Delft University of Technology
Faculty Electical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Power load flow models and their numerical solution

A thesis submitted to the
Delft Institute of Applied Mathematics
as partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Stan Jonker

Delft, Netherlands
July 2020

BSc report APPLIED MATHEMATICS

"Power load flow models and their numerical solution"

STAN JONKER

DELFT UNIVERSITY OF TECHNOLOGY

Supervisor

Dr. Domenico Lahaye

Other committee members

Prof. dr. ir. Mark Veraar

...

Dr. ir. Neil Budko

...

July, 2020

Delft

Preface

This report is on power load flows and the numerical methods to solve those. It has been written as my bachelor thesis for the bachelor Applied Mathematics at Delft University of Technology. I worked on this from February to July 2020.
This project was suggested by dr. Domenico Lahaye. I switched to this project because of another project involving hydrogen fuel generation I am involved in. Unfortunately, due to difficulties with obtaining the right data and the relatively short time frame, I was unable to really connect the two together. Regardless, I gained a lot of experience while working on this project.

First and foremost, I would like to thank dr. Domenico Lahaye for his support and guidance during this project. Next, I would like to thank Bertz Tourgoutian for helping me with questions regarding the real-life power grid, as well as all the people that connected me to him and gave me insights in the world around electrical power distribution. Lastly I would like to thank my friends and family, who helped me keep on track.

I hope you enjoy your reading.

Stan Jonker

Heemstede, July 4th, 2020

Contents

Chapter 1

Introduction

Our daily lives depend heavily on electricity and electrical appliances. From refrigerators to lighting to computers to the machines in hospitals, as a society we consistently need a lot of it. As such, it is important that we generate enough power and that we distribute it well. Distribution of electrical power is done with a power grid, a network of electrical cables and stations connected throughout the land. However, the power demand is not always the same everywhere. For example, whenever someone puts their oven on, they briefly increase their demand of electricity. Yet no one has their oven on all day. This becomes more important when the demand increases a lot more, such as when a factory turns on huge machinery or when a ship docks and draws power from the grid rather than its motor. An example to make this more clear can be found in 1.1. It may also be that many points in the grid increase their demand. In the United Kingdom, electricity companies noticed that the demand went up whenever there was an ad break on the most popular TV channels, because that is when most people went up to turn their boilers on to make tea. This phenomenon is now referred to as 'TV pickup'.

To make sure that everybody has power, we may make a model of this grid to try to see how we can best set up our network and try to predict what happens when demand is increased or lowered in certain points.

In this report, we firstly look at such a model and the laws that govern it. From this, we derive a set of equations, after which we discuss 3 algorithms capable of solving those equations, the Newton-Raphson method, the line-search method and the trust region method. In the literature, the former is mostly used. However, due to an increase in different elements to the power grid

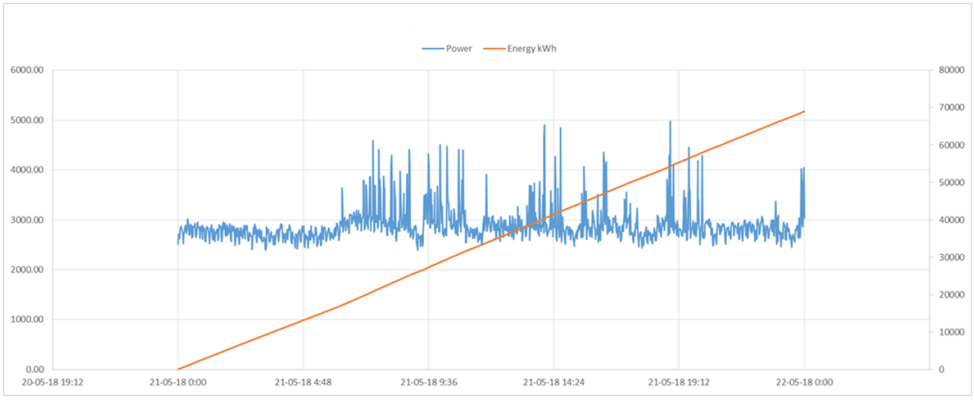


Figure 1.1: An example of a so-called load profile, which (in blue) shows the power demand at various times per day (in kilowatts). While the average power consumption is relatively stable, there are some clear peaks. This image was provided by Bertz Tourgoutian, electrical project engineer with Eneco.

