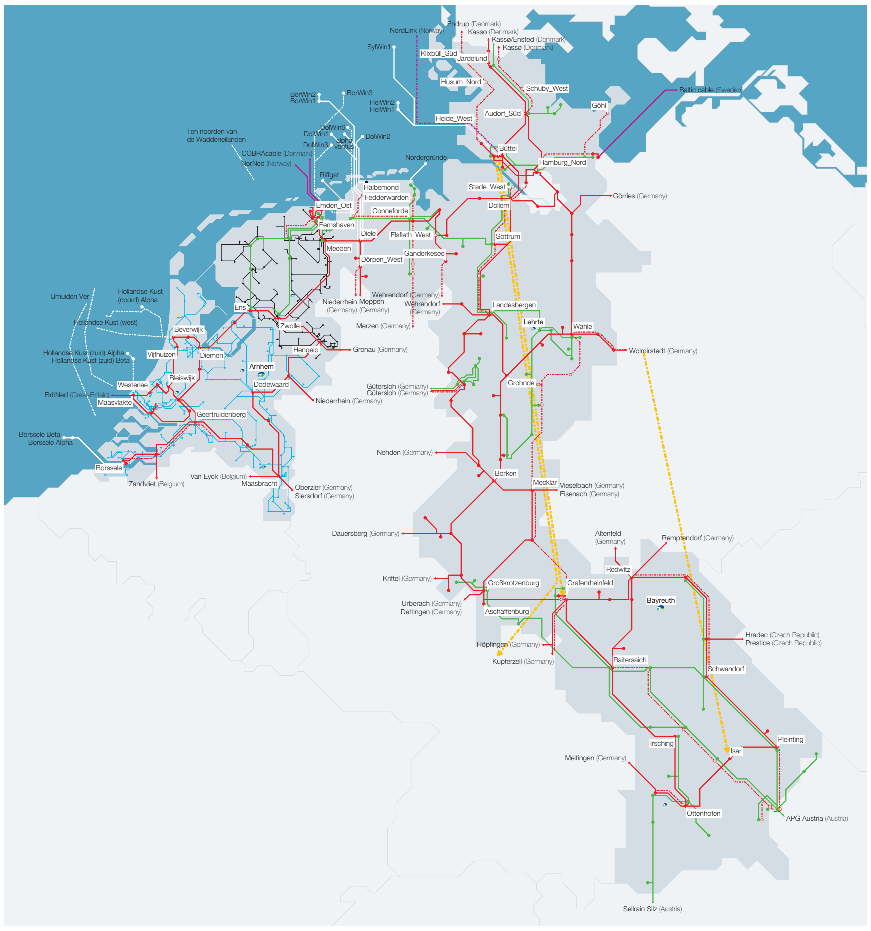


Figure 1.1: Grid map of TenneT

The project is structured as follows. The mathematical model of the transmission system is derived and studied in order to understand the power flow problem which lies at the heart of power flow analysis. To derive the mathematical model, we study the fundamentals of AC circuits and the transmission network topology. Solving the power flow problem requires a good understanding of power flow solvers. We investigate some of the prominent power flow solvers used in power flow analysis and understand their capabilities and limitations. The key to automate simulations is understanding power flow convergence. We present a detailed mathematical background that is necessary to understand the problems that are commonly encountered with power flow convergence. Based on this premise, we design a mathematical framework to solve both converging and diverging power flow problems during year-round simulations.

To practically realize the designed mathematical framework, we develop an algorithm in Python which can be used in TenneT for automating year-round AC power flow simulations. We call this algorithm the *automating algorithm*. As TenneT is currently using a commercial software for power flow analysis which does not allow access to its solvers, a crucial part of the automating algorithm is an interface that converts electricity grid models from the commercial software to an open source Python package. The capabilities and limitations of the automating algorithm are demonstrated by applying it to some test networks that are commonly used in the literature on power flow analysis and two networks that are used in TenneT to perform year-round simulations - the transmission network of the Netherlands and a sub-European transmission network that consists of the Netherlands, Germany, Belgium, France and Luxembourg. Finally, we present our conclusions and recommendations for taking this project ahead in the future.

Chapter 2

The Electric Grid

Electric grids are some of the largest networks humanity has ever built. For more than a hundred years[1] now, they have been doing an incredible job lighting up our societies. It's very hard to imagine life without electricity, for the world then would come to a halt.

The fact that electric power is just a switch away in most parts of the world today is a result of the intricate electric grids that quite seamlessly bridge generation and consumption. The complexity of electric grids is increasing at a rapid pace with integration of renewable energy sources. A strong mathematical approach and collective effort are imperative to understand, model, and control the electric grids of this day better.

This chapter briefly describes the fundamental stages of an electric grid and a few challenges down the road.

2.1 Generation

Conventionally, electric power is generated as Alternating Current (AC) at places known as power plants or generating stations. An *AC generator* transforms mechanical energy to electric energy by electromagnetic induction. The *prime movers* that drive generators could be steam/gas/water/wind turbines, internal combustion engines or nuclear reactors depending on location and availability of resources. For example, electricity in the Netherlands is produced primarily from natural gas and coal [2].

An exception to AC generation is the use of photovoltaic (PV) cells. PV cells transform solar energy to Direct Current (DC) which is then converted to AC by an *inverter* before injecting it into the grid.

[1]World's first central electric generating station - The Pearl Street Station, began operation on 4 September 1882 [1].

Electric power is generated at a frequency of 50 or 60 Hz [3]. This has been widely accepted as the optimal frequency considering several applications on a wide scale over a long period of time. In Europe, the operating frequency is 50 Hz.

2.2 Transmission and Distribution

The distinct advantage of alternating current is that it can be efficiently stepped up and down in voltage using a transformer [4]. High voltage power is preferred for transmission over large distances as resistive losses are less. The high voltage network called the *transmission system* is responsible for transporting electricity in the order of 110-380 kV from power plants to substations [5]. The rest of the grid constituting of medium and low voltage lines is called the *distribution system* and is responsible for distributing electric power to end users. The substations shift the voltage down to the order of 10-20 kV and finally distribution transformers that are locally installed step it further down to the order of utilization voltage ($<$1kV) making it suitable for domestic and commercial use. It should also be noted that high voltage DC (HVDC) is sometimes preferred to AC when transmission distances are long enough to justify a reduction in cost of conductors (three phase AC needs three cables whereas DC needs two) over AC/DC conversion costs.

Electric power is generated and transmitted in three phases for two significant reasons. One, it facilitates smooth conversion of energy by applying uniform torque on generators and motors, unlike single phase AC which results in pulsating torque. This is an engineering advantage as the rotors are well balanced. Two, it offers cost benefits as the same amount of power can be transmitted with fewer conductors when compared to single phase transmission. Currents and voltages add up to zero in a balanced three phase system, eliminating the need for a common return wire[2]. This is under the assumption that all loads have equal impedance[3]. However, in practice the impedances are slightly different, hence requiring a return wire to complete the circuit. This is achieved by connecting the combined return wires to the common ground[4] at both ends.

Transmission systems are modelled and analyzed as single phase AC systems for the very reason of being balanced. That is, currents and voltages are equal in magnitude in all three phases. Whereas in distribution systems, current and voltage magnitudes are different because of the difference in loads and hence distribution systems are modelled

[2]The three return wires are combined as a single return wire.

[3]Impedance is the AC equivalent of resistance.

[4]In electric circuits, ground refers to an electrically neutral reference that has 0 voltage. In this context, it refers to the earth which acts as the return pathway of the circuit.

considering all three phases. In fact, it is interesting to note that residential and most commercial circuits run on single phase AC and have two wires, one live and the other neutral (return).

2.3 Consumption

Loads are devices that consume electric power and are characterized by impedance. Theoretically, they can be broadly classified as *resistive, inductive* and *capacitive* loads. Resistive loads are heating conductors that are seen in incandescent bulbs and heaters. Inductive loads are all kinds of motors, fluorescent lamps and the transformers used in power supplies. Capacitors generally do not do physical work like other loads but are part of electrical circuits [4]. Based on usage, loads can be classified as residential, commercial, industrial and electric railways. Another important category of loads is consumer electronics [4].

From the system and also modelling perspective, electric power consumption is considered as *aggregate* load that combines several consumers. This may include households, city blocks or entire cities and regions. Given that the electric power industry is largely customer driven, capacity planning and serving instantaneous demand are very crucial for grid operators. *Load forecasting* is a discipline in itself and plays an important role in uninterrupted supply of electricity.

2.4 Challenges

The stochastic nature of renewable energy sources such as wind and sun poses unprecedented challenges to the electric grid. Solar and wind farms are highly uncertain in generating power and cause severe problems to grid stability, possibly resulting in overloading and blackouts. Decentralized power generation by small windmills and rooftop solar panels that are connected to the distribution system causes a change in power-flow direction. This results in two-way traffic, making grid control and even power-flow analysis a hard task. Electric vehicle charging is another difficult-to-predict scenario on the distribution side of the electric grid.

The largest power outage in history occurred in north-east India on 30, 31 July, 2012 and affected 620 million people. In 2016, a blackout occurred in South Australia due to storms that caused a cascading failure of the transmission system infrastructure, affecting 1.7 million people. In May 2020, TenneT declared emergency state due to a high voltage incident that occurred due to high RES infeed and low demands.

It is very likely that such events take place in the future owing to the rapid changes the electricity systems are experiencing lately. Hence, power flow simulations play a pivotal role in understanding the steady-state behavior of the constantly evolving electricity grids.

Chapter 3

Modelling the Transmission System

Power flow analysis is the numerical analysis of the flow of electric power and involves determining the operating state of the entire power system. The *state* describes how the power system functions and helps to understand how the system responds to inputs. In this chapter we derive the mathematical model of the transmission system and define the power flow problem which is cardinal to power flow analysis.

3.1 Fundamentals of AC circuits

This section describes the characteristics of an AC circuit, as required to model the transmission system. Definitions and equations are based on *electric circuit theory*.

3.1.1 Current and Voltage

In an AC system, current and voltage are sinusoidal functions characterized by amplitude, frequency and phase. They are expressed as,

$$i(t) = I_{max} \sin(\omega t + \phi_I) \qquad v(t) = V_{max} \sin(\omega t + \phi_V) \tag{3.1}$$

where,

$$\begin{aligned} I_{max} &= \text{amplitude of current},\ A \\ V_{max} &= \text{amplitude of voltage},\ V \\ \omega &= \text{angular frequency},\ rad/s \\ t &= \text{time},\ s \\ \phi &= \text{phase shift},\ rad \end{aligned}$$

For load-flow calculations, average values of currents and voltages are preferred. Averaging is done by considering Root Mean Sqaure (RMS) values of current and voltage functions. Since sinusoidal functions are perfectly symmetric, the effective or rms value is $1/\sqrt{2}$ times the amplitude. Instantaneous current and voltage are now written as,

$$i(t) = \sqrt{2}|I| \sin(\omega t + \phi_I) \qquad v(t) = \sqrt{2}|V| \sin(\omega t + \phi_V) \tag{3.2}$$

where |I| and |V| are rms values which are calculated as follows.

$$|I| = \sqrt{\frac{1}{T}\int_0^T i^2(t)\,dt} \qquad |V| = \sqrt{\frac{1}{T}\int_0^T v^2(t)\,dt} \tag{3.3}$$

where $T = 2\pi/\omega$ [s] is the period of sine wave. Intuitively, rms value is equal to the DC equivalent that dissipates the same amount of electric power in a given resistor per unit time.

In balanced three phase systems, current and voltage are equal in magnitude in all three phases but shifted in phase by $2\pi/3$ rad. Consequently, the power flow problem is solved by considering only one phase and the other two phases are analyzed simply by incorporating the phase shift. Note that it is convention to use the cosine function to describe current and voltage in power flow analysis. For a balanced three phase system, the governing equations are,

$$i(t) = \sqrt{2}|I| \cos(\omega t - \phi - \delta) \qquad v(t) = \sqrt{2}|V| \cos(\omega t - \delta) \tag{3.4}$$

where $\delta = \{0, 2\pi/3, 4\pi/3\}$ rad is the phase shift between the three phases and ϕ is the phase shift between current and voltage.

3.1.2 Phasor notation

Steady-state power flow analysis can be considerably simplified by using *phasors* to represent sinusoidal current and voltage functions. A phasor is an arrow that is imagined to spin in the complex plane and it characterizes a sine wave by specifying its magnitude and angle. Length of the phasor corresponds to amplitude or rms value, its angle with respect to real axis corresponds to time and its rotation corresponds to angular frequency which is constant and is generally not considered for steady-state power flow calculations. Considering sinusoidal current and voltage expressions:

$$i(t) = I_{max} \cos(\omega t - \delta_I) \quad \text{and} \quad v(t) = V_{max} \cos(\omega t - \delta_V),$$

we use Euler's identity $e^{j\phi} = \cos\phi + j\sin\phi$ and obtain,

$$\begin{aligned} i(t) &= \sqrt{2}\mathrm{Re}(|I|e^{j\delta_I}e^{j\omega t}) & v(t) &= \sqrt{2}\mathrm{Re}(|V|e^{j\delta_V}e^{j\omega t}) \\ &= \sqrt{2}\mathrm{Re}(Ie^{j\omega t}) & &= \sqrt{2}\mathrm{Re}(Ve^{j\omega t}) \end{aligned} \tag{3.5}$$

where,

$$I = |I|e^{j\delta_I} \quad \text{and} \quad V = |V|e^{j\delta_V}$$

Here I and V are current and voltage phasors. In a balanced three phase system, current and voltage values of one phase can be used to determine values of other phases just by accordingly rotating the phasors.

3.1.3 Power

Considering the phase with $\delta = 0$ and equations (3.4), instantaneous power can be expressed as,

$$\begin{aligned} p(t) &= v(t)i(t) \\ &= \sqrt{2}|V|\cos(\omega t)\sqrt{2}|I|\cos(\omega t - \phi) \\ &= |V||I|\cos\phi[1 + \cos(2\omega t)] + |V||I|\sin\phi[\sin(2\omega t)] \\ &= P[1 + \cos(2\omega t)] + Q[\sin(2\omega t)] \end{aligned} \tag{3.6}$$

where $P = |V||I|\cos\phi$ is called active power and $Q = |V||I|\sin\phi$ is called reactive power.

As evident from equation (3.6), instantaneous power is made up of two sinusoidal components. The first component $P[1 + \cos(2\omega t)]$ is unidirectional with average value P and the second component $Q[\sin(2\omega t)]$ is bidirectional with an average of 0.

Active power P is measured in watts $[W]$. It represents the power actually transmitted or consumed by loads and is always positive. For instance, for purely resistive loads, active power corresponds entirely to conversion of electric energy to heat or light. Active power is also called real power or average power.

Reactive power Q is expressed in volt-ampere reactive $[Var]$. For loads with reactance, phase difference between current and voltage ϕ is not zero and it results in instantaneous power sometimes being negative which can be interpreted as power flowing backwards from the load to the generator. This power that is oscillated back and forth through the lines is exchanged between electric and magnetic fields and is not dissipated [4]. Reactive power is also called imaginary power.

Power factor, often abbreviated as p.f. is defined by $\cos\phi$. When current lags voltage, ϕ is positive and power factor is said to be lagging. When current leads voltage, ϕ is negative and power factor is said to be leading. As ϕ varies from 0 to $90°$, p.f. varies from 1 to 0 corresponding to the loads from being purely resistive to purely inductive.

The vector sum of P and Q is called complex power and is expressed as,

$$\mathbf{S} = VI^* = P + jQ \tag{3.7}$$

where V and I are voltage and current phasors and I^* is the complex conjugate of I. Another important quantity is *apparent power* which is generally used to specify the rating of an electrical apparatus. It is the product of current and voltage, regardless of their phase shift. It is measured in volt-amperes [VA] and is written as,

$$S = |V||I| \tag{3.8}$$

3.1.4 Impedance and Admittance

In AC circuits, the opposition to flow of current is called impedance which is the vector sum of resistance and reactance. It is given by,

$$\mathbf{Z} = R + jX \tag{3.9}$$

where R is *resistance* (real part) and X is *reactance* (imaginary part). Impedance is measured in ohms [Ω] and comes with every device in an AC circuit. When X is positive, reactance is inductive and $jX = j\omega L$ where L is the inductance. When X is negative, reactance is capacitive and $jX = 1/j\omega C$ where C is the capacitance. Note that R, L and C are always positive. When X is zero, impedance is purely resistive, indicating that there are no inductors and capacitors in the circuit.

The inverse of impedance is called admittance denoted by $\mathbf{Y}$. It is expressed as,

$$\mathbf{Y} = 1/\mathbf{Z} = G + jB \tag{3.10}$$

where G is called *conductance* and B, *susceptance*. Considering the magnitudes of G and B, admittance can be written as,

$$\mathbf{Y} = \frac{R}{Z^2} - j\frac{X}{Z^2} \tag{3.11}$$

where Z is the magnitude of $\mathbf{Z}$. Conductance G, susceptance B and hence admittance $\mathbf{Y}$ are measured in siemens [S].

Furthermore, Ohm's law is extended to AC circuits as follows.

$$V = ZI \text{ or } I = YV \tag{3.12}$$

3.1.5 Kirchhoff's Circuit Laws

Kirchhoff's Voltage Law (KVL) states that the sum of voltages around any closed loop in a circuit must be zero.

$$\sum_j V_j = 0 \tag{3.13}$$

where V_j is the voltage across component j in the closed loop.

Kirchhoff's Current Law (KCL) states that the currents entering and leaving any node in the circuit must add up to zero.

$$\sum_k I_{ik} = 0 \tag{3.14}$$

where I_{ik} is the current flowing from node i to node k.

Kirchhoff's laws are extensively used to calculate currents and voltages in electrical circuits.

3.1.6 Per unit system

It is common practice to normalize numerical values for ease of calculation. In *per unit* (pu) system, the quantities of interest are expressed in terms of *base value* as follows.

$$\text{per unit value} = \frac{\text{actual value}}{\text{base value}}$$

The per unit value is dimensionless. In power system analysis, voltages, currents, impedances and powers are normalized [7].

3.2 Network Topology

The transmission system is modelled as a directed graph of nodes and edges. Nodes are called *buses* and they represent points in the circuit where system components such as loads, generators, transformers, phase shifters, shunts or substations are connected. Edges are transmission *lines* that connect buses and hence system components to each other. In this section, we look at the topology in detail and describe how the components are modelled.

3.2.1 Buses and lines

Buses are considered to be electrically distinct, meaning that there exists an impedance between them which sustains a potential difference. Each bus is characterized by the following four quantities.

- Voltage phasor magnitude, V
- Voltage phase angle, δ
- Injected active power, P
- Injected reactive power, Q

Furthermore, buses are distinguished based on the parameters specified or controlled by components that they are connected to. Following three types are commonly used in power flow analysis.

Load Bus: As loads signify power consumption, they are modelled as such. At each load bus, active power and reactive power are specified, which together constitute negative power injection[1]. A load bus is called PQ bus, suggesting that P and Q are known. V and δ are unknown, corresponding to the fact that loads do not control voltage.

Generator Bus: Generators are known to have control over active power and voltage. Hence, a generator bus is referred to as PV bus. However, wind turbines do not have voltage control and they are treated as PQ buses with a positive P. Another exception is that some generators supply only active power and they are modelled as PQ buses with $Q = 0$. The following reason clarifies this approach.

As mentioned earlier, supply and demand in an electric grid should always be in balance. This is achieved by matching generation and consumption of active and reactive power. Generators are solely responsible for active power balance, whereas reactive power can also be balanced by using devices such as shunts[2]. Hence, generators are modelled as PV or PQ buses depending upon the parameters they control.

Slack Bus: The challenge with achieving power balance is that generation should also account for transmission losses which are not known in advance. For active power, the trick in practice is to make an empirical assumption of what the losses could be and get a fixed power dispatch from all generators except one[3]. This generator's output is said

[1]An exchange of power between the network and a device such as a load or a generator is called power injection.

[2]Shunts are devices that inject or consume reactive power.

[3]Could also be a few but as done in literature, we consider them as one to explain a slack bus.

to be controllable. It takes up the slack by generating more power if losses are greater than expected or less power if losses are smaller. Likewise, in power flow analysis, the slack bus or swing bus is analogous to the variable generator. As real power balance is a manifestation of steady frequency and hence of voltage angle, the phase angle δ is specified for the slack bus. On the other hand, slack in reactive power is shared by shunts and all generators that dispatch reactive power. Hence, V is specified for the slack bus as it is equivalent to requiring a balanced reactive power[4]. Note that it is convention to use $\delta = 0$ for the slack bus.

Table 3.1 summarizes the above described distinction of buses and helps visualize the parameters. N is the total number of buses in the network and N_g is the number of generator buses.

Table 3.1: Bus types and variables

Bus type	Number of buses	Known	Unknown
Slack bus	1	$\delta, \lvert V\rvert$	P, Q
PV bus	N_g	$P, \lvert V\rvert$	δ, Q
PQ bus	$N - N_g - 1$	P, Q	$\delta, \lvert V\rvert$

Buses such as transmission substations that are not connected to generators or loads are modelled as loads with $P = Q = 0$. A bus can also have both generator and load connected. Such buses are modelled as PV buses with $P = P_{gen} + P_{load}$.

A transmission line is modelled as an impedance between two nodes i and j. The impedances of transmission lines are assumed to be time-invariant under any electric potential and current. This allows the application of Ohm's law to determine line currents. Since a balanced three phase system is modelled considering the single phase equivalent, all transmission lines in the model correspond to one phase out of the three.

3.2.2 Tap Transformers and Shunts

The frequency of a transmission system, sometimes called system frequency, is constant throughout the network whereas the voltage is not. The voltage largely depends on the local situation of the system and as a consequence, can only be controlled locally [3]. Tap transformers and shunts are system components that play a very important role in controlling the voltage across the network.

[4]For a more detailed description about buses and choice of variables, we refer to [4], section 7.2.

A tap transformer, also called tap-changing transformer is a transformer in which the turns ratio[5] can be adjusted. A mechanical tap is used to adjust the ratio. The voltages on either side of the transformer are related as,

$$|V_2| = \frac{|V_1|}{t}$$

where t is the turns ratio.

As there is a strong relation between reactive power exchange and voltage levels, shunts are used to balance voltage levels by consuming or injecting reactive power. At a network bus, reactive power consumption results in a lower bus voltage and reactive power injection results in a higher bus voltage. Shunt capacitors inject reactive power whereas shunt inductors consume reactive power. A shunt is modelled as a reactance $z_s = jx_s$ between the bus and the ground. The shunt admittance is defined as follows.

$$y_s = \frac{1}{z_s} = -j\frac{1}{x_s} = jb_s$$

For inductive shunts, x_s is positive and for capacitive shunts, x_s is negative. Note that the shunt susceptance is $b_s = -1/x_s$.

3.3 The Power Flow Model

As the name implies, power flow or load flow simulations involve understanding the flow of electric power from source to destination. Power flow gives insight about the *state* of the transmission system and is one of the most important network computations. State, also referred to as grid state, describes *steady-state* behavior of the network and is defined by three quantities; power, current and voltage. Steady-state means that only power frequency (50 or 60 Hz) is considered for calculations and the time step could be minutes, hours, months or years[6]. Given power injections at different parts of the network, the objective is to compute voltage at every node and current in every line. We derive power flow equations and define the power flow problem as follows.

At each node i of the network, complex power is defined by,

$$S_i = V_i I_i^* \tag{3.15}$$

[5]Turns ratio is the ratio between number of windings on primary and secondary sides of a transformer.

[6]Dynamic(kHz) and transient(MHz) analyses consider milliseconds and microseconds respectively for calculations.

where V_i is the potential difference between the node i and ground, and I_i is the current *injected* at node i. From Kirchhoff's Current Law we have,

$$I_i = \sum_{k=1}^{N} I_{ik} \tag{3.16}$$

where I_{ik} is the current between node i and node $k \neq i$. That is, it's the current flowing from every node in the network to node i. From Ohm's law, line current is related to voltage as,

$$I_{ik} = Y_{ik} V_k \tag{3.17}$$

where V_k is the voltage at node[7] k and Y_{ik} is the admittance[8] of the transmission line joining nodes i and k. In matrix form,

$$I = YV \tag{3.18}$$

where $I \in \mathbb{C}^N$ is the vector of current injections at nodes, $V \in \mathbb{C}^N$ is the vector of node voltages and $Y = [Y_{ik}] \in \mathbb{C}^{N\times N}$ is called *admittance matrix*. The entries $[Y_{ik}]$ define the line impedance between node i and node k. From (3.10) we have, $Y_{ik} = G_{ik} + jB_{ik}$. For nodes not directly connected to node i, $Y_{ik} = 0$ and hence Y is sparse and in KCL, it is sufficient to sum only over nodes that are *directly* connected to node i.

Complex power at node i can now be written as,

$$\begin{aligned} S_i &= V_i (YV)_i^* \\ &= V_i \left(\sum_{k=1}^{N} Y_{ik} V_k \right)^* \end{aligned} \tag{3.19}$$

Using phasors and expanding Y_{ik},

$$\begin{aligned} S_i &= \sum_{k=1}^{N} |V_i||V_k| e^{j\delta_{ik}} (G_{ik} - jB_{ik}) \\ &= \sum_{k=1}^{N} |V_i||V_k| \left(\cos\delta_{ik} + j\sin\delta_{ik}\right) \left(G_{ik} - jB_{ik}\right)) \end{aligned}$$

[7]From Ohm's law, V_k should have been the voltage drop across the impedance but we consider it as voltage at node k for now, as we will see further that the potential difference between nodes i and k arises in the power flow equations once we introduce phasors.