such as huge batteries and great power loads, that might not be sufficient. For that reason, we introduce two methods that converge globally. Lastly, we apply our model and solving methods to small examples to see what kind of results they give and the behaviour of the solving methods. We also compare the 'basic' methods we describe here to a package of minimisation tools where many people have spent many days on. Ideally, that would have been done with real-life examples. However, this information is difficult to procure due to privacy and other concerns. To find the code used in this project, visit https://github.com/Stan−Jonker/BEP.

To read this report, a basic knowledge of complex numbers is required, as well as knowledge of multi-variable calculus.

Chapter 2

The power flow model

To start, we examine the model of a power system. For this, we firstly describe quantities that one might find in such a system and then state the natural laws that give their relations. Under certain assumptions, we may then derive our model and the equations that will allow us to fill in all the quantities. This section is based upon Computational methods in Power System Analysis[1] , Power System Analysis[2] and Fundamentals of Electric Circuits[3] .

2.1 Voltage and current

Current is a measure of a flow of electrical charge carriers, usually electrons. We denote the current by i, with amperes (A) as a unit. **Voltage** is a measure of the work required to move these carriers in a particular direction. We denote voltage as v, with volts (V).
Usually with power lines there are only two directions current can flow in. If the current always flows in the same direction, we call this **direct current (DC)**. This is often seen in circuits with a battery as power source. If the current switches direction over time, we call it **alternating current (AC)**. This type of current is what comes from our outlets and is generated by electric generators, which convert mechanical energy into electrical power. The source for the mechanical energy often consists of a form of turbine. The turbines use some rotating part, that rotates a powerful magnet between two (or more) coils of wire. The rotation of the magnet and thus of the magnetic field induces a current in the coils. But, as the magnet rotates, the direction of the current does too, which is where the *alternating* in alternating current comes from. As the current is induced by a rotating part, it described very well by a sinusoid, i.e. $i(t) = I_{\max}\cos(\omega t)$. We should note that the voltage is of the same type as the current: if the current is alternating, so is the voltage and if we have direct current then the voltage is also in just one direction. However, due to convention, we do not call this 'AV' or 'DV' but still refer to this as AC or DC.

To describe the power flow model, we use AC. In this case, we define:

$$v = v(t) = V_{\max}\cos(\omega t + \delta_V) = \Re\left(V_{\max}e^{\mathrm{i}\delta_V}e^{\mathrm{i}\omega t}\right) \tag{2.1}$$

$$i = i(t) = I_{\max}\cos(\omega t + \delta_I) = \Re\left(I_{\max}e^{\mathrm{i}\delta_I}e^{\mathrm{i}\omega t}\right) \tag{2.2}$$

where i is the imaginary unit ($\mathrm{i}^2 = -1$) and $\Re$ is the operator that takes the real part. We focus on the complex representations. Here, ω is the frequency of the system, which we assume to be constant. Because this is the same everywhere, the term $e^{\mathrm{i}\omega t}$ is not necessary to describe the voltage and current. The remaining terms $V_{\max}e^{\mathrm{i}\delta_V}$ and $I_{\max}e^{\mathrm{i}\delta_I}$ are independent of time and used to represent the voltage. Here, $V_{\max}$ and $I_{\max}$ are the amplitudes of the voltage and the current and δ_V and δ_I are voltage and current angles. These describe the phase differences of the sinusoids. We shall discuss these later.

In power system theory, we use the effective phasor representation:

$$V = |V|e^{\mathrm{i}\delta_V}, \quad \text{with } |V| = \frac{V_{\max}}{\sqrt{2}} \tag{2.3}$$

$$I = |I|e^{\mathrm{i}\delta_I}, \quad \text{with } |I| = \frac{I_{\max}}{\sqrt{2}} \tag{2.4}$$

This representation is sometimes also called the frequency domain representation, as the frequency is the same everywhere and hence suppressed. Throughout the rest of this document, any variable V, I, V_j or I_j shall refer to the effective phasor representation of the voltage and current. Furthermore, if we refer to the voltage as 50000 kV for example, then this means that $|V| = 5 \cdot 10^7$.

2.2 Complex power

We now want to use the voltage and current as defined above to compute the **instantaneous power**. This is a measure of expending or absorbing energy, measured in watts (W). Before we do this, we firstly pick a reference time where the voltage can be written as $v(t) = V_{\max}\cos(\omega t)$. Defining $\phi = \delta_V - \delta_I$ then gives us that the current is written as $i(t) = I_{\max}\cos(\omega t + \phi)$. We shall discuss the quantity ϕ later on.

Now we may compute the instantaneous power $p(t)$:

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

Now defining $P = |V||I|\cos\phi, Q = |V||I|\sin\phi$ gives:

$$p(t) = P[1 + \cos(2\omega t)] + Q[\sin(2\omega t)] \tag{2.5}$$

P is also called the **active or average**[1] **power**, with Watts (W) as unit. Q is called the **reactive or imaginary power**, measured in Volt-Ampéres reactive (VAr). In fact Volt-Ampére reactive is the same unit as the Watt, but the distinction is made to really keep the active and reactive power separate.

Note that in fact

$$P = \Re(V\overline{I}), \quad Q = \Im(V\overline{I}) \tag{2.6}$$

Hence, it would make sense for us to define the **complex power** S of an AC circuit by:

$$S = P + \mathrm{i}Q = V\overline{I} \tag{2.7}$$

If we compute S from V and I, we can use equation (2.5) to obtain the instantaneous power as a function of time. S is sometimes also referred to as the apparent power.

[1] The reason P is called the average power is because for each period $T = \frac{2\pi}{2\omega}$ of $p(t)$, the average instantaneous power over that period is given by $\frac{1}{T}\int_0^T p(t)\,dt = P$. Furthermore, from this we may deduce that in fact $\lim_{s\to\infty}\frac{1}{s}\int_0^s p(t)\,dt = P$.

2.3 Power factor

In the previous section we have briefly seen $\phi = \delta_V - \delta_I$. This quantity is called the **power phase angle**. Is gives rise to one of the most important quantities for a network operator, namely the **power factor** given by $\cos\phi$. The reason this quantity is so important is because it denotes the ratio between the active and reactive power. The active power is the part of the total power that does almost all of the work, which is why we usually want a relatively high part of the total power to be active power. Nonetheless, reactive power is sometimes useful or even required. This is mostly the case in large electrical motors. However, too much reactive power leads to undesirable effects such as voltage drops, equipment heating and reduced active power delivery. While this might not be a problem for the average household, large energy consumers such as skyscrapers or stadia will often be charged more on their energy bill if they have a (relatively) small power factor.

2.4 Lumped elements model

Throughout this document we will make a few simplifications for the elements in the circuits we discuss. For this, we use the lumped elements model.
In reality, as we have seen, power is a flow of electrical charge carriers. These particles flow through power lines, which consist of many atoms and molecules. Obviously, we do not want to simulate all interactions between our carriers and those atoms. For this reason, we treat all those atoms as a single lumped element that behaves as a single entity. What remains is a network where all elements are seen as idealized electrical components connected through perfectly conducting wires.
There are some assumptions we have to make for this. The first is that there is no change in the magnetic flux outside conducting elements. The second is that throughout any conducting elements, we do not lose or gain any charge. Lastly we must assume that the delay caused by propagation of electromagnetic waves is negligible in the time scale we are working with.

However, we can make some more simplifications. For example, places where power is generated usually contain multiple generators. However, all this power gets combined later when it gets put into the power network. Seeing as this is the case, we do not want to consider all possible generators but rather combine them into one. The same holds for, as an example, entire neighbourhoods. We do not want to consider every household separately but rather take them all together. An additional benefit of doing this is that the power consumption of a single household can vary wildly, but taking all households together gives a reasonably steady power consumption. Another reason we do this is because otherwise the network we work with become unmanageably large.

2.5 Circuit elements

Now that we have defined v and i and used them to define the complex power, we now want to find the relation between them. For this, we use Ohm's laws.
We begin with discussing resistors. A resistor is a physical passive element that adds electrical resistance to a circuit to reduce the flow of current.