[8]It is convenient to use admittance Y instead of impedance Z as we can define the admittance between two unconnected nodes as 0.

where $\delta_{ik} = (\delta_i - \delta_k)$ denotes the difference in phase angles between node i and k. Considering the real and imaginary terms of complex power S_i, we have the following two equations for active and reactive power which are called *power flow equations*.

$$P_i = \sum_{k=1}^{N} |V_i||V_k| \left(G_{ik} \cos \delta_{ik} + B_{ik} \sin \delta_{ik}\right) \tag{3.20a}$$

$$Q_i = \sum_{k=1}^{N} |V_i||V_k| \left(G_{ik} \sin \delta_{ik} - B_{ik} \cos \delta_{ik}\right) \tag{3.20b}$$

The power flow problem, also called load flow problem can now be stated as:

Given the power injection S at each node,
find the voltage V ***at every node and current*** I ***in every line.***

This problem is solved by computing V from the power flow equations and then computing I using Ohm's law and KCL.

The power flow equations form a system of non-linear equations for which a closed-form solution is not known to exist. However, it is a root-finding problem and we use well established methods such as the Newton Raphson iterative algorithm to find a numerical solution.

3.4 Applications

Solving the power flow problem is of tremendous importance since it lies at the root of various applications in power system analysis. In this project we focus on the following applications which are important for TenneT.

3.4.1 Year-round simulations

As discussed in chapter 1, year-round AC power flow simulations involve performing simulations for an electricity grid with changing hourly load and generation data. Given consumption and generation data of each hour of a year, the power flow problem is solved to determine the voltage in each bus and current in each line of the network. This gives insight into the steady-state behavior of the electricity grid for an entire year or a decade. Year-round simulations are required in TenneT to plan the infrastructure needed to best prepare the grid for the future. Due to convergence problems, AC power flow simulations are currently performed in TenneT only for a few critical hours. The load and generation that could possibly cause the currents in lines to go beyond the safe operating limit are chosen for simulations and the behavior of the grid is studied for those few hours. The critical hours are chosen based on the results of an approximate method called the DC load flow method (described in the next chapter) which is a linear approximation of the power flow problem that assumes that the voltages across the network are $1\,\mathrm{pu}$. Hence, the selection of critical hours is not guaranteed to be accurate. For this reason, it is imperative that AC power flow simulations are performed to determine the actual flow of electric power in the network.

3.4.2 Reactive power compensation

As we know, the voltages across the transmission network should be within safe limits throughout the year to ensure safe transport of electric power. The voltages in the transmission network can be controlled by balancing the reactive power in the system. Generally, there are two possible ways to achieve reactive power balance. One way is to let the generators exchange reactive power with the network in such a way that the voltages at the generator buses are maintained at predefined levels. The predefined voltage levels are called voltage *setpoints*. For example, if the voltage setpoint at a generator bus is $1.05\,\mathrm{pu}$, the generator exchanges the right amount of reactive power with the network to make sure that the voltage at its bus is $1.05\,\mathrm{pu}$. Another way is to use shunts to balance reactive power in the network. As explained in section 3.2.2, a shunt is a device that can be used to exchange reactive power with the network and hence keep the voltages within safe limits. For a TSO, buying reactive power from generators is expensive and thus it is best practice to use shunts in the network to keep voltages within safe limits throughout the year. Reactive power compensation assessments are made to determine the amount of shunts that are needed in the network for the future.

3.4.3 Contingency analysis

In electricity systems, contingency analysis is an investigation of scenarios where system components such as generators or transmission lines are out of service or are taken down for maintenance. A standard criterion in contingency analysis is the N-1 criterion which is often called N-1 secure (for normal minus one). If an electric grid is called N-1 secure, it means that the grid should be functional even if one system component such as a major transmission line is out of service. For higher security, some electric grids are made N-2 secure in which case the grid should be able to withstand two contingencies, that is, the loss of two system components.

We would like to reiterate that currently in TenneT, due to convergence problems, there is some compromise with all the tasks discussed above. Either an approximate method such as the DC load flow method is used or AC power flow simulations are performed only for a few cases. Even with the few cases, convergence problems persist and are currently being solved by trial and error techniques. For example by adding shunts or more slack buses in the system to ensure power balance. This is certainly not an optimal solution and there is no guarantee that it works for all cases. Even if a solution is found using trial and error techniques, the voltages in the system might not lie within safe limits. Hence, in this project we develop the automating algorithm that can be used in TenneT to perform year-round AC power flow simulations and do reactive power compensation assessments and contingency analyses without having to manually solve convergence problems. We will show that the algorithm, which is based on a mathematical framework, finds optimal solutions to convergence problems.

Chapter 4

Power Flow Solvers

Digital solution methods to solve the power flow problem first appeared in the literature in 1956 and major breakthroughs in power flow computations were made in the 1960s [8]. There has been a lot of research in numerical methods to efficiently solve the power flow problem ever since. In recent times, considering the rapidly growing sizes of electric grids, power flow solvers are of tremendous importance to power system operators.

Since the performance of a power flow solver depends largely on various factors such as problem size, available computing power and ways of implementation, it is often hard to choose the right solver for a given problem. In this chapter, we describe some of the most widely used power flow solvers.

4.1 Newton-Raphson

The Newton-Raphson (NR) method is a widely accepted root-finding algorithm that can be used to solve a system of non-linear equations of the form $F(x) = 0$. Starting with an initial approximation, the iterative scheme involves making successive corrections to vector x. The correction vector Δx is assumed to satisfy $F(x+\Delta x) = 0$ at each iteration and a first order Taylor expansion of $F(x + \Delta x)$ gives the NR iterative formula

$$-J(x)\Delta x = F(x) \tag{4.1}$$

where $J(x)$ is the Jacobian matrix, hereafter called as the Jacobian, and is calculated as $J_{ik} = \frac{\partial F_i(x)}{\partial x_k}$. The partial derivatives represent the slopes of the tangent hyperplanes [8]. Algorithm (1) describes the basic structure of the Newton-Raphson method. Traditionally, direct solvers are used to solve the linear system (4.1) in each iteration. The residual norm $||F(x^k)||$ or the relative residual norm $||\frac{F(x^k)}{F(x^0)}||$ is used as a measure to check convergence. The Newton-Raphson method is known to have quadratic convergence when iterates are close to the solution [7]. The iteration process of the Newton-Raphson method for a one-dimensional function is shown in fig. 4.1.

Algorithm 1: Newton-Raphson Method

```
$i := 0$
Initialize: $x^0$
**while** *not converged* **do**
    Solve for the correction: $-J(x^i)\Delta x^i = F(x^i)$
    Update the approximation: $x^{i+1} = x^i + \Delta x^i$
    $i = i + 1$
**end**
```

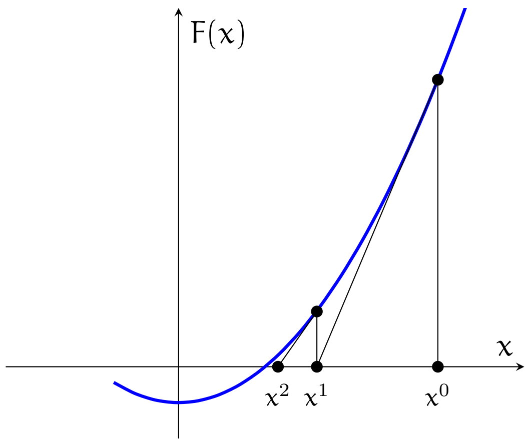


Figure 4.1: Newton-Raphson iterations

The classical approach to initialize x^0 is to use the *flat start* as initial value. That is, all voltage angles are set to 0 and voltage magnitudes are set to $1\,\mathrm{pu}$ (equal to that of the slack bus). For better convergence, the solutions of approximate methods such as DC approximation (described in section 4.3) are used as initial values.

To solve the power flow problem using the NR method, $F(x)$ can be formulated as power or current mismatch functions. The unknown variable vector x can be represented in three different coordinates such as polar, Cartesian and complex form as shown in table 4.1.

The two mismatch formulations of $F(x)$ and three coordinate forms of x result in six possible ways of applying the Newton-Raphson method to solve power flow problems. These six methods are considered as the fundamental Newton power flow methods based on which various modified versions are developed [9].

Table 4.1: Variable x in different coordinates

Coordinates	Variable x
Polar $\left(V_i = \|V_i\|e^{j\delta_i}\right)$	$[\delta_1, \ldots, \delta_n, \|V_1\|, \ldots, \|V_n\|]^T$
Cartesian $(V_i = V_i^r + jV_i^m)$	$[V_1^m, \ldots, V_n^m, V_1^r, \ldots, V_n^r]^T$
Complex form (V_i)	$[V_1, \ldots, V_n]^T$

The most widely used version is power-mismatch formulation with polar coordinates [10]. The current-mismatch versions with polar and Cartesian coordinates developed in [9] are found to perform well for large scale transmission systems. In this report, we review the three versions; polar power-mismatch version as described in [10], polar and Cartesian current-mismatch versions as developed in [9]. For a detailed comparison of NR methods, we refer to [9] where all six versions are investigated with numerical experiments and a general framework for applying NR methods to power flow problems in transmission and distribution systems is presented.

4.1.1 Polar power-mismatch version

The power-mismatch function $F(x)$ is formulated as,

$$\begin{aligned} F_i(x) = \Delta S_i(x) &= S_i^{sp} - S_i(x) \\ &= S_i^{sp} - V_i \sum_{k=1}^{N} Y_{ik}^* V_k^* \end{aligned} \tag{4.2}$$

where $S_i^{sp} = P_i^{sp} + jQ_i^{sp}$ is the specified complex power injection at bus i and $S_i(x)$ is the complex power computed at bus i which follows from (3.19).

In polar coordinates, $\Delta S_i(x)$ is separated using (3.20) as,

$$\Delta P_i(x) = P_i^{sp} - \sum_{k=1}^{N} |V_i||V_k| \left(G_{ik} \cos \delta_{ik} + B_{ik} \sin \delta_{ik}\right) \tag{4.3a}$$

$$\Delta Q_i(x) = Q_i^{sp} - \sum_{k=1}^{N} |V_i||V_k| \left(G_{ik} \sin \delta_{ik} - B_{ik} \cos \delta_{ik}\right) \tag{4.3b}$$

Using the polar power-mismatch function, the Newton-Raphson iterative formula can be written as follows.

$$-\left[\begin{array}{c|c} J^{11} & J^{12} \\ \hline J^{21} & J^{22} \end{array}\right] \left[\begin{array}{c} \Delta\delta \\ \Delta|V| \end{array}\right] = \left[\begin{array}{c} \Delta P \\ \Delta Q \end{array}\right] \tag{4.4}$$

where the Jacobian sub-matrices are defined as $J^{11} = \frac{\partial \Delta P}{\partial \delta}$, $J^{12} = \frac{\partial \Delta P}{\partial |V|}$, $J^{21} = \frac{\partial \Delta Q}{\partial \delta}$, $J^{22} = \frac{\partial \Delta Q}{\partial |V|}$ and the partial derivatives $J_{ik} = \frac{\partial F_i(x)}{\partial x_k}$ are calculated as follows.

$$\begin{aligned}
\frac{\partial \Delta P_i(x)}{\partial |V_k|} &= -|V_i|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik}) \\
\frac{\partial \Delta Q_i(x)}{\partial |V_k|} &= -|V_i|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) \\
\frac{\partial \Delta P_i(x)}{\partial \delta_k} &= -|V_i||V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) \\
\frac{\partial \Delta Q_i(x)}{\partial \delta_k} &= -|V_i||V_k|(-G_{ik}\cos\delta_{ik} - B_{ik}\sin\delta_{ik})
\end{aligned} \qquad (i \neq k)$$

$$\begin{aligned}
\frac{\partial \Delta P_i(x)}{\partial |V_i|} &= -\left(2|V_i|G_{ii} + \sum_{i\neq k}|V_k|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik})\right) \\
\frac{\partial \Delta Q_i(x)}{\partial |V_i|} &= -\left(-2|V_i|B_{ii} + \sum_{i\neq k}|V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik})\right) \\
\frac{\partial \Delta P_i(x)}{\partial \delta_i} &= -\sum_{i\neq k}|V_i||V_k|(-G_{ik}\sin\delta_{ik} + B_{ik}\cos\delta_{ik}) \\
\frac{\partial \Delta Q_i(x)}{\partial \delta_i} &= -\sum_{i\neq k}|V_i||V_k|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik})
\end{aligned} \qquad (i = k)$$

To solve the linear system (4.4), it has to be modified based on the information available at each bus for the following reason. We know from section 3.2.1 that at each PV bus, P and |V| are specified whereas Q and δ are unknown. Hence, for each PV bus j, ΔQ_j cannot be formulated and the corresponding partial derivatives in the Jacobian cannot be computed. As a result, we eliminate the entries $\frac{\partial \Delta P_i}{\partial |V_j|}$, $\frac{\partial \Delta Q_i}{\delta |V_j|}$, $\frac{\partial \Delta Q_j}{\partial |V_i|}$ and $\frac{\partial \Delta Q_j}{\partial \delta_i}$ for all $i = 1 \ldots N$ from the Jacobian $J(x)$, $\Delta|V_j|$ from the correction vector Δx and ΔQ_j from the power mismatch function $F(x)$ for each PV bus j. Similarly, δ and $|V|$ are known for the slack bus and the corresponding entries in the linear system are eliminated. The order of $J(x)$ reduces to $(2N - N_g - 2)$ and the vector x becomes,

$$x = \left[\delta_2, \ldots, \delta_N, |V_{N_g+2}|, \ldots, |V_N|\right]^T$$

Note that conventionally, δ_1 and $|V_1|$ correspond to the slack bus. The modified linear system is solved at each NR iteration.

4.1.2 Polar current-mismatch version

The current-mismatch function is formulated using the current equation (3.16) and the complex power equation (3.15) as follows.

$$\begin{aligned} F_i(x) = \Delta I_i(x) &= I_i^{sp} - I_i(x) \\ &= \left(\frac{S_i^{sp}}{V_i}\right)^* - \sum_{k=1}^{N} Y_{ik} V_k \end{aligned} \tag{4.5}$$

where $I_i^{sp} = \left(\frac{S_i^{sp}}{V_i}\right)^*$ is the complex current injection specified at bus i and $I_i(x)$ is the complex current computed at bus i.

The function $\Delta I_i(x)$ is separated into real $\Delta I_i^r(x)$ and imaginary $\Delta I_i^m(x)$ current-mismatch functions in polar form as follows.

$$\Delta I_i^r(x) = \frac{P_i^{sp}\cos\delta_i + Q_i^{sp}\sin\delta_i}{|V_i|} - \sum_{k=1}^{N} |V_k|\left(G_{ik}\cos\delta_k - B_{ik}\sin\delta_k\right) \tag{4.6a}$$

$$\Delta I_i^m(x) = \frac{P_i^{sp}\sin\delta_i - Q_i^{sp}\cos\delta_i}{|V_i|} - \sum_{k=1}^{N} |V_k|\left(G_{ik}\sin\delta_k + B_{ik}\cos\delta_k\right) \tag{4.6b}$$

The NR iterative formula for the polar current-mismatch version can be written as,

$$-\left[\begin{array}{c|c} J^{11} & J^{12} \\ \hline J^{21} & J^{22} \end{array}\right]\left[\begin{array}{c} \Delta\delta \\ \Delta|V| \end{array}\right] = \left[\begin{array}{c} \Delta I^r \\ \Delta I^m \end{array}\right] \tag{4.7}$$

where $J^{11} = \frac{\partial \Delta I^r}{\partial \delta}$, $J^{12} = \frac{\partial \Delta I^r}{\partial |V|}$, $J^{21} = \frac{\partial \Delta I^m}{\partial \delta}$, and $J^{22} = \frac{\partial \Delta I^m}{\partial |V|}$. The partial derivatives are calculated as follows.

$$\left.\begin{aligned} \frac{\partial \Delta I_i^r(x)}{\partial |V_k|} &= -(G_{ik}\cos\delta_k - B_{ik}\sin\delta_k) \\ \frac{\partial \Delta I_i^m(x)}{\partial |V_k|} &= -(G_{ik}\sin\delta_k + B_{ik}\cos\delta_k) \\ \frac{\partial \Delta I_i^r(x)}{\partial \delta_k} &= |V_k|(G_{ik}\sin\delta_k + B_{ik}\cos\delta_k) \\ \frac{\partial \Delta I_i^m(x)}{\partial \delta_k} &= -|V_k|(G_{ik}\cos\delta_k - B_{ik}\sin\delta_k) \end{aligned}\right. \qquad (i \neq k)$$

$$\begin{aligned}
\frac{\partial \Delta I_i^r(x)}{\partial |V_i|} &= -\left(G_{ii}\cos\delta_i - B_{ii}\sin\delta_i\right) - \frac{P_i^{sp}\cos\delta_i + Q_i^{sp}\sin\delta_i}{|V_i|^2} \\
\frac{\partial \Delta I_i^m(x)}{\partial |V_i|} &= -\left(G_{ii}\sin\delta_i + B_{ii}\cos\delta_i\right) - \frac{P_i^{sp}\sin\delta_i - Q_i^{sp}\cos\delta_i}{|V_i|^2} \\
\frac{\partial \Delta I_i^r(x)}{\partial \delta_i} &= |V_i|\left(G_{ii}\sin\delta_i + B_{ii}\cos\delta_i\right) - \frac{P_i^{sp}\sin\delta_i - Q_i^{sp}\cos\delta_i}{|V_i|} \\
\frac{\partial \Delta I_i^m(x)}{\partial \delta_i} &= -|V_i|\left(G_{ii}\cos\delta_i - B_{ii}\sin\delta_i\right) + \frac{P_i^{sp}\cos\delta_i + Q_i^{sp}\sin\delta_i}{|V_i|}
\end{aligned} \qquad (i = k)$$

Similar to power-mismatch version, the linear system (4.7) has to be modified. For a PQ bus, computation of real and imaginary current-mismatch functions is straightforward since the associated real and reactive power mismatches are known. Whereas representing a PV bus in the linear system is tricky and there are several approaches available in literature. In this report, we review the new approach developed in [9] which is found to be promising.

For each PV bus j, the reactive power Q_j is considered as a dependent variable of $|V|$ and δ. The current-mismatch formulation is directly used. That is, $\Delta I_i^r(x)$ and $\Delta I_i^m(x)$ are calculated using $Q_i^{sp} = Q_j$ in equations (4.6) for each PV bus j at each iteration. The partial derivatives $\frac{\partial \Delta I_i^r}{\partial |V_j|}$ and $\frac{\partial \Delta I_i^m}{\partial |V_j|}$ in the Jacobian $J(x)$ are replaced by $\frac{\partial \Delta I_i^r}{\partial Q_j}$ and $\frac{\partial \Delta I_i^m}{\partial Q_j}$ for all $i = 1 \ldots N$ which are calculated as follows.

$$\begin{aligned}
\frac{\partial \Delta I_i^r(x)}{\partial Q_j} &= 0 \\
\frac{\partial \Delta I_i^m(x)}{\partial Q_j} &= 0
\end{aligned} \qquad (i \neq j)$$

$$\begin{aligned}
\frac{\partial \Delta I_j^r(x)}{\partial Q_j} &= \frac{\sin\delta_j}{|V_j|^{sp}} \\
\frac{\partial \Delta I_j^r(x)}{\partial Q_j} &= -\frac{\cos\delta_j}{|V_j|^{sp}}
\end{aligned} \qquad (i = j)$$

The entries $\Delta|V_j|$ in the correction vector Δx are replaced by ΔQ_j for each PV bus j. The initial reactive power Q_j^0 is calculated for each PV bus j as follows.

$$Q_j^0 = \sum_{k=1}^{N} |V_j||V_k|\left(G_{jk}\sin\delta_{jk} - B_{jk}\cos\delta_{jk}\right)$$

The order of $J(x)$ remains $(2N-2)$. At each NR iteration, the modified linear system is solved and the reactive power Q_j is updated using the computed correction ΔQ_j.

4.1.3 Cartesian current-mismatch version

In Cartesian form, the current-mismatch function $F(x)$ is separated as,

$$\Delta I_i^r(x) = \frac{P_i^{sp} V_i^r + Q_i^{sp} V_i^m}{(V_i^r)^2 + (V_i^m)^2} - \sum_{k=1}^{N} (G_{ik} V_k^r - B_{ik} V_k^m) \tag{4.8a}$$

$$\Delta I_i^m(x) = \frac{P_i^{sp} V_i^m - Q_i^{sp} V_i^r}{(V_i^r)^2 + (V_i^m)^2} - \sum_{k=1}^{N} (G_{ik} V_k^m + B_{ik} V_k^r) \tag{4.8b}$$

and the Jacobian matrix equation is formulated as follows.

$$-\left[\begin{array}{c|c} J^{11} & J^{12} \\ \hline J^{21} & J^{22} \end{array}\right] \begin{bmatrix} \Delta V^m \\ \Delta V^r \end{bmatrix} = \begin{bmatrix} \Delta I^r \\ \Delta I^m \end{bmatrix} \tag{4.9}$$

where $J^{11} = \frac{\partial \Delta I^r}{\partial V^m}$, $J^{12} = \frac{\partial \Delta I^r}{\partial V^r}$, $J^{21} = \frac{\partial \Delta I^m}{\partial V^m}$, and $J^{22} = \frac{\partial \Delta I^m}{\partial V^r}$. The partial derivatives are computed as,

$$\begin{aligned} \frac{\partial \Delta I_i^r(x)}{\partial V_k^r} &= -G_{ik} \\ \frac{\partial \Delta I_i^m(x)}{\partial V_k^r} &= B_{ik} \\ \frac{\partial \Delta I_i^r(x)}{\partial V_k^m} &= B_{ik} \\ \frac{\partial \Delta I_i^m(x)}{\partial V_k^m} &= -G_{ik} \end{aligned} \qquad (i \neq k)$$

$$\begin{aligned} \frac{\partial \Delta I_i^r(x)}{\partial V_i^r} &= -G_{ii} - \frac{P_i^{sp}\left((V_i^r)^2 - (V_i^m)^2\right) + 2V_i^r V_i^m Q_i^{sp}}{|V_i|^4} \\ \frac{\partial \Delta I_i^m(x)}{\partial V_i^r} &= -B_{ii} + \frac{Q_i^{sp}\left((V_i^r)^2 - (V_i^m)^2\right) - 2V_i^r V_i^m P_i^{sp}}{|V_i|^4} \\ \frac{\partial \Delta I_i^r(x)}{\partial V_i^m} &= B_{ii} + \frac{Q_i^{sp}\left((V_i^r)^2 - (V_i^m)^2\right) - 2V_i^r V_i^m P_i^{sp}}{|V_i|^4} \\ \frac{\partial \Delta I_i^m(x)}{\partial V_i^m} &= -G_{ii} + \frac{P_i^{sp}\left((V_i^r)^2 - (V_i^m)^2\right) + 2V_i^r V_i^m Q_i^{sp}}{|V_i|^4} \end{aligned} \qquad (i = k)$$

In [9], the reactive power Q_j is chosen as a dependent variable to represent a PV bus, similar to polar current-mismatch version. The partial derivatives $\frac{\partial \Delta I_i^r}{\partial Q_j}$ and $\frac{\partial \Delta I_i^m}{\partial Q_j}$ computed as,

$$\begin{aligned} \frac{\partial \Delta I_i^r(x)}{\partial Q_j} &= 0 \\ \frac{\partial \Delta I_i^m(x)}{\partial Q_j} &= 0 \end{aligned} \qquad (i \neq j)$$

$$\begin{aligned} \frac{\partial \Delta I_j^r(x)}{\partial Q_j} &= \frac{V_j^m}{(V_j^r)^2 + (V_j^m)^2} \\ \frac{\partial \Delta I_j^r(x)}{\partial Q_j} &= \frac{-V_j^r}{(V_j^r)^2 + (V_j^m)^2} \end{aligned} \qquad (i = j)$$

are added to the Jacobian matrix $J(x)$ and the correction ΔQ_j is added to the correction vector Δx for each PV bus j. As a result, the Jacobian matrix becomes a rectangular matrix. That is, $J(x) \in \mathbb{R}^{(2N)\times(2N+N_g)}$. In order to make the Jacobian matrix square, the equation

$$\Delta|V| = \frac{V^r}{|V|}\Delta V^r + \frac{V^m}{|V|}\Delta V^m$$

is used with $\Delta|V_j| = 0$ since $|V_j|$ is specified for each PV bus j. This gives the relation,

$$\Delta V_j^r = -\frac{V_j^m}{V_j^r}\Delta V_j^m$$

which is used to add the column of the Jacobian corresponding to the derivatives $\frac{\partial \Delta I_i^r}{\partial V_j^r}$ and $\frac{\partial \Delta I_i^m}{\partial V_j^r}$ to the column corresponding to the derivatives $\frac{\partial \Delta I_i^r}{\partial V_j^m}$ and $\frac{\partial \Delta I_i^m}{\partial V_j^m}$ as follows.

$$\begin{aligned} \frac{\partial \Delta I_i^r}{\partial V_j^m}\Delta V_j^m &= \left(\frac{\partial \Delta I_i^r}{\partial V_j^m} - \frac{V_j^m}{V_j^r}\frac{\partial \Delta I_i^r}{\partial V_j^r}\right)\Delta V_j^m \\ \frac{\partial \Delta I_i^m}{\partial V_j^m}\Delta V_j^m &= \left(\frac{\partial \Delta I_i^m}{\partial V_j^m} - \frac{V_j^m}{V_j^r}\frac{\partial \Delta I_i^m}{\partial V_j^r}\right)\Delta V_j^m \end{aligned}$$

The correction vector ΔV_j^r is now eliminated from the correction vector Δx for each PV bus j. The initial reactive power Q_j^0 is calculated for each PV bus j as follows.

$$Q_j^0 = \sum_{k=1}^{N}\left(V_j^m\left(G_{jk}V_k^r - B_{jk}V_k^m\right) - V_j^r\left(B_{jk}V_k^r + G_{jk}V_k^m\right)\right)$$

With the slack bus included, the order of $J(x)$ remains $(2N-2)$. At each NR iteration, the modified linear system is solved and the reactive power Q_j is updated using the computed correction ΔQ_j.