Electrical resistance R (in ohms Ω) is a measure of opposition the the flow of electric current [2]. If assume the current is of the form $i(t) = I_{\max}\cos(\omega t + \delta_I)$, then the voltage across a resistor R is given by

$$v(t) = Ri(t) \tag{2.8}$$

Or, in (effective) phasor representation

$$V = RI \tag{2.9}$$

[2] It should be noted that the resistor is also often referred to as R.

Note that by this computation, the current and voltage are actually both in phase (i.e. $\delta_V = \delta_I$).

Figure 2.1: A resistor.

Equation (2.8) is in fact the same for both AC and DC systems. However, with AC systems another opposition of current plays a crucial role which does not exist in DC systems. Due to the fact that the current and voltage are fluctuating, we must also consider the reactance. The **reactance** X is a measure of the opposition of a circuit element to a change in electric current or voltage. There are two types of reactance. The first is the **inductance**. This is a phenomenon where an inductor, which is typically a coil of wire, opposes a change of current flowing through it. If we are given the value of the inductor, denoted by L, then the relation between v and i is given by

$$v(t) = L\frac{di}{dt}(t) = -\omega L I_{\max} \sin(\omega t + \delta_I) \tag{2.10}$$

Transforming this into phasor representation, we get

$$V = \mathrm{i}\omega L I \tag{2.11}$$

This computation shows that the voltage and current are 90°or $\frac{\pi}{2}$ rad out of phase, meaning that the power factor is 0 for just an inductor.

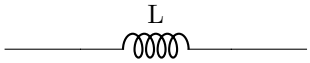


Figure 2.2: An inductance in a circuit.

The second type of reactance is the **capacitance**. A capacitor is a unit consisting of two conducting plates separated by an insulator. The ratio between the charge on one plate of the capacitor and the voltage difference between the two plates is then called the capacitance, denoted by C and measured in farad (F)[3].

Putting that into an equation, we find that

$$q(t) = Cv(t) \tag{2.12}$$

where $q(t)$ denotes the charge of the plate. Now note that i is the flow of charge, so $\frac{d}{dt}q(t) = i(t)$. By differentiating equation (2.12), we obtain

$$i(t) = C\frac{dv}{dt}(t) \tag{2.13}$$

Transforming this equation into phasor representation nets us the following relation

$$V = \frac{I}{\mathrm{i}\omega C} \tag{2.14}$$

Now the current leads the voltage by 90°.

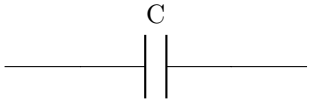


Figure 2.3: A capacitance in a circuit.

[3]Due to the properties of transmission lines, nanofarad is a more common unit.

Most lines have both a inductive and capacitive reactance. For this, we may define the reactance as $X = \omega L - \frac{1}{\omega C}$.
In reality, most elements have a combination of resistance and reactance. For this, we define the **impedance**

$$Z = R + \mathrm{i}X \tag{2.15}$$

which is a measure of opposition to a sinusoidal current. For our purposes, however, we are more interested in the inverse of the impedance, called the **admittance** Y (in siemens S):

$$Y = \frac{1}{Z} = \frac{R}{|Z|^2} - \mathrm{i}\frac{X}{|Z|^2} = G + \mathrm{i}B \tag{2.16}$$

Here G is called the **conductance** and B is called the **susceptance**. Using Ohm's law extended to AC circuits, we find that the relation between V and I is given by

$$V = ZI, \quad \text{or } I = YV \tag{2.17}$$

2.6 Kirchoff's circuit laws

For a lumped element model, we may use two equalities known as Kirchhoff's circuit laws. These equalities were described by Gustav Kirchhoff in 1845 and can be derived from Maxwell's equations under the assumptions of a steady state and the lumped elements model.

- **Kirchhoff's voltage law:** Suppose that we have a closed loop in a circuit with $v_1, v_2, \ldots, v_n$ the voltages on that loop[4]. Then Kirchhoff's voltage law states that $\sum_{i=1}^{n} v_i = 0$. If we write each $v_i = V_{\max,i}\cos(\omega t + \theta_i)$ and $V_i = \frac{V_{\max,i}}{\sqrt{2}} e^{\mathrm{i}\theta_i}$, then from $0 = \sum_{i=1}^{n} v_i = \sum_{i=1}^{n} V_{\max,i}\cos(\omega t + \theta_i) = \Re\left[\left(\sqrt{2}\sum_{i=1}^{n} V_i\right) e^{\mathrm{i}\omega t}\right]$ we may conclude[5] that also in the effective phasor domain the voltage law holds: $\sum_{i=1}^{n} V_i = 0$.
- **Kirchhoff's current law:** If $i_1, i_2, \ldots i_n$ are the currents coming in[6] at a point in the network, then $\sum_{j=1}^{n} i_j = 0$. Using a similar computation as above, we may find that $\sum_{j=1}^{n} I_j = 0$ also holds in the effective phasor representation.

2.7 Nodal analysis

Before the previous section we have only discussed some basic concepts on single elements. We now want to go further in depth about how we actually analyse an entire circuit.
We have up to this point seen a voltage as the voltage over an element. However, we want to use a different kind: nodal voltage. Nodal analysis is a general procedure that uses node voltages as circuit variables to analyse circuits. Using nodal analysis reduces the number of equations we have to solve later on.
We index each node with $1, 2, \ldots, n$ and assign each a nodal voltage $v_1, v_2 \ldots, v_n$ To obtain node voltages, we firstly have to select a node as a reference node, usually node 1 [7]. In general, at the reference node we have $v_1 = 0$. All other $n - 1$ nodes will have nodal voltages with respect to the reference node, which we may obtain by applying Kirchhoff's circuit laws to each of the non-reference nodes.
If there is a voltage source connected between the reference node and a non-reference node, we set the nodal voltage of the non-reference node to the voltage of the source. If there is a voltage source between two non-reference nodes, we shrink those two together into a 'generalised node' or 'supernode', and apply Kirchhoff's laws to that node. Then if we substitute $v_{ab} = v_a - v_b$ for

[4] We emphasize here that voltage has a direction. If a voltage has a direction not according to the loop (i.e. the other way around), it is represented as a negative value.

[5] As $e^{i\omega t} \neq 0$ and the fact that $\Re\left[\left(\sqrt{2}\sum_{i=1}^{n} V_i\right) e^{\mathrm{i}\omega t}\right] = 0$ must always hold.

[6] Again note that all currents going out are represented negatively.

[7] Obviously by interchanging the labels we may call every node "node 1"

the voltage over the element between node a and b, we may create a matrix involving only the admittances and the nodal voltages.

Note that this also works for the effective phasor representation. Fully worked out examples can be found in Fundamentals of Electric Circuits[3] .

2.8 Power system model

Now that we have discussed some concepts of electrical engineering, we use this to build a model of a power system. We model a power system as a network of nodes (called buses) and edges (called branches), where at each bus there are four electrical quantities we either know or want to compute. For each bus i:

- $|V_i|$: the voltage magnitude,
- δ_i: the voltage phase angle,
- P_i: injected active power,
- Q_i: injected reactive power.

We characterize each bus by which of the above quantities are known and which are unknown. These can be found in the following table.

Bus type	Short explanation	Known	Unknown
Load/PQ bus	Active and reactive power get injected into the network here.	P_i, Q_i	$\|V_i\|, \delta_i$
Generator/PV bus	Generates active power, the voltage magnitude may be set.	$P_i, \|V_i\|$	Q_i, δ_i
Slack/swing bus	Models system loss.	$\delta_i, \|V_i\|$	P_i, Q_i

Figure 2.4: A table of different bus types.

A load bus is a bus with a load attached, which is a demand of active power. It is modelled as a bus where a injected active power P and injected reactive power Q are known. Often, the active power a load requires is based upon historical data, predictions and/or measurements (such as for example figure 1.1). In practice the reactive power is not always known but based on assumptions of the power factor, which is expected to be 0.85 or higher.