Despite its widespread popularity, a drawback the Newton-Raphson method has is computational complexity. The Jacobian has to be computed in every iteration as it depends on the current approximation of the solution. Which means, there is a new linear system (4.1) in every iteration for the algorithm to solve. This makes the solution process computationally bound, particularly for applications such as contingency analysis of large networks. These difficulties in solving the AC power flow problem have led to extensive numerical studies and various simplified methods have been proposed and used. The simplified methods involve making a series of approximations to the non-linear powerflow problem (3.20). More the approximations made, the easier it is to find a solution. However, note that the AC power flow methods such as Newton-Raphson and all the approximate methods attempt to solve the same underlying power system. In this report, we review two methods that are commonly found in literature: decoupled load flow and DC approximation. In situations where a full power flow model is an absolute necessity, the solutions of these simplified methods are used as initial values, essentially when it is quite certain that the flat start approximation doesn't converge.

4.2 Fast Decoupled Load Flow

The Fast Decoupled Load Flow (FDLF) method is a simple and fast power flow solution technique which is derived from Newton's method. We briefly describe the basic version of the formulation here and refer to [3, 11] for further details.

The polar power-mismatch version of the Newton-Raphson method described in section 4.1.1 is used as a starting point to derive the decoupled load flow formulation. The linear system that is solved in every iteration of the NR polar power-mismatch version is given by (4.1) as,

$$-\left[\begin{array}{c|c} J^{11} & J^{12} \\ \hline J^{21} & J^{22} \end{array}\right]\left[\begin{array}{c} \Delta\delta \\ \Delta|V| \end{array}\right] = \left[\begin{array}{c} \Delta P \\ \Delta Q \end{array}\right] \tag{4.10}$$

The decoupling principle involves making the following assumptions which generally hold for power systems under normal operating conditions [3].

1. Voltage angle differences between buses are small.

$$\cos\delta_{ik} \approx 1; \quad \sin\delta_{ik} \approx \delta_{ik}$$

2. Transmission line susceptances are much larger than conductances.

$$G_{ik}\sin\delta_{ik} \ll B_{ik}$$

3. The reactive power injected into a node is much smaller than the reactive flow that would result if all lines connected to that bus were short circuited to reference [3].

$$Q_i \ll B_{ii}|V_i|^2$$

Evaluating the partial derivatives $J^{12} = \frac{\partial \Delta P}{\partial |V|}$ and $J^{21} = \frac{\partial \Delta Q}{\partial \delta}$ (see section 4.1.1 for equations) with the first two assumptions shows that the Jacobian sub-matrices J^{12} and J^{21} are small and can be neglected. Additionally, the terms $[J^{22}_{ik}]$ are multiplied with voltage magnitude $|V_i|$, the convenience of which will be clear in the derivation below. The following decoupled system of equations is then obtained.

$$-\left[\, J^{11} \,\right]\left[\, \Delta\delta \,\right] = \left[\, \Delta P \,\right] \tag{4.11a}$$

$$-\left[\, \widetilde{J}^{22} \,\right]\left[\, \Delta|\widetilde{V}| \,\right] = \left[\, \Delta Q \,\right] \tag{4.11b}$$

where, $J^{11}_{ik} = \frac{\partial \Delta P_i}{\partial \delta_k}$, $\widetilde{J}^{22}_{ik} = |V_i|\frac{\partial \Delta Q_i}{\partial |V_k|}$ and $\Delta|\widetilde{V}_i| = \frac{\Delta |V_i|}{|V_i|}$. The Jacobian sub-matrices J^{11} and $\widetilde{J}^{22}$ are computed using the partial derivatives defined in section 4.1.1 with the three assumptions stated above as follows.

$$\begin{aligned} \frac{\partial \Delta P_i}{\partial \delta_k} = |V_i|\frac{\partial \Delta Q_i}{\partial |V_k|} &= -|V_i||V_k|\left(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}\right) && (i \neq k) \\ &\approx |V_i||V_k|B_{ik} \end{aligned}$$

$$\begin{aligned} \frac{\partial \Delta P_i}{\partial \delta_i} &= \sum_{i \neq k} |V_i||V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) && (i = k) \\ &= B_{ii}|V_i|^2 + \sum_{k=1}^{N} |V_i||V_k|\left(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}\right) \\ &= B_{ii}|V_i|^2 + Q_i \\ &\approx B_{ii}|V_i|^2 \end{aligned}$$

$$\begin{aligned} |V_i|\frac{\partial \Delta Q_i}{\partial |V_i|} &= 2B_{ii}|V_i|^2 - \sum_{i \neq k} |V_i||V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) && (i = k) \\ &= B_{ii}|V_i|^2 - Q_i \\ &\approx B_{ii}|V_i|^2 \end{aligned}$$

The decoupled system (4.11) is now written as,

$$-\left[\ |V|^T B |V|\ \right]\left[\ \Delta\delta\ \right] = \left[\ \Delta P\ \right]$$

$$-\left[\ |V|^T B |V|\ \right]\left[\ \Delta\widetilde{|V|}\ \right] = \left[\ \Delta Q\ \right]$$

where $B = [B_{ik}] \in \mathbb{R}^{N\times N}$ is the matrix of line susceptances. Finally, the terms $|V_i|^T$ are taken to the right hand side and the terms $|V_i|$ are set to $1\,\mathrm{pu}$. A distinction is made at this stage for matrix B in the two linear systems for convenience.

With all the above modifications, the final decoupled power flow equations become,

$$-\left[\ B'\ \right]\left[\ \Delta\delta\ \right] = \left[\ \Delta\widetilde{P}\ \right] \tag{4.12a}$$

$$-\left[\ B''\ \right]\left[\ \Delta|V|\ \right] = \left[\ \Delta\widetilde{Q}\ \right] \tag{4.12b}$$

where $\Delta\widetilde{P}_i = \frac{\Delta P_i}{|V_i|}$ and $\Delta\widetilde{Q}_i = \frac{\Delta Q_i}{|V_i|}$. The decoupled system (4.12) is modified to represent a PV bus, similar to NR polar power-mismatch version as described in section 4.1.1. The order of the systems (4.12) will then be $(N-1)$ and $(N-N_g-1)$ respectively.

The matrices B' and B'' depend only on network parameters and are constant in every iteration. This means that the matrices have to be calculated and factorized only once, leading to faster computations of the power flow problem even though the number of iterations needed for convergence could be higher because of the approximations made. The steps involved in the fast decoupled load flow method are given in the following algorithm.

Algorithm 2: Decoupled load flow

```
$k := 0$
Initialize: $\delta^0$ and $|V|^0$
while not converged do
    Compute: $\Delta\widetilde{P}$
    Solve for the correction $\Delta\delta$: $-[B'][\Delta\delta] = [\Delta\widetilde{P}]$
    Update the approximation: $\delta^{k+1} = \delta^k + \Delta\delta^k$
    Use $\delta^{k+1}$ to compute $\Delta\widetilde{Q}$
    Solve for the correction $\Delta|V|$: $-[B''][\Delta|V|] = [\Delta\widetilde{Q}]$
    Update the approximation: $|V|^{k+1} = |V|^k + \Delta|V|^k$
    $k = k + 1$
end
```

4.3 DC approximation

DC approximation or DC load flow is a linear approximation of the power flow problem. The extent to which the non-linear power flow equations (3.20) are approximated and the kinds of assumptions made may vary based on the problem or the application. In fact, 'DC' refers to the collection of approximations made such that the network is *decoupled*. In some cases, the FDLF method is also considered as a DC approximation method. However, there is a fundamental difference between FDLF and DC approximation methods. In FDLF, the non-linear system is solved in each iteration and the approximation is made only to the Jacobian. In DC load flow methods, the non-linear equations are linearized to speed up computation, which considerably affects the accuracy of the final solution.

The DC load flow method described in [3] is derived as follows. The following approximations are made to the power flow problem (3.20).

- Bus voltage magnitudes are set to 1 pu in active power equations (3.20a).
- Voltage magnitudes are approximated as: $|V| = 1 + \Delta|V|$ and $\frac{1}{1+\Delta|V|} = 1 - \Delta|V|$ in reactive power equations (3.20b).
- Conductances of transmission lines are neglected: $G_{ik} \ll B_{ik}$.
- Voltage angles are small: $\cos\delta_{ik} \approx 1$ and $\sin\delta_{ik} \approx \delta_{ik}$.

Under these assumptions, the linearized version of the power flow problem (3.20) is given by,

$$P_i = \sum_{k \neq i} B_{ik}\delta_{ik} \tag{4.13a}$$

$$Q_i + B_{ii} = (Q_i - B_{ii})\Delta|V_i| + \sum_{k \neq i}(1 + \Delta|V_k|)B_{ik} \tag{4.13b}$$

where $\Delta|V|$ represents the deviation of the voltage magnitude from $1\,\mathrm{pu}$ voltage level.

The DC approach provides a fairly good approximation of voltage magnitudes and angles which can be used as initial values in NR or FDLF methods. It is also claimed in [12] that while the DC approximation leads to some loss of accuracy, the results match fairly closely with full power flow solution. The following approach of DC approximation is presented in [12] as the most dramatic of DC approximation methods.

The assumptions made to the non-linear power flow problem are:

- the reactive power balance equations are ignored,
- voltage magnitudes are set to 1 pu,
- line losses are ignored.

These assumptions reduce the non-linear power flow problem to the system of linear equations:

$$[B'] [\delta] = [P] \tag{4.14}$$

where $[B']$ is the line susceptance matrix, $[\delta]$ is the vector of bus voltage angles and $[P]$ is the active power injection vector.

It should be noted that the lack of losses in the DC solution can be reasonably compensated for by increasing the total load by the amount of estimated losses. The DC approach (4.14) has the following significant advantages over the Newton-Raphson method.

1. The linear system is half the size of the full problem since only the active power mismatch equations are solved.
2. The algorithm is not iterative, requiring just one single solution of (4.14).
3. The matrix B' is dependent only on network parameters and needs to be factorized only once.

We refer to [12] for further details, where a comparison between AC and DC methods is also made.

4.4 Gauss-Seidel

Gauss-Seidel method is another iterative technique that can be used to solve the non-linear power flow problem. The iterative scheme is derived from the complex power equation (3.19) with complex voltage V_i as the iteration variable.

$$\begin{aligned}
S_i &= V_i(YV)_i^* \\
&= V_i \left(\sum_{k=1}^{N} Y_{ik} V_k \right)^* \\
&= V_i \left(\sum_{k \neq i} Y_{ik} V_k \right)^* + V_i Y_{ii}^* V_i^* && \Longleftrightarrow \\
V_i Y_{ii}^* V_i^* &= S_i - V_i \left(\sum_{k \neq i} Y_{ik} V_k \right)^* && \Longleftrightarrow \\
V_i^* &= \frac{1}{Y_{ii}^*} \left(\frac{S_i}{V_i} - \sum_{k \neq i} Y_{ik}^* V_k^* \right) && \Longleftrightarrow \\
V_i &= \frac{1}{Y_{ii}} \left(\frac{S_i^*}{V_i^*} - \sum_{k \neq i} Y_{ik} V_k \right) \\
&= \frac{1}{Y_{ii}} \left(\frac{P_i - jQ_i}{V_i^*} - \sum_{k \neq i} Y_{ik} V_k \right) && (4.15)
\end{aligned}$$

The fixed point equation (4.15) leads to the following iterative formula.

$$V_i^{h+1} = \frac{1}{Y_{ii}} \left(\frac{P_i - jQ_i}{V_i^{h*}} - \sum_{k \neq i} Y_{ik} V_k^h \right), \quad h = 0, 1, 2, \ldots \tag{4.16}$$

where V_i^0 is a given initial approximation for each bus i. Equation (4.16) is evaluated at each iteration until convergence is met. If the approximations V_i^h are computed at once for all buses and applied at once in the next iteration, the iterative procedure is called *Jacobi* iteration. If the approximations are computed for one bus at a time and immediately used in the same iteration, the procedure is referred to as *Gauss-Seidel* iteration.

For a PQ bus, it is straightforward to apply the iterative formula (4.16), whereas for a PV bus, modifications are required, similar to Newton-Raphson method.

The Gauss-Seidel method is flexible and relatively easy to implement but it is generally slow and inefficient for large systems. The iterative scheme has a tendency to use a lot of computations, particularly in large scale problems, and could also converge to incorrect solutions. Hence, the Gauss-Seidel method is not preferred in practice. However, despite the shortcomings, it can be used to get a good perspective on power flow problems [13].

4.5 Summary

The following can be inferred from the power flow solvers that are described in the preceding sections (4.1 to 4.4).

- The **Newton-Raphson** method is widely regarded as the most commonly preferred power flow solver [8]. Several formulations of the NR method are found in literature. Among the six fundamental NR methods, the most widely used version is the power-mismatch formulation with polar coordinates [10]. The current-mismatch version with polar and Cartesian coordinates are found to give the best results for transmission networks [9].
- The **Gauss-Seidel** method is another full power flow solver that is used in power flow analysis. Despite the ease of implementation that the Gauss-Seidel method offers, it is seldom preferred in practice due to computational intensity [13].
- The **FDLF** method offers computational benefits against full power flow solvers by making approximations as described in section 4.2. The FDLF method is advantageous in terms of speed and simplicity compared to full power flow solvers [11]. However, the accuracy of the solution is not on par with full power flow solvers because of the approximations made.
- The **DC approximation** method is the simplest of approximate methods to solve the power flow problem. It has been concluded in [12] that the results of the DC approximation method match fairly closely with full power flow solvers despite the loss in accuracy. However, the DC approach considers only active power equations and hence can only substitute a full power flow solver for applications that do not require reactive power to be taken into account.

We would like to emphasize that the selection of a power flow solver is largely problem specific. For instance, while the DC approximation method is well suited for problems that are not sensitive to the approximations made, it cannot be used for applications that require calculation of reactive power in the network. Full power flow solvers such as the Newton-Raphson method and its formulations are always necessary for AC power flow analysis. Hence, for reactive power compensation assessment, AC power flow simulations are absolutely necessary.

Chapter 5

Software Packages

Given the importance of digital solution methods to power flow analysis, there are many commercial software packages available. These software packages are quite powerful and they are extensively used in the industry for power system operation, control and planning among many other applications. In this chapter we explore PowerFactory, PSS®E and pandapower and understand their capability to solve the power flow problem.

5.1 PowerFactory

PowerFactory is an engineering tool for the analysis of transmission, distribution and industrial electrical power systems. It is an integrated and interactive software package dedicated to electrical power systems in order to achieve the main objectives of planning and operation. PowerFactory has a single-line graphical interface which includes drawing functions, editing capabilities, and static and dynamic calculation features. The simulation functions offered by PowerFactory include power flow analysis, contingency analysis, optimal power flow among many other functions. PowerFactory is licensed by DIgSILENT GmbH.

PowerFactory offers both AC and DC power flow methods. In AC power flow method, the user can select one of the following formulations for solving the power flow problem.

1. Newton-Raphson power mismatch
2. Newton-Raphson current mismatch

PowerFactory allows the calculation of both balanced and unbalanced power flows. It is claimed in [14] that the Newton-Raphson power mismatch version converges best for large transmission systems, especially when heavily loaded and the Newton-Raphson current mismatch version converges best for unbalanced distribution systems.

In DC power flow method, only active power flow without losses is considered as explained in section 4.3 with the linear system (4.14).

PowerFactory has options for integration with Python scripts. Python scripts are generally used with PowerFactory for automation of tasks, creation of user defined calculation commands and integration of PowerFactory into other applications. PowerFactory also supports interfaces for softwares such as PSS®E and MATLAB. For further details, we refer to [14].

5.2 PSS®E

PSS®E is a power system simulation and analysis tool for power transmission operation and planning. Similar to PowerFactory, it offers a wide range of simulation functions. For AC power flow simulations, PSS®E has the following algorithms along with a few other modified methods available. PSS®E is licensed by Siemens.

1. Gauss-Seidel
2. Newton-Raphson
3. Decoupled Newton-Raphson

Since the convergence properties of solvers depend on the grid model, the following procedure is suggested in [15] to solve the power flow problem, particularly in situations where the characteristics of new power flow problems are not known.

1. Use flat start values as initial values.
2. Execute Gauss-Seidel method until the corrections made to voltages and angles decrease to, for instance, 0.01 or 0.005 pu in consecutive iterations.
3. Switch to Newton-Raphson method and execute it until the problem converges or until there are signs of divergence.
4. If Newton-Raphson method does not converge within 8 to 10 iterations, switch back to Gauss-Seidel.

PSS®E also has an embedded Python interpreter which can be used to access and run models in PSS®E from Python scripts.

5.3 pandapower

pandapower is an open source tool for power system modelling, analysis and optimization. pandapower combines the data analysis library pandas and the power flow solver PYPOWER. PYPOWER is a port of MATPOWER to the Python programming language. pandapower offers the following power flow methods and a few other modified ones.

1. Newton-Raphson
2. Newton-Rapshon with optimal multipliers
3. Gauss-Seidel
4. Fast decoupled load flow
5. DC power flow

The default solver uses the power mismatch formulation with polar coordinates of the Newton-Raphson method (see section 4.1.1). It is mentioned in [16] that the Gauss-Seidel method is included only for academic interests as it has many disadvantages compared to the Newton's method.

By default, the AC power flow solvers in pandapower solve the power flow problem without considering voltage limits, line current limits and generator limits. Currently, none of the solvers include options for automatic updating of transformer taps and for satisfying constraints such as voltage limits [16]. However, there is an option to keep the generator reactive power within limits, but at the expense of voltage setpoints. That is, when the generator reactive power is kept within limits, the voltages could go beyond the safe operating range which is generally set to 0.9 to $1.1\,\mathrm{pu}$. This is based on a brute force technique which adds an outer loop around the AC power flow solver. If the reactive power limit of a generator is violated, the reactive power injection by that generator is force fixed at the limit and the power flow is solved again. This procedure is repeated until there are no more violations. pandapower also has many test grid models to evaluate power flow algorithms.

In this project, we use PowerFactory and pandapower for our simulations. As the real-world grid models that we use in this project are built in PowerFactory which is a commercial software that does not allow access to its solvers, we develop a software interface that converts grid models from PowerFactory to pandapower. This enables us to apply our mathematical methods to all the real-world grid models that are used on a regular basis in TenneT.

Chapter 6

The Interface

An electricity grid model is a dataset that consists of information about the topology of the grid and about generation and consumption of electric power, in a computer readable format. A transmission system operator like TenneT uses grid models for several purposes such as operation and planning. Hence, grid models are developed and managed by TSOs on a regular basis. Moreover, a transparent exchange of grid models between the TSOs is of utmost importance considering the interconnected nature of electricity networks. For instance, there are 42 TSOs in Europe that collectively represent 35 countries and are responsible for safe operation of the European electricity system, which is the largest interconnected electricity network in the world. The European Network of Transmission System Operators for Electricity (ENTSO-E) is the association that facilitates the cooperation between European TSOs. ENTSO-E is responsible for aggregating grid models and distributing them to all the TSOs. The frequency of data exchange depends on the application. For example, grid models are exchanged on an hourly basis for the purpose of transmission system operation whereas on a yearly basis for transmission system planning.

For a long time now, PowerFactory has been the software of choice for the European TSOs and ENTSO-E for creation, development, management and exchange of grid models. PowerFactory's graphical user interface makes grid model development quite convenient. PowerFactory also has a version control system which is useful for managing grid models and also for creating investment plans for the future. Given the massive size and extent of the European electricity system, grid model exchange is a complex process in itself. The grid models are exchanged based on a standard called Common Information Model (CIM) which was developed by the electric power industry to allow efficient exchange of information regarding electricity networks. Among many other formats, PowerFactory also supports CIM. And of course, as discussed in the previous chapter, PowerFactory is used for power flow simulations, contingency analysis and several applications in TenneT.

However, from the perspective of research, PowerFactory is still a black box. Like most commercial softwares, calculations in PowerFactory are done behind the screen and the user has access only to the results. This is a hindrance, especially for power flow simulations where the freedom to examine the power flow problem and its solution in detail makes a big difference. For example, in case there is no power flow convergence, analyzing the power mismatches in each bus of the network gives a lot of insight into the problem and could possibly lead us to a solution. At the moment, it is unfortunately not possible to access the power flow solvers in PowerFactory. Hence, either there is a solution or there is not. Even if a small mismatch at a single bus is causing the power flow problem to diverge, the user would have to blindly accept that there is no solution. This leads to an uncertain attempt of trial and error based on heuristics to solve a diverging power flow problem. These limitations of PowerFactory formed a daunting barrier in the way of achieving the objectives of this project.

Luckily, PowerFactory has an Application Programming Interface (API)[1] that can be used by other applications to externally access grid model data and functionality of PowerFactory. We use the API to develop a software interface in Python that translates grid models from PowerFactory to pandapower. The interface is an API in itself, that is built as a user friendly software package that can be used in TenneT not only for power flow simulations but also for several other applications such as grid topology reconfiguration[2] and calculation of Power Transfer Distribution Factors (PTDFs)[3] and Line Outage Distribution Factors (LODFs)[4]. We thus have the best of both worlds: the eminent strengths of PowerFactory and the transparency of pandapower. We hope that the interface helps to bridge the gap between research and practice in the years to come.

6.1 Grid Models

The amount and accuracy of information that an electricity grid model contains depend on its usage. The reactances of lines and active power injections in the network suffice for DC power flow simulations whereas AC power flow simulations also require line resistances and reactive power injections. Dynamic power flow, optimal power flow and short circuit analysis would need even more details about the grid. In this project, for the development of the interface and for simulations, we focus on grid model details

[1]An API is a software interface that connects computer programs.

[2]Grid topology reconfiguration is performed, for example using machine learning, to determine the best grid topology for safe operation of the electricity system [17].

[3]PTDFs indicate the relative change in active power that occurs on transmission lines due to power transfers between two regions which could be countries, areas or zones [18].

[4]LODFs describe the changes in active power flow in the network when a particular line in the network fails [18].

that are sufficient to perform steady state AC power flow simulations. The elements of a grid model can be categorized into the elements that form the topology of the grid and the ones that correspond to load and generation.

Grid topology

As discussed in chapter 3, the transmission network is modelled as a directed graph of buses and lines. Each bus is defined by a unique index and its voltage level. Some real-world grid models also contain geographic coordinates of buses. A line is defined by its resistance and reactance.

Switches are devices that are used to close or open electrical circuits. An ideal switch has zero impedance. In the transmission network model, two buses connected by an ideal switch are merged together. That is, the two buses are replaced by a single bus. Even though in reality a switch would have a small impedance, it is assumed to be zero in steady state power flow simulations. This has the advantage that the number of buses in the network that are considered for calculations reduces and the power flow convergence problems that could arise due to small impedances are avoided. A real world transmission network contains a large number of switches. For example, the Dutch transmission network has about 15517 switches which is more than the 14685 buses that it has. Merging the switches results in a simplified network with just 1700 buses. This means that the size of the Jacobian becomes significantly smaller, leading to computational benefits. If the impedance of a switch is large enough to be considered necessary for calculations, it is modelled as a line.

The voltages in the network are shifted up and down using transformers. In our models, we have two-winding and three-winding transformers. As the names suggest, a two-winding transformer creates two different voltage levels in the network and a three-winding transformer creates three different voltage levels. For details about modelling various types of transformers see [14] and [19].

Shunts are devices that exchange reactive power with the network. A shunt inductor consumes reactive power whereas a shunt capacitor injects reactive power into the network. As discussed in chapter 3, a shunt is modelled as a reactance.

All the above elements which constitute the topology of the grid are built into the admittance matrix. That is, the admittance matrix of the network contains the necessary information about the elements that form the topology (structure) of the electric grid.

Load and generation

Loads and generators are elements that exchange electric power with the network. Loads consume electric power and are generally aggregated, in the sense that several loads are combined into a single load and connected to a single bus. Generators inject electric power into the network. A generator that does not control its voltage is called a static generator and is modelled similar to a load. For this reason, active power and reactive power are both defined for a static generator. A voltage-controlled generator, hereafter referred to as generator, adjusts its reactive power output to maintain the defined voltage. Hence, for a generator, active power and the voltage setpoint are defined whereas reactive power is calculated during power flow simulations. Loads and generators are collectively called power injectors. If power is consumed, as by a load, the power injection is negative. The distinction between loads, generators and static generators is defined using the concepts of PQ buses and PV buses as explained in chapter 3. In PowerFactory, a generator is called a synchronous machine.

By default, the slack bus is modelled as an external grid. In principle, an external grid is another network, such as the network of a neighboring country, that is modelled as a power source or sink at a single bus. As we know from chapter 3, the voltage magnitude and the voltage angle are defined for a slack bus. In fact, a generator can also be a slack bus, in which case the voltage angle is also predefined along with voltage magnitude.

6.2 The Interface

The interface is written in Python and it uses the PowerFactory API as the channel of communication. PowerFactory is written in C++ and so is its API. To facilitate communication with Python, the PowerFactory API consists of a module[5] that can be dynamically imported into Python. That is, the module and its contents can be imported into Python runtime, without having to decide in advance the imports that are needed by the program. Hence, a dynamic import of the Python module means that the software PowerFactory can be accessed from Python runtime. This possibility serves as the basis for the interface.