At a physical generator, it is often possible to control the active power it generates and the voltage magnitude. As such, we presume that these values are known. The reactive power a generator needs to generate to support the voltage magnitude cannot be known in advance an can therefore not be specified, which also means that the voltage angle is an unknown quantity. However, it is not always the case that a generator has voltage magnitude controls. These are then modelled as loads with positive injected active power.
In almost every system there are power losses, which we also have to model. We do this by assigning a generator node that, in addition to its normal generation, also has to compensate for the difference between total generation and total consumption. The bus we assign for this is called the slack bus or swing bus. Obviously, for this generator we cannot specify P and Q. However, we can specify the voltage magnitude and the voltage phase angle. Recall that the voltage phase angle is given as a reference to some node. Since this reference node is not yet chosen, we may pick the slack bus for this. While it does not necessarily matter which phase angle is chosen, it is common to set it to 0 for the slack bus.

2.9 Shunts and transformers

Before we go on to properly define the power flow equations, we consider two additional parts of a circuit. Firstly, we examine how to model (electical) shunts. Shunts are circuit elements that make it possible for current to travel between two points in a circuit via a path with low resistance. It is used to adjust the voltage level by adding or taking away reactive power. Shunt capacitors may be used to inject reactive power, which lowers the node voltage, and shunt inductors may do the opposite.
Secondly, let us examine the transformer. Transformers can be put between two points with different voltage levels. This is necessary between, as an example, the high-voltage ($\geq$ 110kV) transmission grid and the distribution grid ($\sim$50kV).

Because a shunt has almost no resistance, it is modelled as just its reactance, i.e. $Z_s = \mathfrak{i}X_s$. Hence, the shunt admittance is given by $Y_s = \frac{1}{Z_s} = -\mathfrak{i}\frac{X_s}{X_s^2} = \mathfrak{i}B_s$. A shunt is modelled by giving both points an equal amount of the admittance.

Transformers have a transformer ratio $T : 1$, where $|T|$ determines the change in voltage magnitude and $\arg(T)$ determines the shift in voltage phase angle.

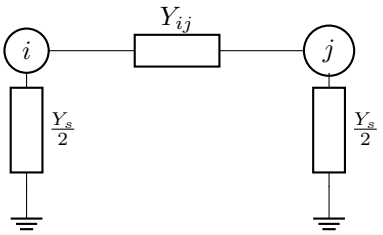


Figure 2.5: Model of a transmission line with a shunt.

Figure 2.6: Model of a Transformer.

2.10 Power flow equations

We now want to use all previous sections to derive the power flow equations. For this, we firstly introduce some notation. First, we introduce the admittance matrix $\mathbf{Y}$. If we take $\mathbf{I} = (I_1, \ldots, I_N)^T$ and $\mathbf{V} = (V_1, \ldots, V_N)^T$, then they are related through

$$\mathbf{I} = \mathbf{YV} \tag{2.18}$$

Which would give us that $I_i = \sum_{k=1}^{N} \mathbf{Y}_{ik} V_k$. As of yet, we still need to construct this matrix $\mathbf{Y}$. For this, firstly note that from Kirchhoff's equations

$$I_i = \sum_{k=1}^{N} I_{ij} \tag{2.19}$$

Recall from Ohm's equations that the current over the line between nodes i and j, $i \neq j$, is given by

$$I_{ij} = Y_{ij}(V_i - V_j), \quad I_{ji} = -I_{ij} \tag{2.20}$$

In matrix notation, this may be given by the equation

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = Y_{ij} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix} \tag{2.21}$$

Adding these all together according to (2.19), we would get that $\mathbf{Y}_{ii} = \sum_{\substack{k=1 \\ k \neq i}}^{N} Y_{ij}$ and $\mathbf{Y}_{ij} = -Y_{ij}$ if $j \neq i$. However, we have now only taken the lines into consideration. We still need to add shunts and transformers. Shunts may be connected to a node itself or to a line. In the first case, we add the shunt admittance Y_s to the admittance $\mathbf{Y}_{ii}$. In case the shunt is connected to a line between i and j, it is modelled by adding half of the shunt admittance to both buses (so $\frac{Y_s}{2}$) to both $\mathbf{Y}_{ii}$ and $\mathbf{Y}_{jj}$. We find

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = \left(Y_{ij} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \frac{Y_s}{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} V_i \\ V_j \end{bmatrix} \tag{2.22}$$

Up to this point, we can summarize the matrix by

$$\mathbf{Y}_{ij} = \begin{cases} Y_i + \sum_{\substack{k=1 \\ k \neq i}}^{N} Y_{ik} & i = j \\ -Y_{ij} & i \neq j \end{cases} \tag{2.23}$$

Adding transformers is a little more difficult. Suppose that E is the voltage induced by the transformer. Then $V_i = TE$, where T is the transformer ratio. The current flowing from bus j to bus i is given by

$$I_{ji} = Y_{ij}(V_j - E) = Y_{ij} \left(V_j - \frac{V_i}{T} \right) \tag{2.24}$$

Now by conservation of power within the transformer we obtain

$$V_i \overline{I_{ij}} = -E \overline{I_{ji}} \tag{2.25}$$

From which we can find

$$I_{ij} = -\frac{I_{ji}}{\overline{T}} = Y_{ij} \left(\frac{V_i}{|T|^2} - \frac{V_j}{\overline{T}} \right) \tag{2.26}$$

The above can be summarised in matrix form:

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = \left(Y_{ij} \begin{bmatrix} \frac{1}{|T|^2} & -\frac{1}{\overline{T}} \\ -\frac{1}{\overline{T}} & 1 \end{bmatrix} + \frac{Y_s}{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} V_i \\ V_j \end{bmatrix} \tag{2.27}$$

Where $T = 1$ if there is no transformer in the line. If there is a transformer, we may now modify the corresponding entries in equation (2.23) according to the above equation (2.27).

We know that the total complex power is given by $S_i = V_i \overline{I_i}$. By the above, we can now write this as

$$S_i = V_i \overline{I_i} = V_i \sum_{k=1}^{N} \overline{Y_{ik} V_k} \tag{2.28}$$

for every bus i. These are known as the power flow equations.

2.11 Per unit

Electrical grids are often split up in multiple parts with different base voltage levels, separated by transformers. Rather than specifying the actual voltage levels of the nodes, it is common to refer to those voltages as the fraction or *per unit* of the base voltage level. For example, in a 50 kV part of the grid, 45 kV becomes 0.9 per unit. Note that these values apply to the voltage magnitude, not to the voltage angle. Per unit is abbreviated as p.u..
We do not only want to convert the voltage magnitude into per unit, but all other variables as

well. A system in which this is the case is called a per-unit sytem. For this, we need to chose a base for another quantity, for example P or S. Usually, S is given a base value. All other base values are then computed by $S = V\bar{I}$, $P = |S|\cos\phi, Q = |S|\sin\phi$ and $I = YV$. Choosing V_{base} and S_{base} as base values for V and S, we obtain:

$$\begin{aligned} I_{\text{base}} &= \frac{S_{\text{base}}}{V_{\text{base}}} \\ P_{\text{base}} &= S_{\text{base}}\cos\phi \\ Q_{\text{base}} &= S_{\text{base}}\sin\phi \\ Y_{\text{base}} &= \frac{I_{\text{base}}}{V_{\text{base}}} = \frac{S_{\text{base}}}{V_{\text{base}}^2} \end{aligned}$$

There are a few reasons to use a per-unit system. The first comes from the engineers building and using the power grids. In a per-unit system it is easier to spot, as an example, a large voltage drop. For this, one just has to look at where the voltage is low per unit, whereas normally it would be required to compare the voltage in a point compared to the base level of the grid. The second, which is more useful for our purposes, is that it makes it easier to solve the power flow equations numerically.

Chapter 3

Power mismatch function

In this chapter, we want to give a setup for solving the power flow equations (2.28). To do this, we transform the problem into a root-finding problem by a function referred to as the power mismatch function. We also discuss the Jacobian of this function and how to modify it when we take different bus types into account.