As the interface is part of a larger initiative of software migration towards Python in TenneT and not meant only for this project, we built it as an importable Python package that also comes with a few useful functions such as validation and diagnostics along with grid model conversion.

[5]A Python module is a file that contains Python definitions and statements.

6.2.1 Grid model conversion

In PowerFactory, a grid model is made of objects[6]. Each element of a grid model is an object and an object contains more objects. For example, the object that represents a line contains an object each for the two buses that the line connects. Hence, the dynamically imported Python module is like a sack of objects that collectively represent a grid model. The characteristics of the elements such as the resistance of a line are read from the corresponding object using attributes[7].

In pandapower, a grid model is a collection of pandas dataframes[8]. The collection of pandas dataframes is built as a pandapower data structure, which is used for creating grid models and doing calculations. The results are stored in the same pandapower data structure as well. This makes handling a grid model very convenient in pandapower.

The interface picks the objects from the Python module one by one, just like peeling the layers of an onion, and builds a pandapower data structure. This is done by looping over all the elements of a PowerFactory grid model and reading their characteristics in each iteration using the corresponding attributes. The elements and their characteristics are then built into a grid model in pandapower.

In fact, the interface is a map between two data structures that represent the same grid model in two different ways. As we know, a grid model is fundamentally a graph of buses and lines. All other elements are connected to buses. We represent the graph by $G = (N, E)$, where N is a set of nodes (buses) and E is a set of edges (lines). To connect the two data structures, we need a function that maps the buses in PowerFactory to the buses in pandapower. That is, we need a function that maps the nodes of the graph of the PowerFactory grid model to the nodes of the graph of the pandapower grid model. This function forms the core of the interface. We denote the function by $f : N_{pf} \rightarrow N_{pp}$, where $N_{pf} = \{v_1, v_2, v_3, \ldots, v_n\}$ is a set of unique objects that represent the buses of the PowerFactory grid model and $N_{pp} = \{1, 2, 3, \ldots, n\}$ is a set of unique indices that represent the buses of the pandapower grid model.

The function f is used throughout the interface. For example, to create lines in the pandapower grid model, the interface loops over all the objects that represent lines. In each iteration, the two buses that the line connects are extracted from the object using the corresponding attributes. The two buses are objects themselves. The function f is then used to identify the indices of the two buses in pandapower. The indices and the char-

[6]In Object Oriented Programming (OOP), an object is a bundle of data structures and functions.

[7]An attribute is a specification that defines the property of an object.

[8]pandas is a data analysis library written in Python. A pandas dataframe is a two-dimensional data structure in tabular format, similar to a spreadsheet.

acteristics of the line such as resistance and reactance are then used to create the line in pandapower. All other elements are similarly created.

In our grid models, we have the elements that are listed below. The attributes shown in brackets fetch a list of objects that represent the elements, from the Python module. For example, ’elmlne’ is the attribute used to extract all the lines from the PowerFactory grid model through the Python module.

- Buses (elmterm)
- Switches (elmcoup)
- Lines (elmlne)
- Two winding transformers (elmtr2)
- Three winding transformers (elmtr3)
- Loads (elmlod)
- Static generators (elmgenstat)
- Generators (elmsym)
- Shunts (elmshnt)
- External grids (elmxnet)

The characteristics of the elements are similarly read from their objects using attributes. For example, ’R1’ is the attribute used to read the resistance of a line, ’X1’ is the one used to read the reactance and so on. Not all attributes are mentioned in the user manuals of PowerFactory and they can only be found in the PowerFactory application itself, distributed across the software. Which means, one of the challenges in developing an interface like this is finding the right attributes for all the elements. Hence, this interface is also an extensive library of objects and attributes of a PowerFactory grid model. This saves a significant amount of time for further developing Python scripts to communicate with PowerFactory for other applications, without having to search every corner of the software for attributes.

In PowerFactory, the nomenclature for buses is that all the buses are called *terminals* and some of them are regarded as *busbars* depending upon their usage. That is, a terminal is called a busbar if it is connected to several other terminals and elements. Figure 6.1 shows the screenshot of a substation (within the dotted rectangle) in a PowerFactory grid model. The buses A and B (long red horizontal lines) are called busbars. All the red dots correspond to terminals. This arrangement of two busbars is called as double-linked bus chain. As can be seen in the figure, all the terminals are connected via closed switches. This means, all the terminals can be merged and the entire substation can be modelled as a single bus in pandapower. As discussed in section 6.1, this considerably simplifies grid models. In this project, we develop an algorithm to merge terminals that are connected by closed switches into one single bus.

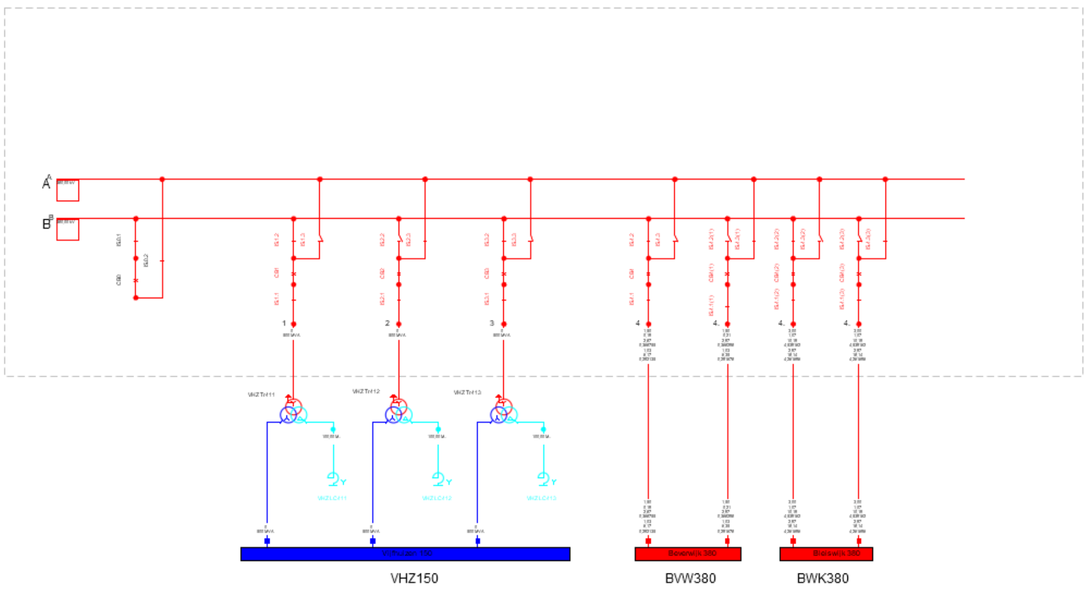


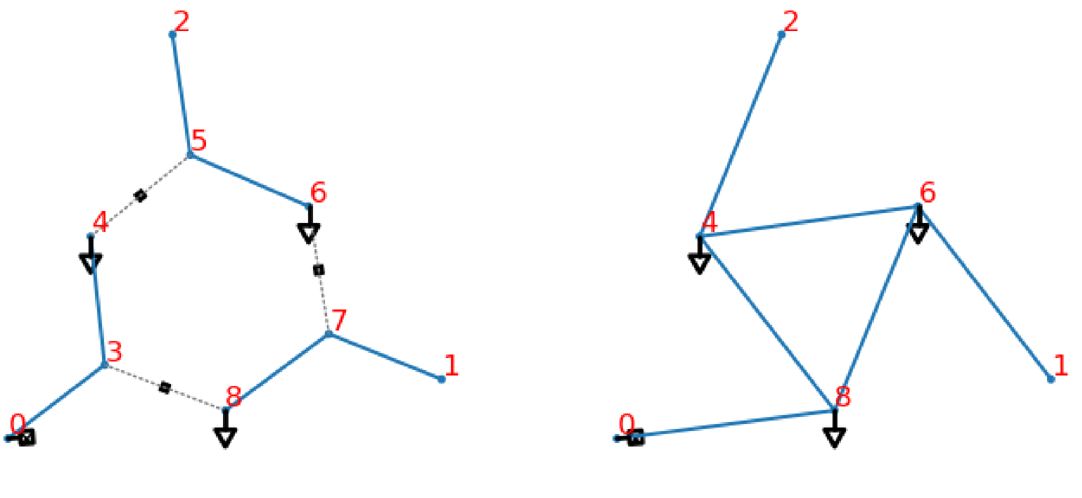
Figure 6.1: A substation in a PowerFactory grid model

Merging buses

If the impedance of a switch is assumed to be zero, the buses connected by the switch cannot independently exist in the mathematical model. If they do, as the impedance between them is *assumed* to be zero, the two buses are consequently disconnected in the model but they are connected in the physical grid. Hence, either the switch should be modelled as a line with an impedance or the two buses should be merged into a single bus. Algorithm 3 shows the steps involved in merging the buses connected by closed zero-impedance switches.

Algorithm 3: Merging buses

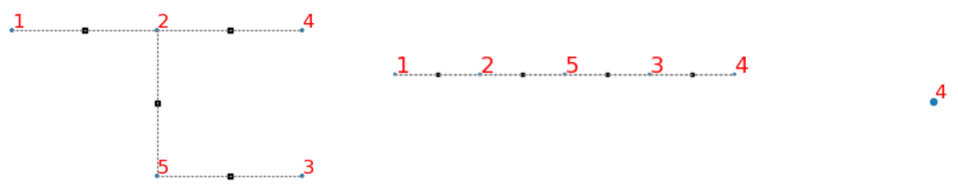
```
Initialize:
$N_{pp} = \{1, 2, 3, \ldots, n\}$
**for** *each switch* **do**
    $p_i$ = parent of bus $i$
    $p_j$ = parent of bus $j$
    $N_{pp}[p_i] = p_j$
**end**
**for** *each bus* **do**
    $k$ = index of last bus in the chain
    $N_{pp}[i] = k$
**end**
```



(a) With switches (b) With buses fused

Figure 6.2: Merging buses

Figure 6.2 shows a small network and its buses merged. The working principle of the algorithm is as follows. By default, a unique index is assigned to each bus by defining the set $N_{pp} = \{1, 2, 3, \ldots, n\}$. The algorithm loops over the switches first, and then over buses. In the first loop, the elements of the set N_{pp} are modified in such a way that if a bus j is to be merged with bus i, the number j is placed at index i in the set N_{pp}. For example, to merge bus 1 with bus 2, 2 is placed at index 1 of N_{pp}. That is, $N_{pp}[1] = 2$. This implies that bus 1 is mapped to bus 2 $(i \rightarrow j)$. For single switches, the buses i and j could just be the buses of a switch (i, j). However, there could be a bunch of switches connected together as shown in figure 6.3a. In such cases, a few more steps are necessary to merge the buses together.



(a) Initialize (b) Step 1 (c) Step 2

Figure 6.3: Steps of the merging algorithm

The idea is to form a chain (figure 6.3b) for each bunch of switches in the network such that every bus leads to the end of the chain that it belongs to. For example, after the transition from 6.3a to 6.3b, the set N_{pp} will be $\{2, 5, 4, 4, 3\}$. This is obtained by merging the ends of the chains (called parents in algorithm 3) that bus i and bus j belong to, for a switch (i, j) in the first loop of the algorithm.

The second loop runs over all the buses and for each bus, finds the last index in the chain and assigns it to the bus. Now for the example, the set N_{pp} will be $\{4, 4, 4, 4, 4\}$ after the second loop. This means that all the buses $\{1, 2, 3, 4, 5\}$ are connected to bus 4 (see figure 6.3c). Note that any bus could be the last bus of the chain depending upon the order of switches read by the first loop. In the end, the set N_{pp} represents a map between it's elements and their indices in N_{pp}. The set N_{pp} is then used to create buses in pandapower. That is, a bunch of buses is modelled as a single bus in pandapower.

6.2.2 Validation

As developing an interface that can convert a grid model as big as the transmission grid of the Netherlands from a commercial software to an open source software is an intense task, we do it in small steps. We choose small grid models and few elements at first and then extend the interface to larger grid models. To validate the interface in every step, we develop a subroutine that can perform AC and DC power flow simulations in both PowerFactory and pandapower, and compare the results. The relative error in voltage magnitudes (ϵ_V) and in voltage angles (ϵ_δ) are calculated as,

$$\epsilon_V = \frac{||V_{pf} - V_{pp}||_2}{||V_{pp}||_2} \tag{6.1}$$

$$\epsilon_\delta = \frac{||\delta_{pf} - \delta_{pp}||_2}{||\delta_{pp}||_2} \tag{6.2}$$

where V_{pf} is the voltage magnitude vector calculated by PowerFactory and V_{pp} is the voltage magnitude vector calculated by pandapower. Similarly, δ_{pf} and δ_{pp} are voltage angles calculated in PowerFactory and pandapower respectively. To demonstrate the interface, we convert a small part of the 110 kV grid in the Overijssel[9] region and the Dutch grid.

The 110kV Overijssel grid

A small part of the 110 kV grid in Overijssel, which is part of the Dutch grid is converted from PowerFactory to pandapower. A PowerFactory schematic of the grid model is shown in figure 6.4. The PowerFactory grid model consists of 32 terminals, 7 loads

[9]Overijssel is a province in the east of the Netherlands.

and 1 generator (slack bus). Merging the terminals leads to 8 buses and the size of the Jacobian is $2N - N_g - 2 = 13$. We use the polar power mismatch version of the Newton-Raphson method (4.1.1) to perform AC power flow simulations in PowerFactory and pandapower. Surprisingly, the errors are exactly zero:

$$\epsilon_V = 0.0000$$
$$\epsilon_\delta = 0.0000$$

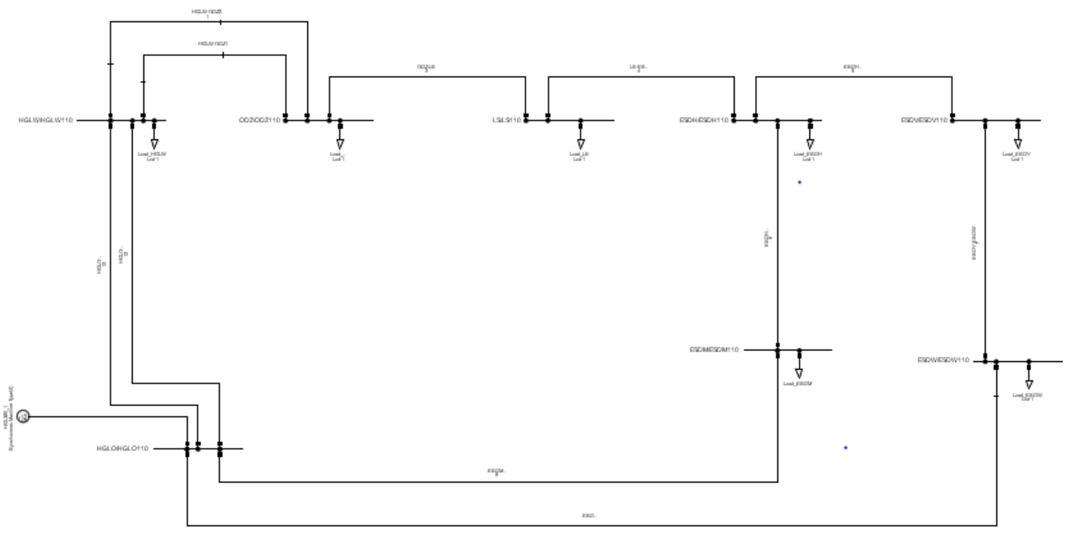


Figure 6.4: A small part of the Overijssel 110 kV grid model in PowerFactory

The Dutch grid

The high voltage transmission grid of the Netherlands consists of the following elements.

- Terminals: 14685
- Switches: 15517
- Lines: 1197
- Shunts: 160
- Two-winding transformers: 439
- Three-winding transformers: 216
- Loads: 334
- Generators: 4
- Static generators: 352
- External grids: 1

The voltage levels in the grid are 110 kV, 150 kV, 220 kV and 380 kV. Merging the terminals leads to 1700 buses and the size of the Jacobian is 2700. Doing AC power flow simulations using the polar power mismatch of the Newton-Raphson method gives the following errors.

$$\epsilon_V = 0.037$$
$$\epsilon_\delta = 0.042$$

Figure 6.5 shows a comparison between the voltage magnitudes and angles calculated in PowerFactory and pandapower. The errors are due to the following reasons.

1. Attributes: As discussed earlier in this chapter, finding the right attribute for a characteristic of an element is difficult in PowerFactory. Moreover, for certain elements, there could be two attributes for the same characteristic with slightly different functions. For example, the attribute for active power of a load is 'plini' which is defined by the user whereas PowerFactory calculates the active power of a load during power flow simulations which could be different from the active power defined by the user. The attribute for the calculated value is 'plini_a', where 'a' stands for 'actual values'. These nuances are imperceptible and can only be understood by working closely with PowerFactory for a long time.

2. PowerFactory API: It has been found that at a few buses in the grid, the calculated values are not communicated to Python by the PowerFactory API. However, it has been manually checked that these values are the same in both PowerFactory and pandapower. The exact reason for this has not been found yet.

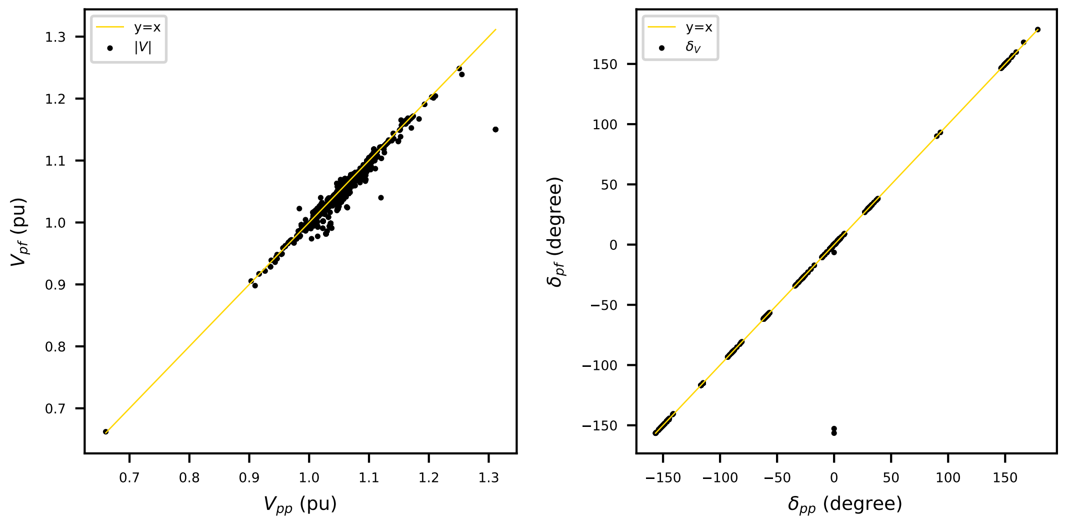


Figure 6.5: Voltage magnitudes and angles in PowerFactory and pandapower

The errors can be further reduced by fixing the above listed issues and we feel that it can be achieved over a period of time by validating the interface with more PowerFactory grid models. In any case, the errors are already quite less and we use the interface throughout this project to convert grid models from PowerFactory to pandapower.

Chapter 7

Power Flow Convergence

A fact well known in power flow analysis is that the power flow problem often fails to converge to a solution or rather diverges from the solution. This occurs especially during contingency analyses and system planning when the grid topology and the power injections frequently vary. In such cases, most power flow analysis softwares to this day give up saying that the power flow problem did not converge and provide very little insight about what went wrong. As the power flow problem could diverge due to a wide variety of reasons and given that the commercial power flow analysis softwares do not allow access to their solvers, it is often hard to identify the cause and find a solution. In practice, this problem is tackled by using trial and error techniques that are based on heuristics to make changes to the grid topology or to the power injections or to both. Even though this could lead to a solution in most cases, it is time consuming and the solution is not guaranteed to be optimal. During year-round simulations where the power injections change for every hour, it is certainly not feasible to manually solve convergence problems. This makes it impossible to perform AC year-round simulations.

In this project, we develop a rationale to understand and solve convergence problems using mathematical methods. We develop the automating algorithm that can automatically fix convergence problems and perform year-round simulations with little or no manual assistance. In the remaining parts of this thesis we design the mathematical framework, develop the automating algorithm and demonstrate capabilities and limitations by applying them to a variety of grid models and test cases.

7.1 Classification and methods

To develop the automating algorithm, it is convenient to classify power flow problems on the basis of convergence properties. In this section, we describe each problem classification and the methods we use to solve them. Power flow problems can be mathematically classified on the basis of convergence properties as follows.

1. Well-posed problems
    (a) Well-conditioned problems
    (b) Ill-conditioned problems
2. Ill-posed problems

A mathematical problem is called *well-posed*[1] if the following conditions hold.

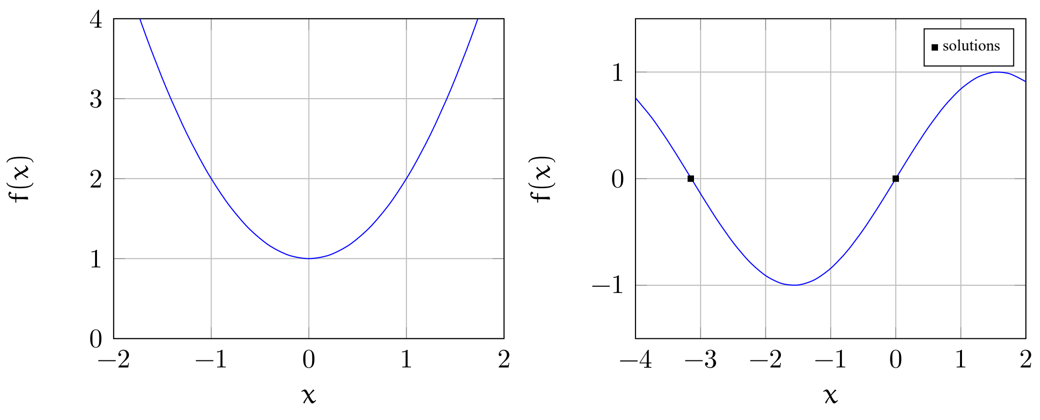
- Existence: There exists at least one solution.
- Uniqueness: There is at most one solution.
- Stability: The solution's behavior changes with the initial conditions.

A problem which is not well-posed is called *ill-posed*. Figure 7.1 shows two ill-posed root-finding problems of the form $f(x) = 0$, where $f(x)$ is a single variable function. In power flow analysis, the power flow problem is called ill-posed if any one of the following conditions is true.

- The system has no solution. For example, in fig. 7.1a, $f(x)$ never crosses the x - axis. Hence it has no solution.
- The system has multiple solutions. For example, in fig. 7.1b, $f(x)$ crosses the x - axis twice, leading to two solutions. For power flow problems, the practical consequence of this is that the solution may converge to a lower voltage value.

The above classification is very general. That is, the power flow problem could be ill-posed due to several reasons that are often hard to identify. For example, modelling errors such as a line in the network that has a disconnected end, line resistance and reactance values set to zero and so on. A list of such modelling errors could be endless and is hard for an automated algorithm to comprehend. Hence, in this thesis we restrict the scope of power flow convergence to power injections of the power flow problem. We will show that the algorithms developed also give insight into modelling errors that could be causing divergence, although the algorithms do not fix the errors automatically during year-round simulations.

[1]Widely known as Hadamard well-posedness, named after Jacques Hadamard, a 20th century mathematician who introduced the concept.



(a) No solution

(b) Multiple solutions

Figure 7.1: Ill-posedness

We define a space of input parameters which contains the following power injections.

- Active powers and reactive powers of loads and static generators.
- Reactive powers of shunts and active powers of generators.

For simplicity, the voltage setpoints and hence reactive powers of generators are not included in the parameter space. Based on convergence properties, the parameter space can be divided into three regions as shown in figure 7.2. The regions 1.a, 1.b and 2 correspond to the set of input parameters that make the power flow problem well-conditioned, ill-conditioned and ill-posed respectively. Region 1 (a and b) is called the solvable region and region 2 is called the unsolvable region. The boundary between the solvable region and the unsolvable region is denoted by σ.

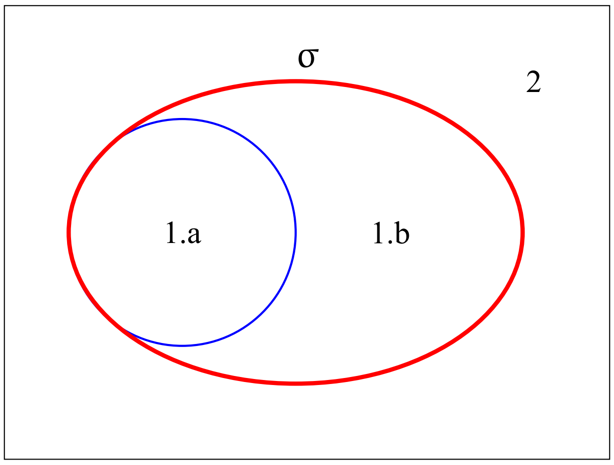


Figure 7.2: Regions in parameter space

During year-round simulations, the power flow problem could move back and forth across the boundary σ. In the following sections we describe each classification and the methods we use to solve the power flow problem in all the three regions.

7.1.1 Well-conditioned problems

A power flow problem is called well-conditioned if it can be solved using conventional power flow methods such as the Newton-Raphson method. If a well-conditioned power flow problem is solved using the Newton-Raphson method, the Jacobian (see section 4.1) is non-singular in every iteration. In this case, the number of iterations is generally small. In most cases, the flat start initial condition suffices to solve a well-conditioned power flow problem.

7.1.2 Ill-conditioned problems

A power flow problem is termed ill-conditioned if the Jacobian matrix is singular or nearly singular. Use of conventional methods to solve ill-conditioned power flow problems results in very slow convergence or divergence. That is, even though a solution exists, the conventional power flow solver cannot find it or convergence is very difficult. This is known as false divergence and could be due to a poor initial estimate. For example, fig. 7.3 shows the tendency to diverge when the initial value x_1 is far from the solution. It is clear that $x_3 > x_1$ and continuing the iteration further from x_3 causes the solution to diverge. Hence, more advanced methods are needed to solve ill-conditioned power flow problems. The most widely used method is the optimal multiplier method which is also called the Iwamoto multiplier method. This method was developed in 1981 by Iwamoto and Tamura [20]. The method is derived as follows.

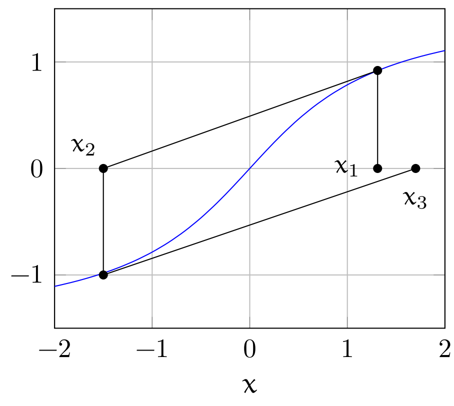


Figure 7.3: False divergence

Optimal multiplier method

The Newton-Raphson method has *local* quadratic convergence as mentioned in section 4.1. That is, the NR method converges quadratically when the initial value is close to the solution. A method that converges for any initial value is called *globally* convergent. The optimal multiplier method ensures global convergence and aims to solve the problems of false divergence of the NR method. The idea is that a scalar multiplier μ is introduced in the update step $x^{k+1} = x^k + \Delta x^k$ of the NR method (see algorithm 1) as follows.

$$x^{k+1} = x^k + \mu \Delta x^k$$

Determining the optimal multiplier μ^* is an unconstrained non-linear optimization problem which can be solved in several ways. In [20], μ^* is determined as follows.

In the Newton-Raphson method, the first order Taylor expansion of the mismatch function $F(x)$ is expressed as (see section 4.1),

$$F(x + \Delta x) \approx F(x) + J(x)\Delta x$$

where the higher order terms of the expansion are neglected. In [20], the higher order terms are collectively expressed as the mismatch function evaluated at the correction Δx. That is,

$$F(x + \Delta x) = F(x) + J(x)\Delta x + F(\Delta x)$$

Incorporating the multiplier μ,

$$F(x + \mu\Delta x) = F(x) + \mu J(x)\Delta x + \mu^2 F(\Delta x)$$

A cost function $C : \mathbb{R}^n \to \mathbb{R}$ which represents the proximity of the approximation to the solution defined as,

$$C(x) = \frac{1}{2} F(x + \mu\Delta x)^T F(x + \mu\Delta x) \tag{7.1}$$

is used to determine the optimal value of μ by evaluating the following expression.

$$\frac{dC}{d\mu} = 0 \tag{7.2}$$

Algorithm 4 describes the modified Newton-Raphson algorithm with the optimal multiplier incorporated.

Algorithm 4: NR algorithm with optimal multiplier

```
k := 0
Initialize: $x^0$
while not converged do
    Solve for the correction: $-J(x^j)\Delta x^j = F(x^j)$
    Compute $\mu^j$ by evaluating $\frac{dC}{d\mu} = 0$
    Set $\Delta x^j = \mu^j \Delta x^j$
    Update the approximation: $x^{j+1} = x^j + \Delta x^j$
    $j = j + 1$
end
```

Note that the cost function $C(x)$ is a fourth order function of μ and the expression (7.2) results in a third order polynomial in μ which can be solved using well known methods. Figure 7.4 shows how the optimal multiplier method solves the false divergence problem in one dimension.

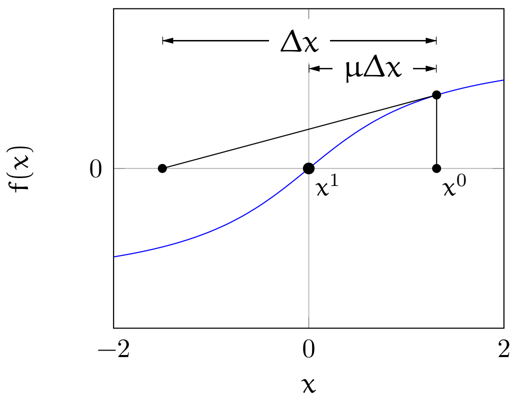


Figure 7.4: Optimal multiplier

Line search technique: The optimal multiplier μ can be determined using another popular method known as the line search method [21]. The line search is an iterative method to find a local minimum x^* of an objective function $f : \mathbb{R}^n \to \mathbb{R}$. In our case, in each iteration the following optimization problem is solved to determine μ^*.

$$\min_{\mu \in [0,1]} C(x^k + \mu \Delta x^k) \tag{7.3}$$

Since $C(x)$ represents the proximity of the approximation to the solution, the scalar multiplier μ must satisfy $C(x^k + \mu\Delta x^k) < C(x^k)$ at each iteration. Hence, the search is limited to $\mu \in [0, 1]$. The line search procedure is described as follows.

The procedure is similar to the binary search algorithm and starts with $[\mu_1, \mu_2] = [0, 1]$. The domain is then divided in half. That is, $[\mu_1, \mu_3, \mu_2] = [0, 0.5, 1]$. The function $C(x)$ is evaluated at the three points and the following conditions are checked.

$$C(\mu_1) > C(\mu_3)$$
$$C(\mu_2) > C(\mu_3)$$

- If the conditions hold true, the two subdomains $[0, 0.5]$ and $[0.5, 1]$ are further divided in half and the procedure is continued with $[\mu_4, \mu_3, \mu_5] = [0.25, 0.5, 0.75]$ and so on (see fig. 7.5a).
- If one of the conditions is violated, the corresponding subdomain is excluded and the procedure is continued. That is, if $C(\mu_2) < C(\mu_3)$, the three points will be $[\mu_3, \mu_4, \mu_2]$ where μ_4 is the point at the center of $[\mu_3, \mu_2]$ (see fig. 7.5b).
- The procedure terminates when the search domain converges to a point, which is the optimal multiplier μ^*.

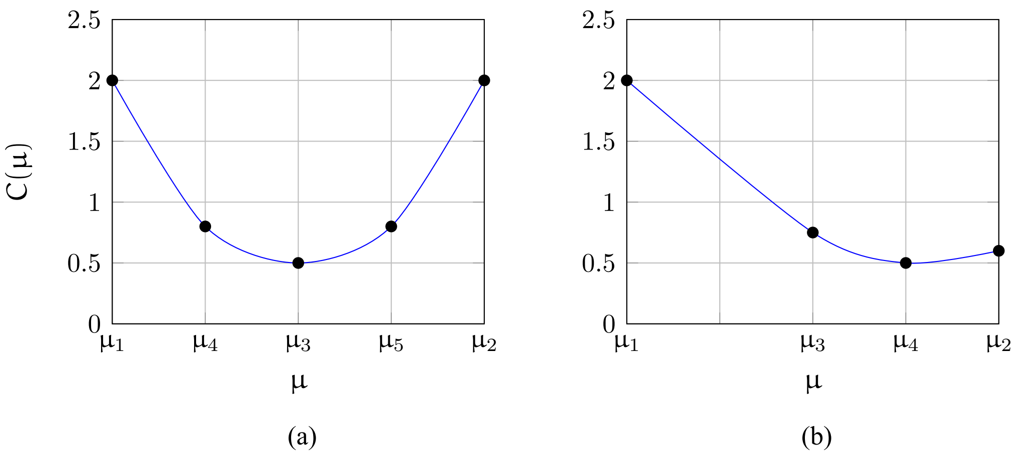


Figure 7.5: Line search: subdividing the search domains

Numerical experiments have demonstrated that the optimal multiplier method has good convergence properties even for very ill-conditioned problems [22].

7.1.3 Ill-posed problems

As discussed in the introduction of this section, an ill-posed power flow problem is harder to understand and solve. The first step involves finding the underlying reason that is making the problem unsolvable. The second step is to determine the optimal changes that can be made to the power flow problem to restore solvability. In mathematics, the technique of reformulating an ill-posed problem to be able to numerically solve it is called *regularization*.

In this thesis, we seek an explanation for ill-posed power flow problems and based on our findings, we develop methods to regularize them. It is to be noted however that the reasons we find and explain here are far from exhaustive and are intended to give the reader a broad insight into ill-posed power flow problems. We classify the errors that could possibly make a power flow problem ill-posed into two categories: modelling errors and errors in input parameters.

Modelling errors

As discussed in chapter 6, the size of a real-world electricity grid makes grid model development a complex task. Hence, grid models are often prone to modelling errors. In this project, we have come across the following modelling errors.

1. Errors in *characteristics* of grid elements such as line impedances, short-circuit voltages[2] and losses[3] of transformers. One of the possible reasons for these errors is that in PowerFactory, not all the default characteristics of a newly added element are right. For example, when a line is added to the grid model, the impedance of the line is zero by default. As we know, this is not realistic due the fact that zero impedance means infinite admittance. The right values are then supposed to be manually fed into the software by the modeller. This leads to human error as the grid models are large and the grid elements are detailed with many characteristics to be fed in. In case of wrong values, PowerFactory overwrites them with right ones to a certain extent. This is often not conveyed to the user, possibly leading to confusion. Hence, in this project, the interface is built in such a way that it also acts as a filter that identifies and tells the user about such anomalies in PowerFactory grid models before converting them to pandapower.

2. Errors in the *structure* of the transmission network such as disconnections. For example, grid elements with a disconnected end or a part of the network totally

[2]The short-circuit voltage of a transformer is the overall voltage decrease of the transformer during rated loading.

[3]In an electrical machine such as a transformer, loss is defined as the difference between input power and output power.

disconnected from the rest of the network. Again, this is likely due to human error which occurs because of the changes that are constantly made to the topology of the grid during transmission system planning. For instance, if a line breaks or if it is put off for maintenance, it is not deleted from the grid model but instead not considered for calculations. This is called putting an element *out of service*. If a line that connects two parts of the network alone is put out of service, the network is split into two parts. This is known as islanding. Power system analysis softwares such as PowerFactory and pandapower allow islanding in calculations by solving the parts of the network separately, provided a generator is present to power the island. In the absence of a generator, the island is disregarded for calculations. Even though this option of allowing islands in the network by power system analysis softwares is an essential feature for studying real-world grid model scenarios, it could lead to difficulties in power flow convergence since the power balance in the grid is affected. In PowerFactory, the islands are called isolated areas. Disconnections in the network also arise due to switching. For instance, if a switch that connects a line and a transformer is kept open, it leads to disconnected ends for both the line and the transformer. Switching is very often done to change the topology of the grid during operation and planning.

Another commonly observed difficulty is with the fictitious border nodes. In PowerFactory, a fictitious border node is a fictitious bus that is placed in between the networks of two neighboring countries. If the network of a neighboring country is disconnected, the fictitious buses remain in the grid model unless they are manually removed by the user. Even though this does not affect the calculations, it makes analyzing the results troublesome as there is no load flow in most of the fictitious buses. In the Dutch grid, there are 799 fictitious border nodes out of which only 13 are regarded for calculations. Moreover, the fictitious border nodes could also lead to isolated areas in the network as discussed above. In the Dutch grid model considered in this project, there are 45 isolated areas and in the sub-European grid model there are 485 isolated areas.

The larger the grid model, the harder it is to find these anomalies. Moreover, a list of such modelling errors is endless and it seems unrealistic to expect a software to tie the loose ends on its own. In fact, the changes that PowerFactory makes to the characteristics of grid elements already demonstrate a certain level of automation of fixing modelling errors. The corrections that can be made are limited and are unfortunately not transparent enough for the user to easily see their implications. Hence, it seems more reasonable to avoid modelling errors during the grid model development phase itself than to fix them later using an automated subroutine. Doing so greatly improves the quality of the grid models and ensures that the modelling errors are not propagated throughout a grid model's lifecycle. We thus recommend that the modelling phase be very stringent in checking that the modelling errors do not pass by unnoticed. All the errors discussed above were found during the development of the interface. We have tried to build these options in the interface and we hope that it helps to clean up modelling errors at least during the conversion of grid models to pandapower.

To automate power flow simulations in this project, we thus devote all our attention to the parameter space of the power flow problem.

Errors in parameters

As discussed in the introduction of this section, we define a space called the parameter space which consists of the power injections of the power flow problem. That is, the parameter space is constituted by the load and generation data. In TenneT, for year-round simulations, the load and generation data are generated by an excel sheet and for given total load and total generation values, a Python program distributes active and reactive powers among the loads and generators in the grid model. However, for the distributed load and generation, only DC power flow simulations, but not AC power flow simulations, are performed in PSS®E to ensure balance in active power. This means that the reactive power injections could be unrealistic for the grid model and the power flow problem could possibly diverge for a number of cases during year-round simulations. Physically, such a scenario leads to system instability or voltage collapse. That is, the voltages in the network decline and could cause a blackout. In this project, we develop methods that can be used to optimally change the power injections such that a power flow solution can be determined.

A literature review on power flow problems shows that over the past few decades, a substantial amount of effort has been put into developing methods to solve an ill-posed power flow problem by making changes to input parameters (power injections). It has been found in the literature that an ill-posed power flow problem can be reformulated in several ways depending upon the application. We highlight two formulations here that can be considered as the fundamental approaches to solving an ill-posed power flow problem. Most of the methods can be seen as extensions of the two formulations. For example, minimum load shedding and generator re-dispatch problems[4] can be solved by further imposing constraints to any of the two fundamental approaches, which are:

1. An approach that uses the left eigenvector associated with the zero eigenvalue of the Jacobian to steer the problem towards solvability.

2. An approach that involves formulating the problem as a non-linear optimization problem and solving it using Lagrange multipliers.

[4]Grid operators do minimum load shedding and generator re-dispatch to relax a stressed network and hence operate the grid within safety limits.

In this thesis, for the sake of simplicity and the possibility of easily incorporating the method into the Newton-Raphson solvers, we use the first approach to find a solution and then apply constraints to make the obtained solution more realistic to our application.

We use the methods presented in [22] and [23] as a foundation to construct our algorithms. The idea is to steer the problem from the unsolvable region (see figure 7.2) towards the solvable region by making optimal changes to input parameters. The goal is to find the solvable[5] operating point which is closest in the parameter space to the desired[6] operating point.

Recall from section 4.1 that the power flow problem is a system of non-linear equations given by,

$$F(x) = \Delta S(x) = S^{sp} - S(x) \tag{7.4}$$

where, S^{sp} is the specified complex power injection vector and $S(x)$ is the complex power vector computed by the Newton-Raphson solver. That is, $F(x)$ represents the mismatch between the specified values and calculated values of complex power at each bus of the system. If the power flow problem converges to a solution in the case of well-posed problems, $F(x) = 0$, meaning that there is no mismatch between specified power and calculated power at each bus. However, if the problem is ill-posed, there exists no solution $x \in \mathbb{R}^N$ that satisfies $F(x) = 0$.

From the optimal multiplier method described in section 7.1.2, we know that the following cost function is minimized in every step of the Newton-Raphson algorithm (see algorithm 4).

$$\begin{aligned} C(x) &= \frac{1}{2}F(x)^T F(x) \\ &= \frac{1}{2}[S^{sp} - S(x)]^T [S^{sp} - S(x)] \end{aligned} \tag{7.5}$$

As the cost function $C(x)$ is equivalent to the Euclidean distance between S^{sp} and $S(x)$, for a well-posed problem it approaches zero as the approximation $S(x)$ approaches the solution S^{sp}. Whereas for ill-posed problems, the mismatch $F(x)$ is,

$$F(x) = \Delta S(x) = S^{sp} - S(x) \neq 0 \tag{7.6}$$

And hence, provided there exists a minimum, the cost function $C(x)$ eventually reduces to the minimum value but not zero. Starting from an initial estimate such as flat start for

[5]Solvable operating point refers to the input parameters for which a power flow solution exists.

[6]Desired operating point refers to the input parameters of the ill-posed problem.

example, the approximation corresponding to the minimum value of the cost function is considered as the best possible approximation to the ill-posed problem. Let this approximation be denoted as $\widehat{x}$. As finding the optimal multiplier is a minimization problem, at the approximation $\widehat{x}$ we have (see equation 7.3),

$$\min_{\mu\in[0,1]} C(x+\mu\Delta x) = C(\widehat{x}) \tag{7.7}$$

An important observation to be made here is that the optimal multiplier method minimizes the cost function $C(x)$ in the direction of the correction vector Δx. Hence, the gradient of $C(x)$ is orthogonal to Δx at $\widehat{x}$. That is,

$$\nabla C(\widehat{x}) \cdot \frac{\Delta\widehat{x}}{||\Delta\widehat{x}||} = 0 \tag{7.8}$$

$$[F(\widehat{x})^T J(\widehat{x})] \cdot \frac{\Delta\widehat{x}}{||\Delta\widehat{x}||} = 0 \tag{7.9}$$

Since the mismatch $F(\widehat{x}) \neq 0$ for the ill-posed problem and $\Delta\widehat{x} \neq 0$, the Jacobain matrix $J(\widehat{x})$ is singular. As $J(x)$ approaches singularity, the optimal multiplier $\mu \to 0$ and $||\Delta x|| \to \infty$.

Figure 7.6 illustrates how the Newton-Raphson method with optimal multiplier tries to reduce the distance between S^{sp} and $S(x)$ in every iteration. The point $S(x^0)$ corresponds to the complex power vector calculated in the first iteration of the Newton-Raphson algorithm using the initial approximation x^0 which could be flat start or the solution of an approximate method such as DC load flow or fast decoupled load flow. If the problem is ill-posed, the desired operating point S^{sp} lies in the unsolvable region and during some iteration of the Newton-Raphson algorithm, the boundary σ is encountered. This boundary σ between the solvable region and the unsolvable region in parameter space plays a crucial role in regularizing the ill-posed power flow problem.

By definition, the boundary is the set of all input parameters for which the Jacobian is singular. That is, the boundary is a hypersurface in the parameter space that includes all the points corresponding to the input parameters that result in a singular Jacobian at some iteration of the optimal multiplier method. in principle a saddle node bifurcation occurs at the boundary.

Physically, as the operating point of the power system moves towards the boundary, voltage stability decreases and going beyond the boundary leads to voltage collapse. The phenomenon of voltage collapse is widely studied using the concepts of *bifurcation theory*. Bifurcation theory is the mathematical study of the changes in the stability of a

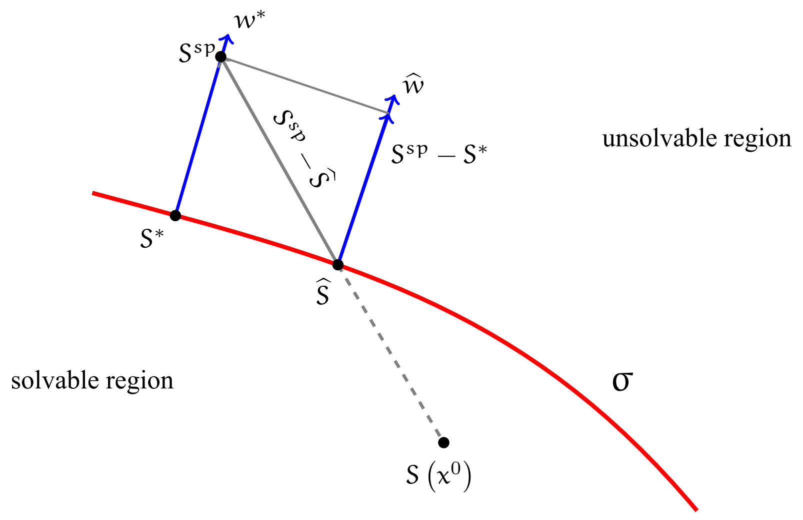


Figure 7.6: Moving the operating point in parameter space

dynamical system as the parameters of the system change.

In our case, the power system can be modelled as a dynamical system of the form:

$$\frac{dx}{dt} = f(x, S^{sp}(t)) \tag{7.10}$$

where $S^{sp}(t)$ is the specified power injection vector that changes over time, as in year-round simulations for example. The equation 7.10 is a parameter dependent differential equation and hence the stability of the system depends on the parameter $S^{sp}(t)$. The system is stable as long as the vector $S^{sp}(t)$ lies in the solvable region of the parameter space and is unstable if it is in the unsolvable region. As the vector $S^{sp}(t)$ moves from the solvable region to the unsolvable region, the system undergoes a saddle node bifurcation at the boundary. For further details about using bifurcation theory to understand the phenomenon of voltage collapse, see [24] and [25].

It is highly unlikely that the complex power vector $S(\widehat{x})$ calculated at the approximation $\widehat{x}$ is the solvable operating point that is closest to the desired operating point. That is, $S(\widehat{x}) \neq S(x^*)$, where $S(x^*)$ is the closest solvable operating point (see fig. 7.6 [7]). This is due to the fact that $S(\widehat{x})$ computed by the optimal multiplier method is not unique and depends on the initial approximation of the Newton-Rapshon method. As can be seen in figure 7.6, $S(x^0)$ could be anywhere in the solvable region depending upon the initial approximation x^0. Hence, in order to move from $S(\widehat{x})$ to $S(x^*)$, an iterative method is developed as follows.

[7] $S(\widehat{x})$ and $S(x^*)$ are denoted as $\widehat{S}$ and S^* respectively in fig. 7.6 for simplicity.

It is shown in [25] that the left eigenvector corresponding to the zero eigenvalue of the singular Jacobian is parallel to the normal vector to the boundary σ at $S(\widehat{x})$ and likewise at $S(x^*)$. We denote the two left eigenvectors, one at $S(\widehat{x})$ and the other at $S(x^*)$ by $\widehat{w}$ and w^* respectively. If the surface of the boundary σ is flat, the two eigenvectors $\widehat{w}$ and w^* are parallel and a projection of $[S^{sp} - S(\widehat{x})]$ onto the normalized direction of the left eigenvector $\widehat{w}$ gives the value $[S^{sp} - S(x^*)]$ from which $S(x^*)$ can be calculated:

$$S(x^*) = S^{sp} - [(S^{sp} - S(\widehat{x})) \cdot \widehat{w}]\widehat{w} \tag{7.11}$$

However, if the surface of σ is not flat, (7.11) is an approximation since the two left eigenvectors $\widehat{w}$ and w^* are no longer parallel. Hence, (7.11) is used as an update in every iteration of the optimal multiplier method, which leads to algorithm 5. The iterative method also emphasizes that the convergence of the algorithm is dependent on the shape of the boundary σ and the distance of the desired operating point S^{sp} from the boundary. Note that the algorithm is based on the assumption that the Jacobian has a unique zero eigenvalue at the boundary and to obtain a solution, the algorithm needs to run until the solvable operating point is a little inside the solvable region but not exactly on the boundary.

Algorithm 5: Regularizing ill-posed power flow problems

```
$k := 0$
Initialize: $x^0$, $S^0 = S^{sp}$
while not converged do
    Solve the problem using optimal multiplier method
    if converged then
        break
    else
        Denote the approximation as $\widehat{x}^k$
        Determine the normalized left eigenvector $\widehat{w}^k$
        Set $S^{k+1} = S^{sp} - [(S^{sp} - S(\widehat{x}^k))^T \widehat{w}^k]\widehat{w}^k$
        $k = k + 1$
    end
end
```

The difference between S^{sp} and $S(x^*)$ signifies the optimal change needed to best solve the ill-posed power flow problem. The distance in parameter space between the desired operating point S^{sp} and the closest solvable operating point $S(x^*)$ tells us how far the problem is from being solvable. This distance is introduced as the degree of unsolvability in [22] and provides good insight into the ill-posed power flow problems.

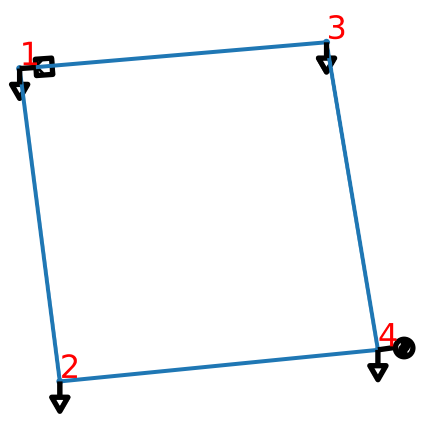
Considering Euclidean distance we have the degree of unsolvability as follows.

$$dS = \sqrt{(S^{sp} - S(x^*))^T (S^{sp} - S(x^*))} \tag{7.12}$$

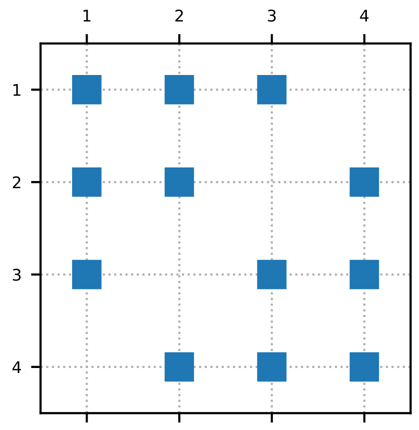
Example: 4-bus grid

To demonstrate the working principle of the methods and algorithms discussed in this chapter, we apply them to the 4-bus grid model available in pandapower (and MATPOWER). Figure 7.7 shows the grid model and the sparsity of the admittance matrix. The grid model consists of the following elements.

- Buses: 4
- Lines: 4
- Loads: 4
- Generators: 1
- External grids: 1



(a) Network



(b) Admittance matrix

Figure 7.7: 4-bus grid model

In this example, the 4-bus grid model is considered as the basis model for implementing the algorithms. That is, the basis model acts as a reference grid model for which power flow solution exists and can be calculated using the Newton-Raphson method (using algorithm 1). Now to demonstrate the algorithms, the power flow problem needs to be ill-posed. As discussed earlier in this chapter, a power flow problem could become ill-posed during contingency analysis and system planning due to changes in grid topology and input parameters, leading to a highly stressed system. Therefore, even in the literature on ill-conditioned and ill-posed power flow problems, case studies are designed in such a way that the power flow problem is ill-conditioned or ill-posed for a number of cases. Depending upon the application, a grid model is chosen as a basis model and either contingencies are defined or load scaling is done. Multiplying the active and reactive power demands of all the loads by a scalar is called as load scaling. Several examples of case studies can be found in [26].

For our example here, we define a combination of contingency and load scaling. The line between bus 1 and bus 3 (see fig. 7.7a) is put out of service[8] and the loads are scaled by a factor of three. Figure 7.8 shows the active power and reactive power of all the loads in the model multiplied by a scalar k. Note that $k = 1$ represents the base case and $k = 3$ represents our case study. The two models are to be thought of as the grid models that correspond to any two hours of a year. These are also known as snapshots. Specifically, a snapshot represents the input parameters of the grid model for a particular hour of the year.

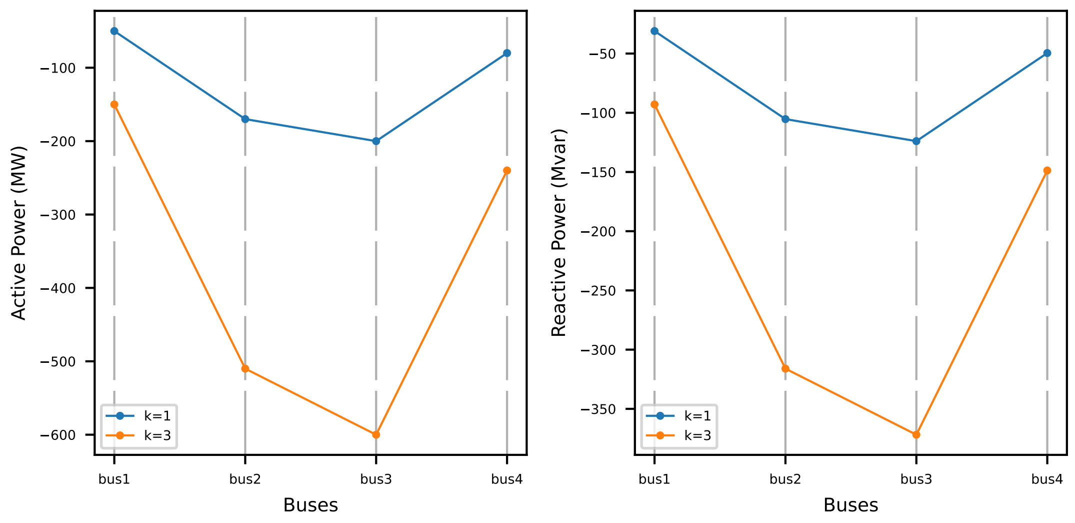


Figure 7.8: Load scaling

The polar power-mismatch version (see section 4.1.1) of the Newton-Raphson method is used for power flow simulations of the basis model and the modified model. Buses 1 and 4 are modelled as PV buses as they are connected to an external grid (slack bus) and a generator respectively. Buses 2 and 3 are modelled as PQ buses as they only have loads connected to them (see fig. 7.7a).

[8]If an element such as a line or a generator is put out of service, it is not considered for calculations.

For the basis model, the NR method yields a solution and is straightforward. However, for the modified model, the NR method fails to converge to a solution. Figure 7.9 shows the Euclidean norm of the mismatch function $F(x)$ and the condition number[9] of the Jacobian for both the cases. As is evident from the figure, $\|F(x)\|_2 \to 0$ for the basis model whereas for the modified model it oscillates and never reaches zero. For the basis model, the condition number of the Jacobian is small in all iterations but for the modified model, it oscillates and is higher than the basis model.

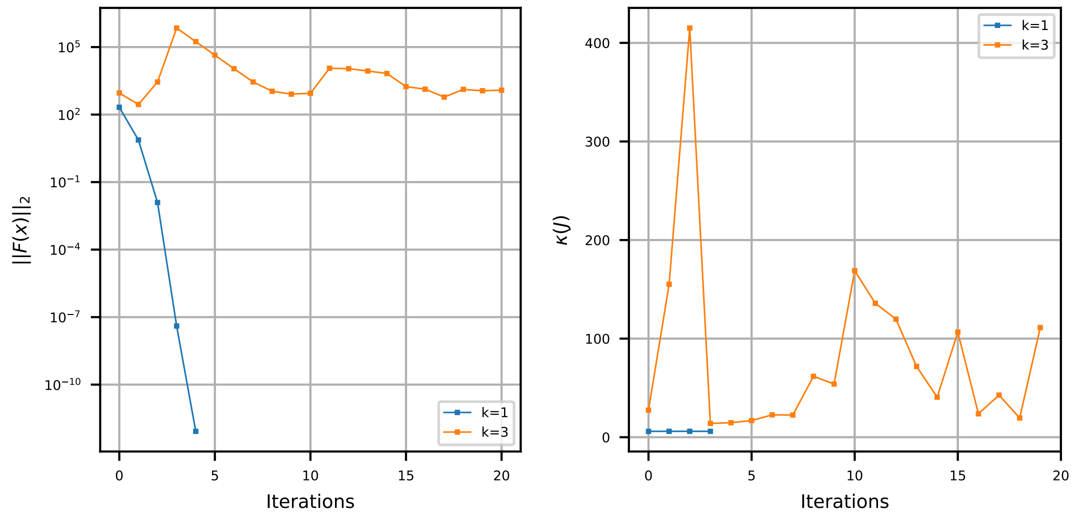


Figure 7.9: Mismatch function and condition number

At this point, it can be said that the power flow problem for the modified grid model is either ill-conditioned or ill-posed. That is, it could either lie in region 1.b or in region 2 of the parameter space (see figure 7.2). Next, we try to solve the problem using the optimal multiplier method. Despite using the optimal multipliers, the problem does not converge to a solution. Figure 7.10 shows the behavior of $\|F(x)\|_2$ and the condition number of the Jacobian in each iteration.

[9]The condition number of a non-singular matrix A is given by $\kappa(A) = \|A\|\|A^{-1}\|$ of the Jacobian matrix and tells us how far the matrix is from being singular.

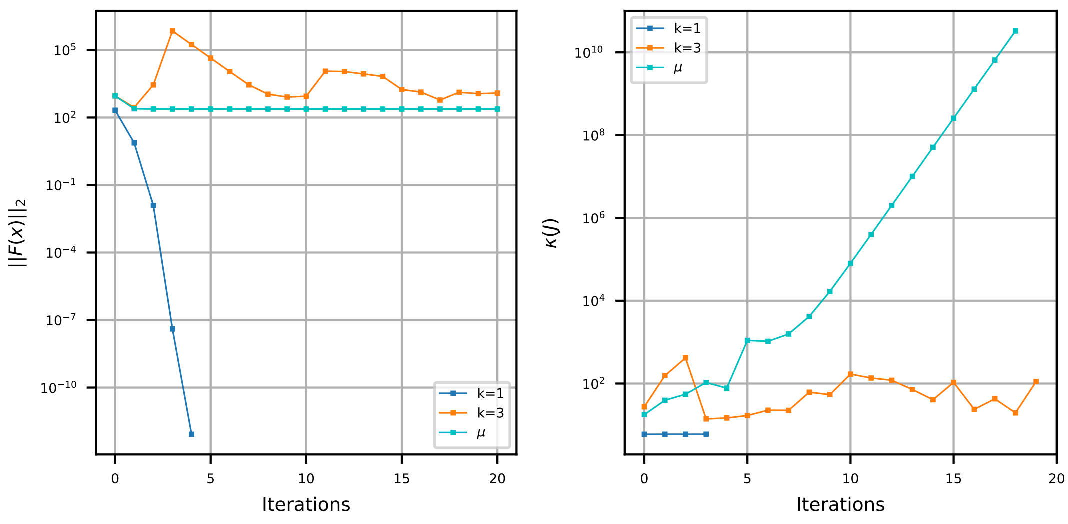


Figure 7.10: Using optimal multiplier method

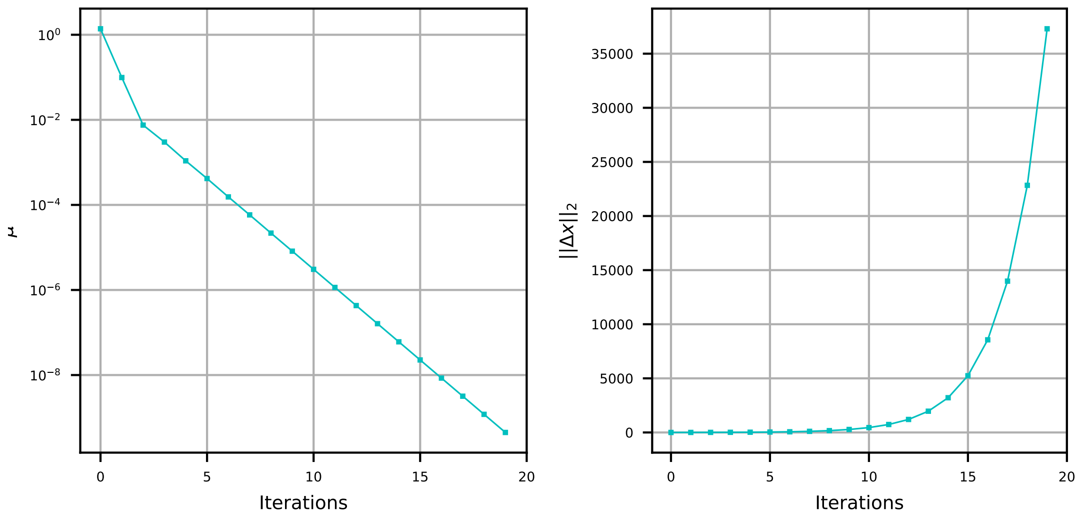


Figure 7.11: Using optimal multiplier method

It can be seen that $\|F(x)\|_2$ does not go to zero but settles down at a positive value and the Jacobian approaches singularity as indicated by the increase in the condition number. Figure 7.11 shows that the optimal multiplier $\mu \to 0$ and $\|\Delta x\|_2 \to \infty$. This corresponds to the theory discussed earlier in this section, in particular equation 7.8, thus confirming that the problem is ill-posed.

To regularize the problem, algorithm 5 is used. For the first iteration (outer loop) of the regularizing algorithm, flat start is considered as the initial approximation and for the subsequent iterations, the solution of the previous iteration is used as the initial approximation for the optimal multiplier method which forms the inner loop of the algorithm. The first iteration runs until the Jacobian matrix becomes singular (as shown in fig. 7.10). At this point, the rank of the Jacobian matrix is $N - 1$ and the power injection vector calculated by the optimal multiplier method corresponds to the point $S(\widehat{x})$ on the boundary σ in the parameter space (fig. 7.6). The singular Jacobian matrix indicates that the power injection vector $S(x)$ has encountered the boundary σ, leading to a saddle node bifurcation. From this point on, the left eigenvector is used in each iteration to update the power injection vector and thereby steer the system towards solvability.

In three outer iterations, the power flow problem converges to a solution. Figure 7.12 shows the condition number and the rank of the Jacobian matrix in all the iterations (including the inner loop) since the start of the algorithm. It is clear from the figure that the condition number of the Jacobian matrix decreases as the updates to the power injections are made and the rank is back to N in just a few iterations.

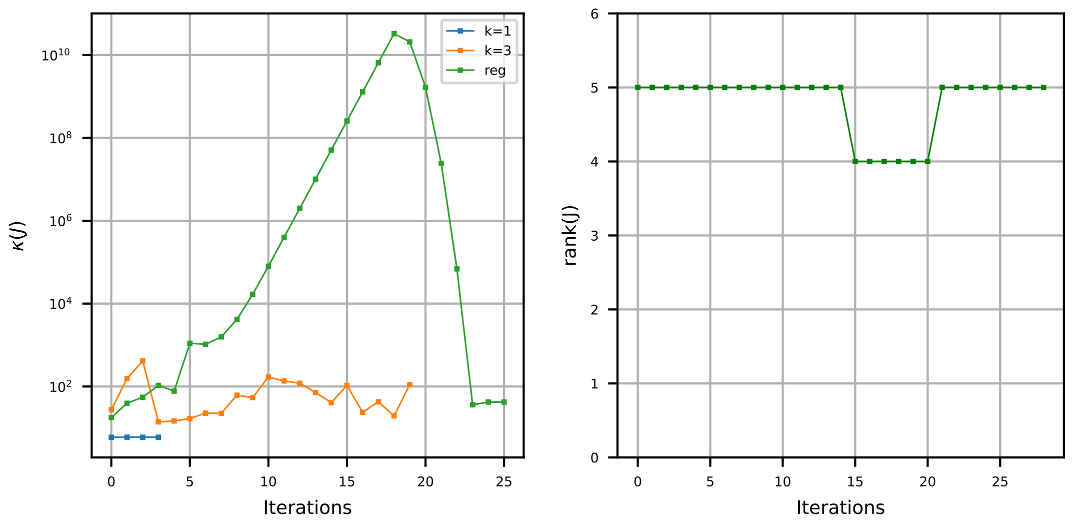


Figure 7.12: Condition number and rank of the Jacobian

Changes are made to active and reactive power of loads and active power of the generator as shown in figure 7.13. The power injection vector corresponding to the updated power injections is the closest solvable operating point $S(x^*)$ in the parameter space (see fig. 7.6). Note that in figure 7.13 and in the figures hereafter, *reg* stands for regularized.

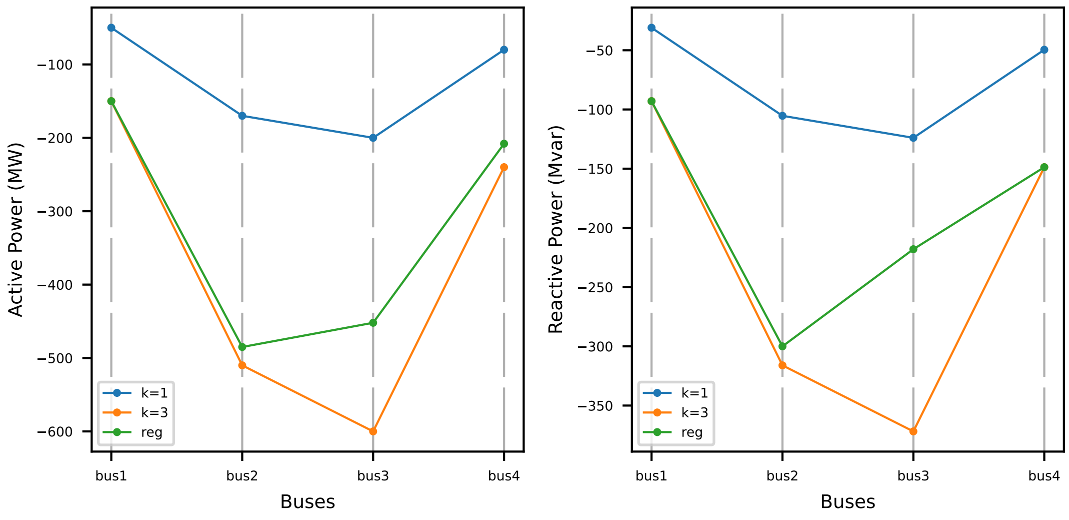


Figure 7.13: Updated power injections

Figure 7.14 shows a comparison of voltage magnitudes and voltage angles between the basis model and the regularized modified model.

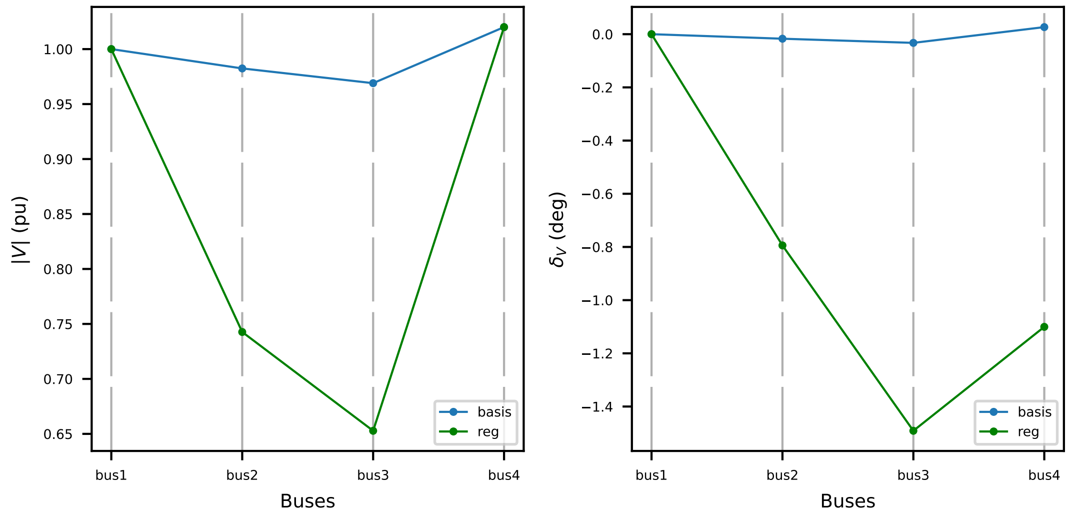


Figure 7.14: Voltage magnitudes and angles

This example demonstrates in detail the working principle of the mathematical framework that we developed to solve both converging and diverging power flow problems and hence automate year-round AC power flow simulations. In the remaining chapters, we describe the automating algorithm and the practical challenges of implementing it.

Chapter 8

Automating AC Power Flow Simulations

The goal of this project is to develop an algorithm that can automate AC power flow simulations for performing year-round calculations. By automation, we mean that the algorithm should be able to perform power flow simulations with a high degree of success. That is, given a grid model, irrespective of whether its power flow problem is well-posed or ill-posed, the objective is to find a solution. While a well-posed problem is solved as it is, an ill-posed problem is optimally regularized. Furthermore, as often emphasized in the previous chapter, the algorithm is based on the assumption that the grid model is free of modelling errors. The algorithm is meant to be used in TenneT to perform AC year-round simulations with little manual intervention. We would like to reiterate that generally, during year-round simulations, only the load and generation data change while the topology of the grid remains as it is.

In this chapter, the algorithm and its implementation details are described. To demonstrate the capabilities and limitations, several test networks most of which are commonly used in the literature to test power flow solution methods are considered.

8.1 The automating algorithm

In essence, the automating algorithm is a consolidation of all the relevant power flow solution methods discussed in the previous chapters. Figure 8.1 shows a flowchart that depicts the steps involved in the automating algorithm. The three steps $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$ correspond to well-conditioned, ill-conditioned and ill-posed problems respectively. Algorithm 6 shows the process of automating AC power flow simulations for several grid models or a single grid model with varying hourly load and generation data.

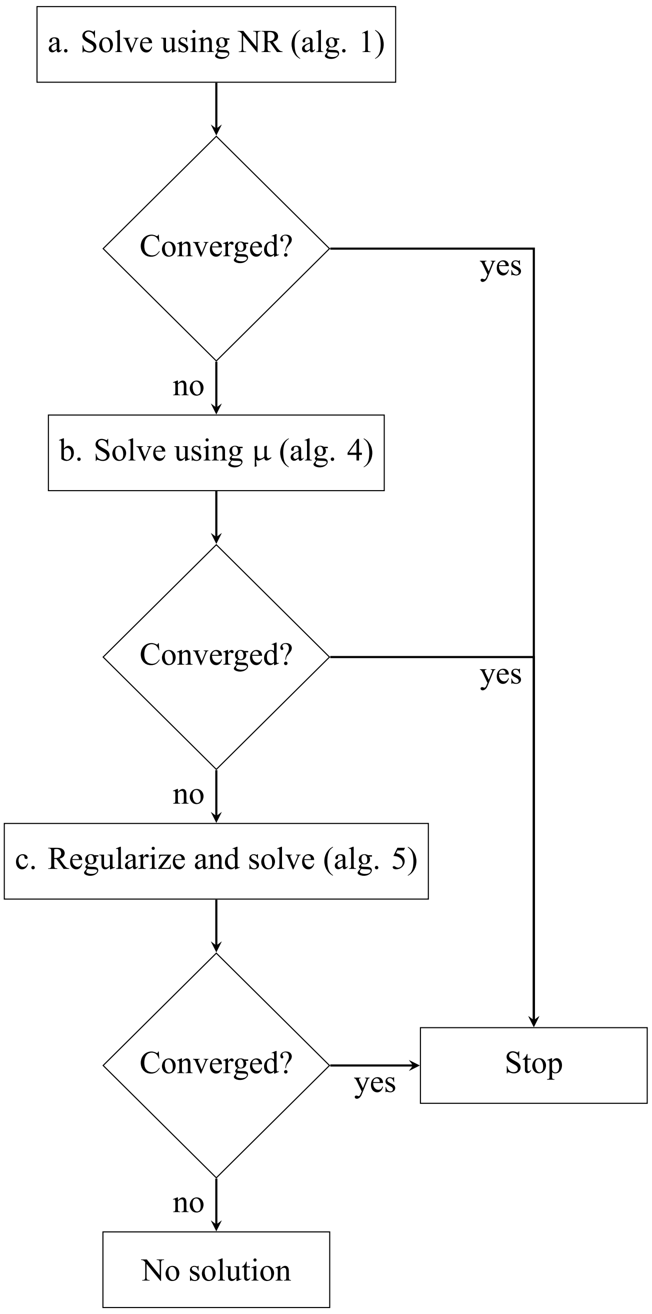

a. Solve using NR (alg. 1)
Converged?
yes
no
b. Solve using μ (alg. 4)
Converged?
yes
no
c. Regularize and solve (alg. 5)
Converged?
yes
Stop
no
No solution

Figure 8.1: Flow of automation

(a) The Newton-Raphson method: The polar power-mismatch version of the Newton-Rapshon method (4.1.1) is used to solve the power flow problem. The linear system $-J(x)\Delta x = F(x)$ is solved in double precision using sparse LU decomposition. The convergence tolerance is set to 10^{-8} and the maximum number of iterations is set to 10. In this project we use pandapower [19] for doing power flow simulations.

(b) Optimal multipliers: An ill-conditioned power flow problem is solved using optimal multipliers that reduce the step size of the Newton-Raphson method in each iteration. To determine the optimal multipliers, the following cost function in minimized (see section 7.1.2).

$$C(x) = \frac{1}{2}F(x + \mu\Delta x)^T F(x + \mu\Delta x) \tag{8.1}$$

where,

$$F(x + \mu\Delta x) = F(x) + \mu J(x)\Delta x + \mu^2 F(\Delta x) \tag{8.2}$$

For simplicity, equation 8.2 can be written as $a + b\mu + c\mu^2$, where $a = F(x)$, $b = J(x)\Delta x$ and $c = F(\Delta x)$. The cost function $C(x)$ is minimized using $\frac{dC}{d\mu} = 0$, which results in the following scalar cubic equation.

$$g_0 + g_1\mu + g_2\mu^2 + g_3\mu^3 = 0 \tag{8.3}$$

where, $g_0 = a^T b$, $g_1 = 2a^T c + b^T b$, $g_2 = 3b^T c$ and $g_3 = 2c^T c$. We solve the equation 8.3 using NumPy to determine the optimal multiplier μ in each iteration of the optimal multiplier method.

(c) Regularization: Ill-posed power flow problems are regularized using the method discussed in section 7.1.3. Algorithm 5 is an extension of the optimal multiplier method, the addition being the computation of eigenvectors. We compute the eigenvectors using the shift and invert Arnoldi method[1], which is known to be well suited for finding eigenvalues close to a certain scalar α. In our case, we intend to find the eigenvalues closest to $\alpha = 0$. In the first outer iteration of algorithm 5, the inner loop (optimal multiplier method) is executed until the eigenvalue closest to $\alpha = 0$ is less than a predefined tolerance close to zero (10^{-8}), which indicates that the Jacobian is singular. In each subsequent outer iteration, the inner loop is executed for a predefined number of iterations (10 or 20 depending upon the problem) and the eigenvector associated with the eigenvalue that is closest to zero is used for updating the power injection vector S (see alg. 6). Using the eigenvalues that are computed anyhow in each iteration to check for singularity of the Jacobian matrix in the first outer iteration significantly improves the speed of the algorithm. For a detailed explanation about the shift and invert Arnoldi method, see [27]. In our algorithms, we use the Python library SciPy which is based on ARPACK [28] to solve the eigenvalue problems.

[1]The Arnoldi method is an iterative method that is used to approximate eigenvalues of matrices.

Algorithm 6: Automating AC power flow simulations

```
for each grid model do
    $i := 0$, $x^0$ = flat start
    while not converged do
        Solve for the correction: $-J(x^i)\Delta x^i = F(x^i)$
        Update the approximation: $x^{i+1} = x^i + \Delta x^i$
        $i = i + 1$
    end
    if converged then
        continue
    else
        $j := 0$, $x^0$ = flat start
        while not converged do
            Solve for the correction: $-J(x^j)\Delta x^j = F(x^j)$
            Compute $\mu^j$ by evaluating $\frac{dC}{d\mu} = 0$, set $\Delta x^j = \mu^j \Delta x^j$
            Update the approximation: $x^{j+1} = x^j + \Delta x^j$
            $j = j + 1$
        end
        if converged then
            continue
        else
            $k := 0$, $\widehat{x}^k = x^{j+1}$
            while not converged do
                Determine the normalized left eigenvector $\widehat{w}^k$ of $J(\widehat{x}^k)$
                Set $S^{k+1} = S^{sp} - [(S^{sp} - S(\widehat{x}^k))^T \widehat{w}^k]\widehat{w}^k$
                $l := 0$, $\widehat{x}^l = \widehat{x}^k$
                while $\|F(\widehat{x}^l)\|_2 > 10^{-8}$ or $l < 10$ do
                    Solve for the correction: $-J(\widehat{x}^l)\Delta\widehat{x}^l = S^{k+1} - S(\widehat{x}^l)$
                    Compute $\mu^l$ by evaluating $\frac{dC}{d\mu} = 0$, set $\Delta x^l = \mu^l \Delta x^l$
                    Update the approximation: $x^{l+1} = x^l + \Delta x^l$
                    $l = l + 1$
                end
                $\widehat{x}^{k+1} = \widehat{x}^l$, $k = k + 1$
            end
            if converged then
                continue
            else
                there is no solution
            end
        end
    end
end
```

8.2 Implementation

To demonstrate the feasibility and practical challenges of the automating algorithm, we apply it to the test network Case 1354pegase, which represents a part of the European transmission grid [29]. The grid is made of the following elements.

- Buses: 1354
- Lines: 1751
- Shunts: 1082
- Two-winding transformers: 240
- Loads: 621
- Generators: 259
- Static generators: 52
- External grids: 1

The grid model consists of 259 PV buses, 1094 PQ buses and 1 slack bus. The Jacobian $J(x) \in \mathbb{R}^{2447 \times 2447}$. Figure 8.2 shows the sparsity of the admittance matrix of the network.

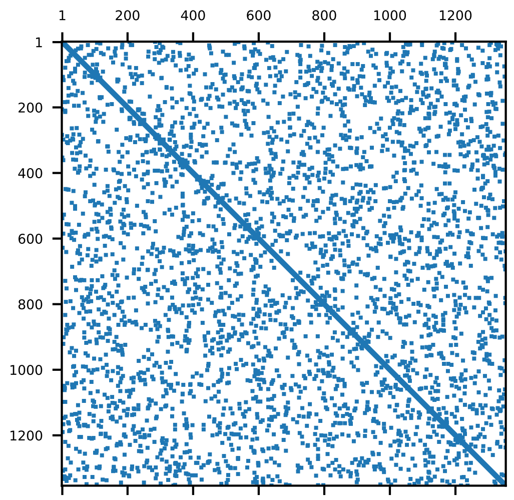


Figure 8.2: Case 1354pegase: Admittance Matrix

Similar to the 4-bus example in chapter 7, the loads of the 1354-bus grid model in this example are scaled by a factor of $k = 1.4$, which for this grid model is the smallest scalar for which a solution cannot be found using the conventional Newton-Raphson method. Additionally, $N - 1$ contingencies are defined for the lines. That is, as many contingencies as there are lines in the grid model are defined and in each contingency, a single line is removed from the grid model. Hence, in this example, 1751 contingencies are defined for the 1354-bus grid model. The automating algorithm is applied to each contingency and it successfully regularizes and solves 1744 out of 1751 contingencies.

Figure 8.3 shows the 20 smallest voltage magnitudes in the network of a randomly chosen regularized contingency. The numbers on the x-axis are the bus indices in the pandapower grid model. As expected, the voltage magnitudes drop due to the scaling of loads.

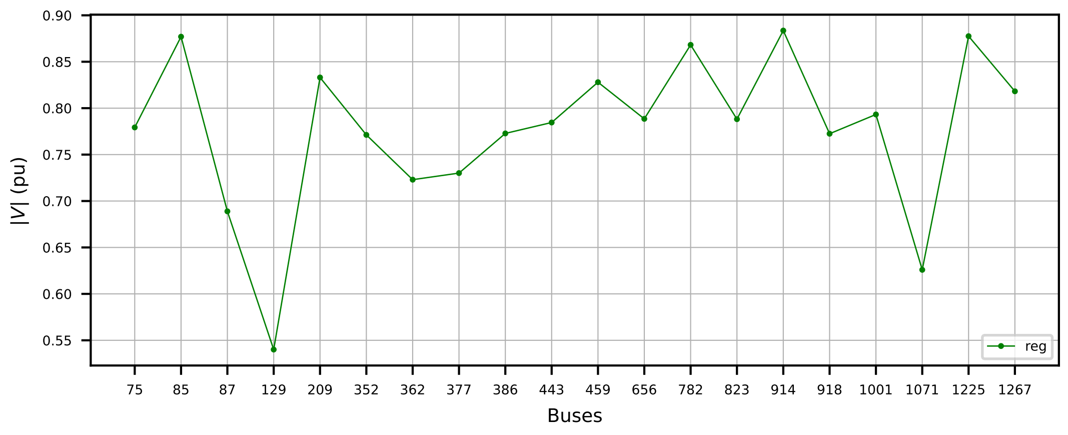


Figure 8.3: Case1354pegase: 20 lowest voltage magnitudes in the network of a randomly chosen contingency.

Table 8.1 shows the total active and reactive powers of loads. Active power is the total demand (and supply) in the grid, and reactive power is the sum of the absolute values of reactive powers of loads.

Table 8.1: Case1354pegase: active and reactive powers of loads.

Case1354pegase	Active Power (MW)	Reactive Power $(Mvar)$
Basis model	74146	17018
Scaled model $(k = 1.4)$	103804	23826

Among all the contingencies, figure 8.4 shows 20 largest changes made to the total active power of loads and figure 8.5 shows 20 largest changes made to the total reactive power of loads. The numbers on the x-axis correspond to the indices of the lines that are removed from the grid model to define contingencies. Note that modifications are also made to active powers of generators and static generators, and reactive powers of shunts. In this example, we only show the modifications made to the loads to get an idea about how the changes are made to regularize and solve an ill-posed power flow problem during contingency analysis. It can be seen from the figures that the changes made to reactive powers are very less compared to the changes made to active powers. This explains that there is no need to change the reactive powers as much as active powers in the network to obtain a solution, which is confirmed by the fact that the problem will still be well-posed if only reactive powers of loads are multiplied by the scalar $k = 1.4$ and active powers are not.

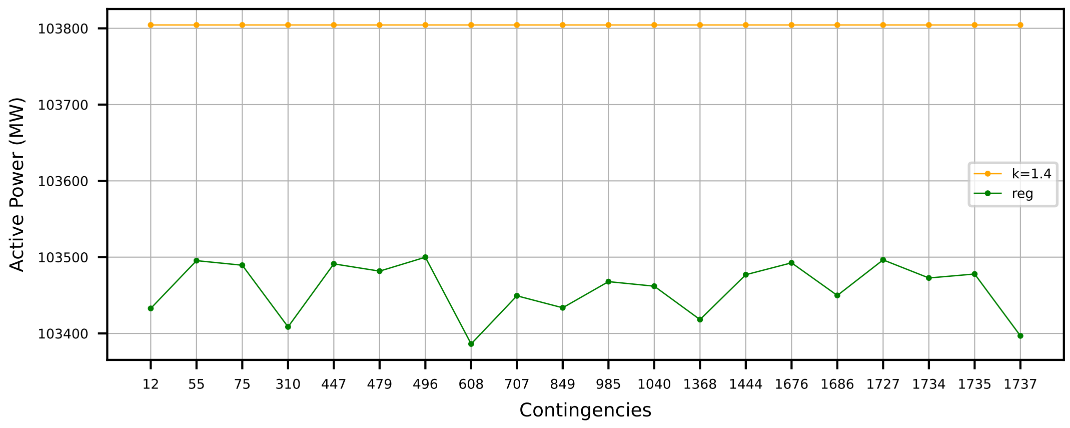


Figure 8.4: Case1354pegase: 20 largest changes in total active power of loads.

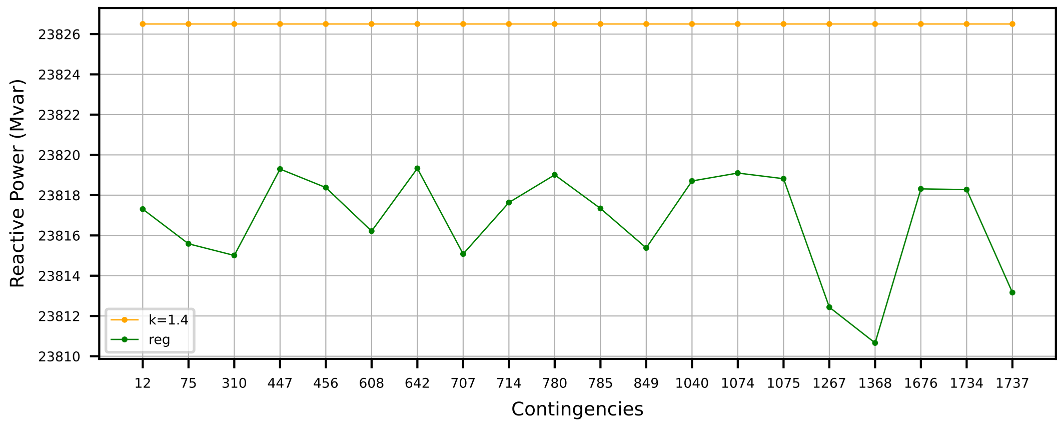


Figure 8.5: Case1354pegase: 20 largest changes in total reactive power of loads.

Furthermore, we apply the automating algorithm to 26 test networks that are available in pandapower. This exercise is to demonstrate the applicability of the automating algorithm to a variety of grid models, ranging from a small 4-bus grid to a large 9241-bus grid. The loads of each grid model are appropriately scaled by a factor k to make sure that the power flow problem is ill-posed. In this exercise, contingencies are not defined. The automating algorithm solves 19 grid models out of 26. Table 8.2 shows the scalar k used for each grid model and the active and reactive powers of loads.

Table 8.2: Test grids: Total active and reactive powers of loads.

Grid model	k	Active Power (MW)		Reactive Power $(Mvar)$	
		Scaled	Regularized	Scaled	Regularized
Case 4gs	4.6	2300	2285	1425	1413
Case 6ww	3.6	756	628	756	666
Case 9	4.0	1260	936	460	300
Case 14	4.2	1087	1067	341	335
Case 24	2.0	5700	5577	1160	1130
Case 30	5.0	946	876	536	528
Case IEEE30	3.5	991	979	441	432
Case 33bw	4.0	14	14	9	8
Case 39	1.4	8443	83	2301	2295
Case 57	2.0	2501	2495	672	667
Case 89pegase	2.5	20396	20558	5778	6034
Case Illinois200	2.5	5571	5437	1587	1567
Case 300	1.1	26232	26039	8802	8799
Case 1888rte	1.2	71528	71573	12980	13015
Case 2848rte	1.2	64606	64581	14774	14788
Case 6470rte	1.2	130293	129500	24708	24754
Case 2869pegase	1.2	166721	166388	40805	40804
Case 9241pegase	1.1	368950	368469	112675	112675

The test cases that do not converge to a solution can be understood by the following explanation.

Singularity of the Jacobian

As discussed in chapter 7 (section 7.1.3), for the regularizing method to work, the optimal multiplier μ must approach zero which indicates that the Jacobian becomes singular. This is explained by the equation 7.8 which is given by,

$$[F(\widehat{x})^T J(\widehat{x})] \cdot \frac{\Delta\widehat{x}}{||\Delta\widehat{x}||} = 0$$

and corresponds to the minimum of the cost function $C(x)$. This is indeed true for all the previously discussed test cases that converge to a solution. Whereas for the test cases that do not converge, the optimal multiplier oscillates and the Jacobian does not become singular, which means that the method fails to determine the minimum of the cost function $C(x)$. For example, this is observed in the IEEE-118 grid model with $k = 2$. Figure

8.6 shows the optimal multiplier μ and the cost function $C(x)$, which indicate that the Jacobian is non-singular.

In this case, to determine the minimum of the cost function $C(x)$, in each iteration we multiply the optimal multiplier by a preconditioner[2] $\omega \in (0, 1)$. Note that $\omega = 1$ corresponds to the optimal multiplier method discussed so far in this thesis. Algorithm 7 shows the optimal multiplier method with the preconditioner.

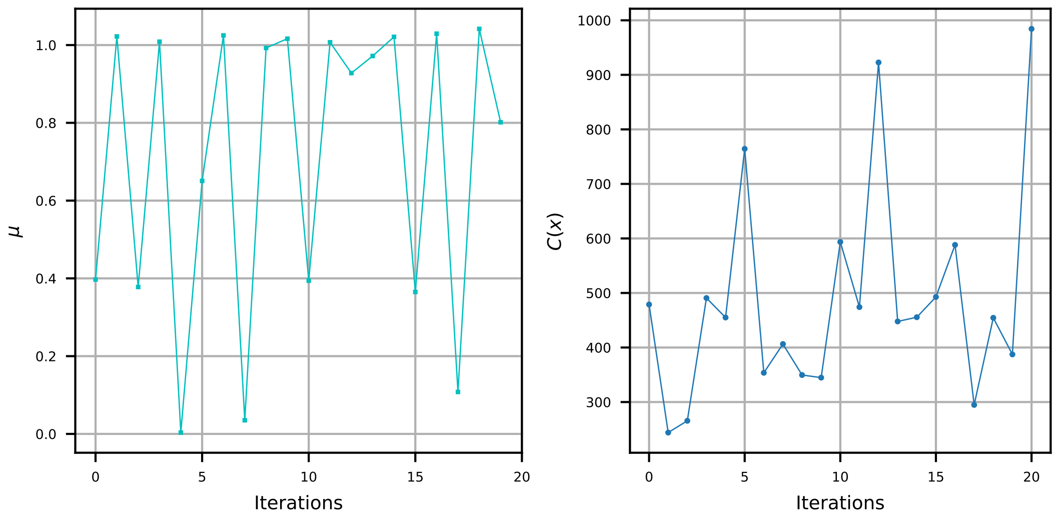


Figure 8.6: IEEE-118: Optimal multiplier μ and cost function $C(x)$ in each iteration of the optimal multiplier method.

Algorithm 7: Optimal multiplier method with preconditioner

```
$k := 0$
Initialize: $x^0$, $\omega$
**while** *not converged* **do**
    Solve for the correction: $-J(x^j)\Delta x^j = F(x^j)$
    Compute $\mu^j$ by evaluating $\frac{dC}{d\mu} = 0$
    Set $\Delta x^j = \omega\mu^j\Delta x^j$
    Update the approximation: $x^{j+1} = x^j + \Delta x^j$
    $j = j + 1$
**end**
```

Choosing an optimal preconditioner ω for the problem depends on the application. In our case, the objective is to further reduce the step size so that the minimum of the

[2]In mathematics, a preconditioner is used to make a problem better suited for numerical treatment.

cost function can be determined. For the IEEE-118 grid model with $k = 2$, choosing $\omega = 0.2$ leads to the minimum of the cost function in each iteration and hence a solution can be determined by the automating algorithm. The problem converges to a solution in 7 iterations. Figure 8.7 shows the condition number of the Jacobian for all the cases (basis, scaled and regularized), and the Euclidean norm of the mismatch function in each iteration. In each outer iteration (color coded), the norm reduces as can be seen in the figure (right). Similarly, the same preconditioner $(\omega = 0.2)$ also solves the rest of the test grids considered in this project, for which a solution cannot be found without using a preconditioner. Table 8.3 shows the results of those grid models.

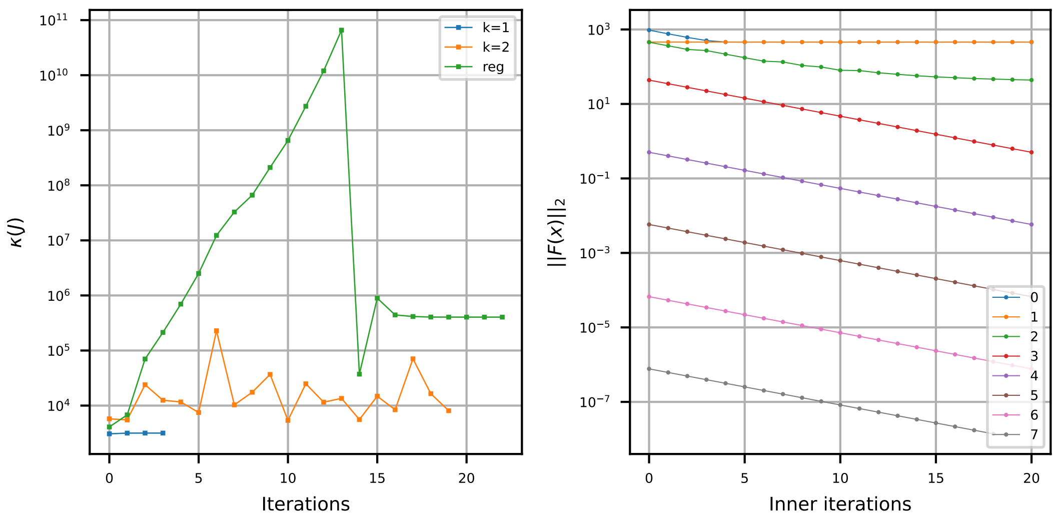


Figure 8.7: IEEE-118: Convergence of the ill-posed problem, using preconditioner.

Table 8.3: Test grids: Total active and reactive powers of loads. Solved using $\omega = 0.2$.

Grid model	k	Active Power (MW)		Reactive Power $(Mvar)$	
		Scaled	Regularized	Scaled	Regularized
Case IEEE-118	2.0	8484	8160	2876	2872
Case IEEE-145	1.1	314687	314455	87728	87810
Iceland	1.4	1914	1864	613	609
Case 3120sp	2.6	55071	44450	23558	23081
Case 6495rte	1.2	137783	136245	26793	26726
Case 6515rte	1.2	142898	137298	28491	28244
GB network	1.2	103106	100854	35482	34934
GB reduced network	2.5	140814	139686	40623	40614

8.3 Constraints

As demonstrated so far, the automating algorithm regularizes and solves ill-posed power flow problems by making optimal changes to power injections. Optimal changes mean that the algorithm makes least changes to power injections to just find a solution. Even though finding a solution makes a big difference in understanding the grid and its power flow problem better, it is not sufficient for practical purposes. For example, the algorithm might impose changes in power to buses in the network that are not connected either to a generator or to a load but connected only to lines. This is obviously not a practical solution. Furthermore, in TenneT, there is no need to change active power injections since active power is always in balance during year-round simulations whereas changes are to be made to reactive power injections by adding or removing shunts, for applications such as reactive power compensation assessments. Hence, to make the algorithm suitable for real-world networks and practical applications, we define constraints to the problem.

In this project we define constraints such that the power injections are modified only at predefined buses. That is, based on the application and feasibility, a set of buses is chosen in which power injections are allowed to change. For example, if buses that are connected to loads are chosen, only the power injections of loads are changed. The power injections in the rest of the buses in the network remain as they are. To impose these constraints, we define a constraint vector $g := [g_i] \in \mathbb{N}_0^{2N-N_g-2}$ where,

$$g_i = \begin{cases} 1, & \text{if a change is allowed at the bus} \\ 0, & \text{otherwise} \end{cases} \tag{8.4}$$

In each outer iteration of algorithm 5, the left eigenvector $\widehat{w}$ is multiplied element-wise with the constraint vector g. Algorithm 8 shows algorithm 5 with the additional step included. Note that the vector g can also be used for other elements such as generators. For example, if the power injections of a few generators are not allowed to change, the constraint vector g can be suitably defined. Simply put, the vector g ensures that changes to the specified power injection vector S^{sp} are made only at buses where $g_i = 1$.

Example: 1354-bus grid

To demonstrate the working of the algorithm with constraints, we apply it to the 1354-bus test grid considered earlier in this chapter. The loads are scaled by $k = 1.4$ to make the power flow problem ill-posed. We define the constraint vector g such that changes in reactive power are allowed only at buses that are connected to shunts and changes in active power are allowed at loads, generators and static generators. The algorithm

Algorithm 8: Regularizing ill-posed power flow problems with constraints

```
k := 0
Initialize: $x^0$, $S^0 = S^{sp}$, g
**while** *not converged* **do**
    Solve the problem using optimal multiplier method
    **if** *converged* **then**
        break
    **else**
        Denote the approximation as $\widehat{x}^k$
        Determine the normalized left eigenvector $\widehat{w}^k$
        $\widehat{w}^k = \widehat{w}^k \circ g$
        Set $S^{k+1} = S^{sp} - [(S^{sp} - S(\widehat{x}^k))^T \widehat{w}^k]\widehat{w}^k$
        $k = k + 1$
    **end**
**end**
```

regularizes and solves the problem in 9 iterations. Figure 8.8 shows the Euclidean norm of the mismatch $F(x)$ in each iteration, both inner (along x-axis) and outer (colored), and the eigenvalues (λ) in each outer iteration.

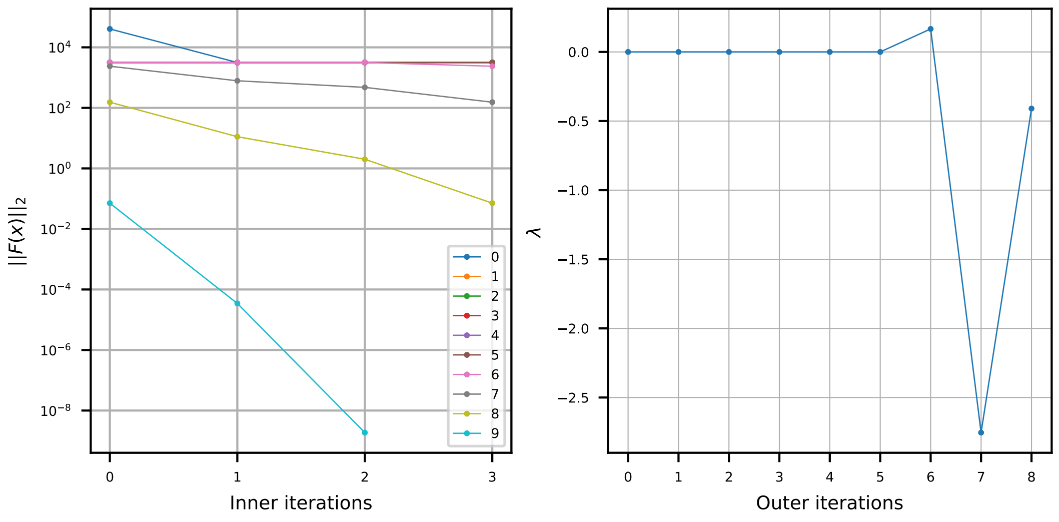


Figure 8.8: Case1354pegase: Euclidean norm of the mismatch function $F(x)$ in all the iterations (left), eigenvalues of the Jacobian closest to zero in each outer iteration (right).

The first step is to solve the ill-posed power flow problem using the optimal multiplier method until the Jacobian becomes singular. We call this the 0^{th} iteration. As seen in the figure, $||F(x)||_2$ reduces and settles at a positive value greater than zero in the 0^{th} iteration. In the subsequent outer iterations, $||F(x)||_2$ reduces and in the 9^{th} iteration becomes lesser than the tolerance (10^{-8}), hence converging to the solution. As discussed in chapter 7, to obtain a solution, the closest solvable operating point $S(x^*)$ needs to be inside the solvable region. This means that in the last few iterations of the algorithm, the Jacobian is not singular. This can be seen in figure 8.8 where the smallest eigenvalues are not zero in the last three outer iterations.

For this example, we restrict the number of inner iterations to 3 since the cost function remains the same after 3 iterations and does not make a difference to the convergence of the automating algorithm. Note that the more stringent the constraints are, the more number of iterations the problem takes to converge. Hence, the constraints should be carefully defined to ensure convergence.

Table 8.4 shows the power exchanges in the grid. As we know, scaling the loads by $k = 1.4$ means a 40% increase in active and reactive powers of loads. As shown in the table, regularizing the problem with the constraints applied to this problem involves a 0.44% decrease in total active power of loads and a 0.028% decrease in total reactive power of shunts. This means that a solution to the ill-posed problem can be determined by $461\ MW$ of load shedding and $4\ Mvar$ of reactive power compensation.

Table 8.4: Case1354pegase: power exchanges.

Case1354pegase	Active Power (MW)	Reactive Power $(Mvar)$	
		Loads	Shunts
Basis model	74146	17018	14820
Scaled model	103804	23826	14820
Regularized model	103343	23826	14816

This example shows that provided the constraints are feasible, an ill-posed power flow problem can be regularized and solved by the automating algorithm even with constraints defined. This makes the algorithm suitable for practical applications such as year-round AC power flow simulations, reactive power compensation assessments and contingency analysis.

Chapter 9

Results

In this chapter we discuss the results of applying the automating algorithm to two real-world electricity grids: the Dutch transmission grid and a sub-European transmission grid.

9.1 The Dutch transmission grid

In the Netherlands, the national high-voltage transmission grid, hereafter called the Dutch grid, is operated and managed by TenneT. The Dutch grid has cross border connections with Germany, Belgium, England, Norway and Denmark. The voltage levels in the grid are $110\,\mathrm{kV}$and higher, and the frequency is $50\,\mathrm{Hz}$. The grid constitutes about $11,500\,\mathrm{km}$ of high-voltage lines. Figure 9.1 shows the map of the Dutch grid.

For our simulations we consider the basis model of the Dutch grid, and load and generation data of 7 randomly chosen hours of the year 2030. This can be analogously extended to all the hours of a year or a decade, which constitutes year-round simulations. The basis model consists of the following elements.

- Buses: 14685
- Switches: 15517
- Lines: 1197
- Shunts: 160
- Two-winding transformers: 439
- Three-winding transformers: 216
- Loads: 334
- Generators: 4
- Static generators: 352
- External grids: 1

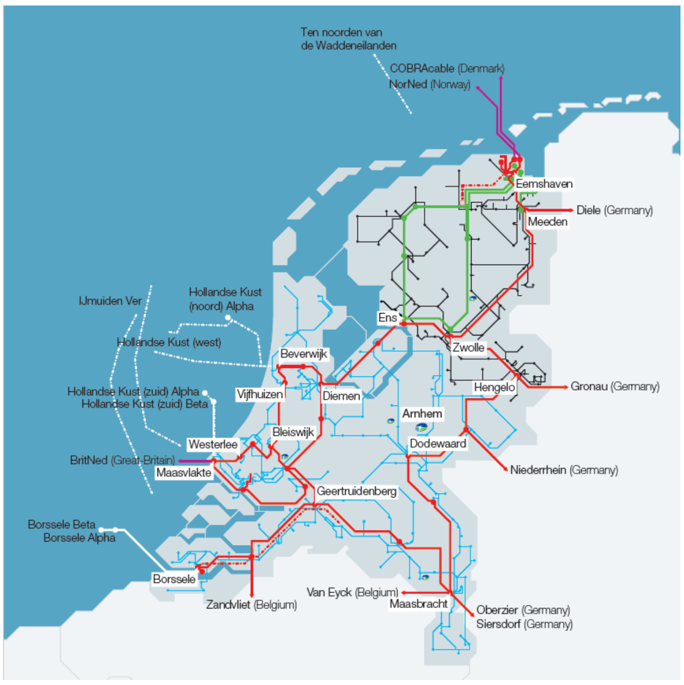


Figure 9.1: The Dutch transmission grid

Figure 9.2 shows the total active power demand (or supply) in the grid for the 7 hours considered in this project. The hours are counted from the beginning of the year. For example, hour 1 corresponds to 1 January 12:00 to 1:00 and hour 0737 corresponds to 31 January 16:00 to 17:00 and so on.

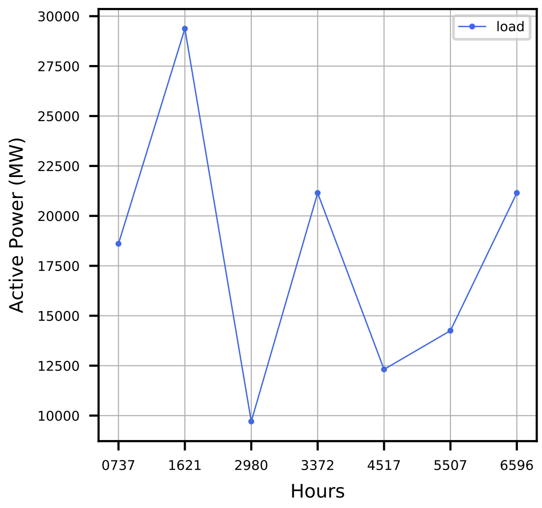


Figure 9.2: The Dutch grid: Total active power demand of loads (or total active power supply from generators).

Each hour of the year is represented by a PowerFactory grid model which consists of the basis model with load and generation data for one particular hour of the year. Hence, for one year there are 8760 PowerFactory grid models. In this project we convert the PowerFactory grid models that correspond to the 7 hours considered in this project to pandapower using the interface for simulations.

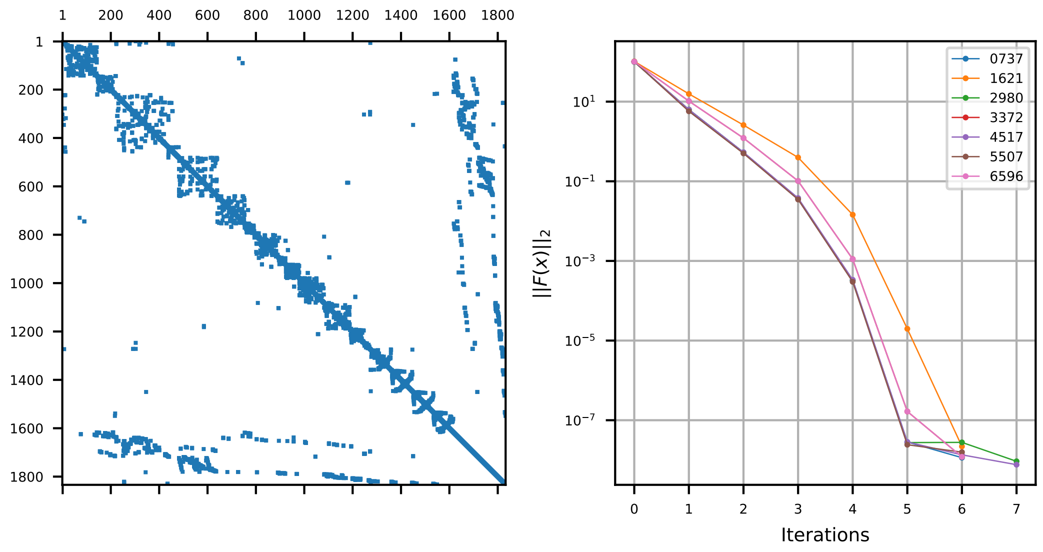


Figure 9.3: The Dutch grid: admittance matrix (left) and convergence (right).

The basis model consists of 4 PV buses, 1828 PQ buses and 1 slack bus. Merging the buses connected by closed switches leads to 1833 buses in the network and the order of the Jacobian is 3660. We perform AC power flow simulations using the polar power-mismatch version of the Newton-Raphson method. The convergence tolerance is set to 10^{-8} and maximum number of iterations is set to 10. All 7 power flow problems are well-posed and converge to a solution within 7 iterations. Figure 9.3 shows the admittance matrix of the network and convergence.

9.2 The sub-European transmission grid

The sub-European transmission grid considered in this project, hereafter called as the EU grid, consists of the high-voltage transmission grids of the Netherlands, Germany, Belgium, Luxembourg and France. The voltage levels in the network are $110\ \mathrm{kV}$ and higher, and the frequency is $50\,\mathrm{Hz}$. The grid constitutes about $83,527\,\mathrm{km}$ of high-voltage lines, spread roughly across an area of $1,075,641\,\mathrm{km}^2$.

For our simulations, we consider two versions of the basis model of the sub-European grid. Table 9.1 shows the grid elements of the two versions and table 9.2 shows active and reactive power exchanges in the grid.

Table 9.1: Grid elements of two versions of the sub-European grid

Grid Elements	Version-1	Version-2
Terminals	23548	24508
Switches	17225	17928
Lines	7355	7585
Shunts	827	868
Transformers (2w)	1637	1786
Transformers (3w)	1276	1305
External Grids	230	230
Loads	3947	3920
Generators	176	140
Static Generators	9871	8873

Table 9.2: EU grid: power exchanges

EU grid	Active Power (MW)	Reactive Power $(Mvar)$	
		Loads	Shunts
Version 1	134256	31113	705374
Version 2	192288	41625	708099

We convert the two versions from PowerFactory to pandapower using the interface. The power flow problem of version-1 is found to be well-posed in PowerFactory but ill-posed in pandapower. This can be explained by the fact that PowerFactory removes modelling errors to some extent by making changes to the characteristics of grid elements as explained in chapter 7. However, the changes made are not known us and hence the errors are carried over to pandapower through the interface. We thus regard version-1 as ill-posed and solve it using the automating algorithm. Version-2 is ill-posed in both PowerFactory and pandapower. The results are as follows.

EU grid: version 1

The grid model of version-1 consists of 171 PV buses, 8948 PQ buses and 233 slack buses. Merging the buses connected by closed switches leads to 9352 buses in the network and the order of the Jacobian is 18067. Applying the automating algorithm with preconditioner $\omega = 0.2$ leads to convergence in 6 iterations. Figure 9.4 shows the admittance matrix and convergence. Figure 9.5 shows the voltage magnitudes in the network. In most parts of the network, voltage magnitudes are beyond safe limits.

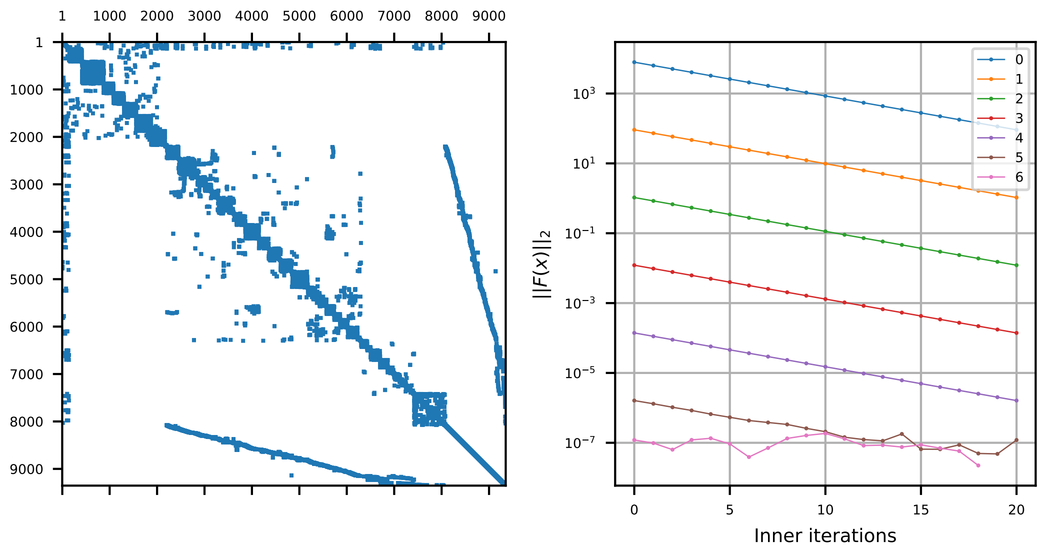


Figure 9.4: EU grid version-1: admittance matrix (left) and convergence (right).

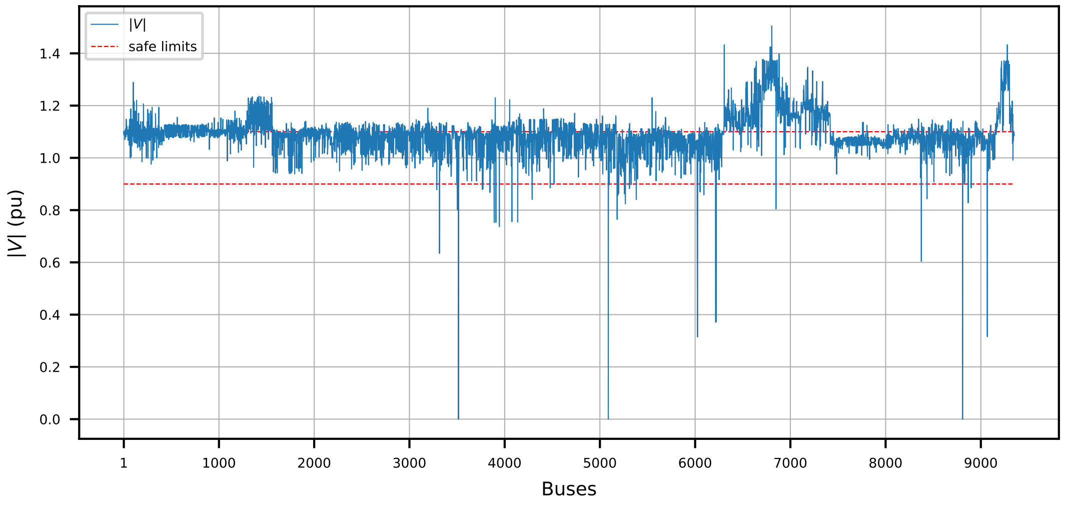


Figure 9.5: EU grid version-1: Voltage magnitudes.

Generally, year-round simulations are started with a basis model that is well-posed and has a feasible solution for at least one snapshot of load and generation data. The automating algorithm can then be used to fix convergence problems if any, during year-round simulations when load and generation data change for each hour. However, this example shows that the automating algorithm can also be used to find a solution if the basis model is ill-posed in the first place.

EU grid: version 2

The grid model of version-2 consists of 135 PV buses, 9247 PQ buses and 233 slack buses. Merging the buses connected by closed switches leads to 9795 buses in the network and the order of the Jacobian is 18989.

For this version, despite applying the automating algorithm with preconditioner $\omega = 0.2$, a solution has not been found. Figure 9.6 shows the admittance matrix and divergence. During the last few iterations of the automating algorithm, $\|\mathsf{F}(x)\|_2$ remains constant and does not meet the tolerance 10^{-8}. The same behavior has been observed even with other preconditioners $\omega \in (0, 1)$. We suspect that this version has modelling errors that are prohibiting the automating algorithm to find a solution. Hence, a detailed evaluation of the mismatch function $\mathsf{F}(x)$ is needed to identify modelling errors which need to be rectified before applying the automating algorithm. We exclude the exercise of finding modelling errors of this grid model in this project and recommend that they be fixed before doing year-round simulations with this version of the sub-European grid.

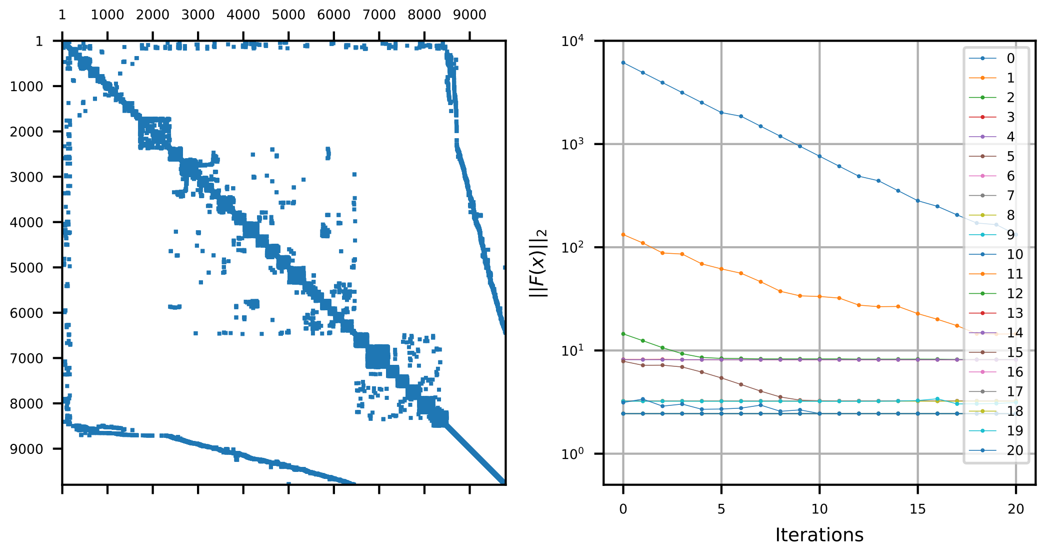


Figure 9.6: EU grid version-2: admittance matrix (left) and divergence (right).

In fact, the two versions of the EU grid considered in this project are supposed to correspond to two hours of a year during year-round simulations, in which case, as we know the topology of the grid model is not allowed to change. However, there is a significant difference between the topology of the two grid models, which is also evident from the admittance matrices. Hence, we suspect that the excel sheet template and the Python program (discussed in chapter 8) that generate load and generation data for year-round simulations in TenneT might also be changing the topology of the EU grid. Thus, solving this problem also requires an investigation of the Python scripts that distribute load and generation data among the generators and loads for year-round simulations, which is beyond the scope of this project.

Chapter 10

Conclusion

The objective of this project has been to develop an algorithm that can automate year-round AC power flow simulations for real-world grid models such as the Dutch grid model and the sub-European grid model that are used in TenneT for several applications like reactive power compensation assessment and contingency analysis. We conclude the following from our research, study and the execution of this project.

1. Power flow problems can be classified into well-conditioned, ill-conditioned and ill-posed problems. Based on this classification we develop the automating algorithm which can solve both well-posed and ill-posed power flow problems.

2. The Newton-Raphson method is the most distinguished AC power flow solver. In this project we use the polar power-mismatch version of the Newton-Raphson method as the basis solver for all our simulations and it has given satisfactory results. A well-conditioned power flow problem can be efficiently solved using the polar power-mismatch version of the Newton-Raphson method. All the well-conditioned power flow problems that we have solved in this project converge within 7 iterations. The linear system in each iteration of the Newton-Raphson method is solved using sparse LU decomposition.

3. We solve ill-conditioned power flow problems using the optimal multiplier method. The role of the optimal multiplier is to reduce the step size of the Newton-Raphson method so that the norm of the mismatch function is reduced in each iteration. The mathematical framework that we have developed in this project to solve ill-posed power flow problems is an extension of the optimal multiplier method.

4. An ill-posed power flow problem is regularized by making optimal changes to power injections. The left eigenvector associated with the unique zero eigenvalue of the Jacobian is used to navigate to the closest solvable operating point in the parameter space of the power flow problem. We compute zero eigenvalues using the shift and invert Arnoldi method.

5. In some cases, the optimal multiplier method fails to determine the minimum of the cost function in each iteration. For such problems we use a preconditioner to further reduce the step size and determine the minimum of the cost function. This method is found to perform exceptionally well in solving ill-posed power flow problems.

6. The shape of the boundary between the solvable region and the unsolvable region of the parameter space, and the distance of the desired operating point from the boundary play a very important role in convergence properties of the automating algorithm.

7. To make the automating algorithm suitable for practical applications, we apply constraints to the problem in such a way that power injections are allowed to change only at a set of predefined buses. Naturally, convergence of the automating algorithm slows down if constraints are defined. If constraints are unrealistic, the automating algorithm diverges. Hence, the constraints are to be carefully defined depending upon the application.

8. We use pandapower for power flow simulations. Since the grid models in TenneT are built in PowerFactory, we develop an interface in Python to convert grid models from PowerFactory to pandapower. We have successfully converted 10 grid models in this project for doing simulations.

9. The automating algorithm has been applied to 30 grid models (28 test grids and 2 real-world grids), 1751 contingencies and 9 snapshots of hourly load and generation data. This is a total of 1787 power flow problems (7 well-posed and 1780 ill-posed). Except for one power flow problem (one of the versions of the sub-European grid), the automating algorithm successfully solves the rest of the power flow problems considered in this project.

10. Our experiments indicate that if a grid model has a solution for at least one snapshot of load and generation data during year-round simulations, the automating algorithm very likely solves all the convergence problems during year-round simulations of that grid model. Even if there is no feasible solution for a grid model, the algorithm can still be used to solve it as we have demonstrated for one of the versions of the sub-European grid. If there is no solution, the results of the algorithm can be used to analyze the problem further and get a better insight into the problem, which could possibly lead to a solution.

Future Work

We recommend the following for taking this project ahead in the future.

1. The optimal multiplier method can be improved by using better step size optimization techniques such as line search. This could eliminate the need for a preconditioner, which at the moment is to be manually determined based on the problem and cannot be included in the automating subroutine.

2. In this project, to keep the model simple, voltage setpoints of generators are not included in the parameter space of the power flow problem. Including them could improve convergence of the automating algorithm. It could be easier to incorporate the voltage setpoints of generators in the parameter space and hence the automating algorithm if the polar current-mismatch version or the Cartesian-current mismatch version of the Newton-Raphson method (described in chapter 4) is used. This is because of the reason that the voltages of PV buses in these two solvers are not removed from the linear system (Jacobian).

3. Convergence of the automating algorithm can be improved by studying the geometry of the boundary between the solvable region and the unsolvable region of the parameter space in detail. This requires some more research about bifurcation theory and eigenvalue problems.

4. The automating algorithm finds a solution to ill-posed power flow problems by making optimal changes to power injections. However, there is no guarantee that the voltages are within safe limits. Keeping the voltages within safe limits requires performing optimal power flow simulations for the grid models. Optimal power flow involves determining the best control action to change the load and generation data in such a way that the voltages are within safe limits. This is a non-linear constrained optimization problem and can be solved using standard non-linear programming (NLP) solvers.

5. More constraints can be added to the automating algorithm such as power limits. For example, imposing a reactive power limit on a generator.

6. To scale this project further, the interface and the automating algorithm would have to be built into pandapower.

From the author

Thank you for showing interest in my project. I feel fortunate to have had the opportunity to work on a real-world problem and I hope that I have been able to produce a valuable end result. It has been a wonderful experience!

I would like to thank my supervisors Marieke, Martin, Kees and Jorrit for guiding me through, for trusting me and for always being kind and supportive. I am happy to have built the interface with Jasper van Casteren, who has been kind and encouraging right from the start.

I am honored to have made friends and memories for a lifetime in two wonderful years of COSSE. I am especially proud to have come this far despite the pandemic. I am grateful to my parents and my little sister for having my back, no matter what.

- Shravan

Bibliography

[1] Carl Sulzberger. Pearl street in miniature: Models of the electric generating station [history]. *IEEE Power and Energy Magazine*, 11(2):76–85, 2013.

[2] TenneT. Electricity Producers: The Netherlands and Germany.

[3] Pieter Schavemaker and Lou van der Sluis. *Electrical Power System Essentials*. John Wiley & Sons, Inc., Sussex, United Kingdom, 2008.

[4] Alexandra von Meier. *Electric Power Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2006.

[5] M.E. Kootte, J.E. Romate, and C. Vuik. Load Flow Computations for (Integrated) Transmission and Distribution Systems. A Literature Review. Technical report, Delft University of Technology, 2020.

[6] Fons van der Plas. *Power Grid Failures*. Thesis bsc mathematics, Radboud University Nijmegen, 2019.

[7] R. Idema, D.J.P. Lahaye, and C. Vuik. Load Flow Literature Survey. Technical report, Delft University of Technology, 2009.

[8] Brian Stott. Review of Load-Flow Calculation Methods. *Proceedings of the IEEE*, 62(7):916–929, 1974.

[9] Baljinnyam Sereeter, Cornelis Vuik, and Cees Witteveen. On a comparison of Newton–Raphson solvers for power flow problems. *Journal of Computational and Applied Mathematics*, 360:157–169, 2019.

[10] William. F. Tinney and Clifford E. Hart. Power Flow Solution by Newton's Method. *IEEE Transactions on Power Apparatus and Systems*, PAS-86(11):1449–1460, 1967.

[11] B. Stott and O. Alsac. Fast decoupled load flow. *IEEE Transactions on Power Apparatus and Systems*, PAS-93(3):859–869, 1974.

[12] Thomas J. Overbye, Xu Cheng, and Yan Sun. A comparison of the AC and DC power flow models for LMP calculations. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 37, pages 725–734, 2004.

[13] James L. Kirtley. *Electric power principles: sources, conversion, distribution, and use*. John Wiley & Sons, Ltd, 2010.

[14] DIgSILENT GmbH. PowerFactory User Manual. Technical report, 2020.

[15] SIEMENS. PSSE Program Operation Manual. Technical report, 2017.

[16] MATPOWER. User's Manual, version 7.1. Technical report, 2020.

[17] Medha Subramanian, Jan Viebahn, Simon H. Tindemans, Benjamin Donnot, and Antoine Marot. Exploring grid topology reconfiguration using a simple deep reinforcement learning approach. nov 2020.

[18] Henrik Ronellenfitsch, Marc Timme, and Dirk Witthaut. A Dual Method for Computing Power Transfer Distribution Factors. *IEEE Transactions on Power Systems*, 32(2):1007–1015, mar 2017.

[19] pandapower - Convenient Power System Modelling and Analysis based on PYPOWER and pandas. Technical report, Fraunhofer IWES, Universität Kassel, 2017.

[20] S. Iwamoto and Y. Tamura. A load flow calculation method for ill-conditioned power systems. *IEEE Transactions on Power Apparatus and Systems*, PAS-100(4):1736–1743, 1981.

[21] Patricia Rousseaux and Thierry Van Cutsem. Quasi steady-state simulation diagnosis using Newton method with optimal multiplier. In *2006 IEEE Power Engineering Society General Meeting, PES*. IEEE Computer Society, 2006.

[22] Thomas J. Overbye. A Power Flow Measure for Unsolvable Cases. *IEEE Transactions on Power Systems*, 9(3):1359–1365, 1994.

[23] Thomas J. Overbye. Computation of a practical method to restore power flow solvability. *IEEE Transactions on Power Systems*, 10(1):280–287, 1995.

[24] Ian Dobson and Liming Lu. New methods for computing a closest saddle node bifurcation and worst case load power margin for voltage collapse. *IEEE Transactions on Power Systems*, 8(3):905–913, 1993.

[25] Ian Dobson. Observations on the Geometry of Saddle Node Bifurcation and Voltage Collapse in Electrical Power Systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(3):240–243, 1992.

[26] Joseph Euzebe Tate and Thomas J. Overbye. A comparison of the optimal multiplier in polar and rectangular coordinates. *IEEE Transactions on Power Systems*, 20(4):1667–1674, nov 2005.

[27] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, jan 2011.

[28] R B Lehoucq, D C Sorensen, and C Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, 1998.

[29] Cédric Josz, Stéphane Fliscounakis, Jean Maeght, and Patrick Panciatici. AC Power Flow Data in MATPOWER and QCQP Format: iTesla, RTE Snapshots, and PEGASE. mar 2016.

[30] Konrad Purchala, Leonardo Meeus, Daniel Van Dommelen, and Ronnie Belmans. Usefulness of DC power flow for active power flow analysis. In *2005 IEEE Power Engineering Society General Meeting*, volume 1, pages 454–459, 2005.