3.1 Power mismatch function

In the previous section, we defined the power flow equations, for each bus i given by

$$S_i = \sum_{k=1}^{N} V_i \overline{Y_{ik} V_k} \tag{3.1}$$

It is, unfortunately, quite difficult to solve these equations in their current form. The fact that this equation consists of only complex numbers plays a big part in the difficulty. For this reason, we want to do some work on the equations to see if it is possible split the above equation into its real and imaginary part. Luckily, we can substitute the original forms of $V_i = |V_i|e^{\mathrm{i}\delta_i}$ and $Y_{ik} = G_{ik} + \mathrm{i}B_{ik}$ back to obtain

$$S_i = \sum_{k=1}^{N} |V_i| e^{\mathrm{i}\delta_i} \left(G_{ik} - \mathrm{i}B_{ik}\right) |V_k| e^{-\mathrm{i}\delta_k} \tag{3.2}$$

$$= \sum_{k=1}^{N} |V_i||V_k| e^{\mathrm{i}\delta_{ik}} (G_{ik} - \mathrm{i}B_{ik}) \tag{3.3}$$

where $\delta_{ik} = \delta_i - \delta_k$.

Now recall Euler's identity $e^{\mathrm{i}\delta_{ik}} = \cos(\delta_{ik}) + \mathrm{i}\sin(\delta_{ik})$. Substituting that in results in

$$S_i = \sum_{k=1}^{N} |V_i||V_k| \left[(G_{ik}\cos(\delta_{ik}) + B_{ik}\sin(\delta_{ik})) + \mathrm{i}(G_{ik}\sin(\delta_{ik}) - B_{ik}\cos(\delta_{ik}))\right] \tag{3.4}$$

Now define

$$\mathbf{x} = (\delta_1, \ldots, \delta_N, |V_1|, \ldots, |V_N|)^T \tag{3.5}$$

We may split the above equation into its real and imaginary parts $P_i(\mathbf{x})$ and $Q_i(\mathbf{x})$, given by

$$\mathcal{P}_i(\mathbf{x}) = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\cos(\delta_{ik}) + B_{ik}\sin(\delta_{ik}))$$

$$\mathcal{Q}_i(\mathbf{x}) = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\sin(\delta_{ik}) - B_{ik}\cos(\delta_{ik}))$$

Now by (3.4), we must have that $P_i + \mathrm{i}Q_i = S_i = \mathcal{P}_i(\mathbf{x}) + \mathrm{i}\mathcal{Q}_i(\mathbf{x})$. So we must have that $P_i = \mathcal{P}_i(\mathbf{x})$ and $Q_i = \mathcal{Q}_i(\mathbf{x})$ for all buses i. Setting $\mathbf{P} = (P_1, \ldots, P_N)^T, \boldsymbol{\mathcal{P}}(\mathbf{x}) = (P_1(\mathbf{x}), \ldots, P_N(\mathbf{x}))^T, \mathbf{Q} = (Q_1, \ldots, Q_N)^T$ and $\boldsymbol{\mathcal{Q}}(\mathbf{x}) = (Q_1(\mathbf{x}), \ldots, Q_N(\mathbf{x}))^T$, we define the power mismatch function

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \mathbf{P} - \boldsymbol{\mathcal{P}}(\boldsymbol{x}) \\ \mathbf{Q} - \boldsymbol{\mathcal{Q}}(\boldsymbol{x}) \end{bmatrix} \tag{3.6}$$

which is a real-valued vector function. This is useful later on when we want to solve the equation

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{3.7}$$

which is, in fact, the power flow problem written as a root-finding problem. The reason we have done this is because there are multiple algorithms to help us find the root of the power mismatch function.

All algorithms we will discuss are iterative root-finding processes or iteration schemes. These schemes start with some initial guess $\mathbf{x}_0$ for the vector $\mathbf{x}^\star$ that satisfies $\mathbf{F}(\mathbf{x}^\star) = \mathbf{0}$. Then at each step i, we try to use the current iterate $\mathbf{x}_i$ to find some update vector $\mathbf{s}_i$, after which we make the new iterate $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{s}_i$. This process is continued until either $||\mathbf{F}(\mathbf{x}_i)|| < \varepsilon$ for some pre-specified value of ε, the iteration scheme diverges or until we have reached a maximum number of iteration steps. This last stopping condition is in place in the case that the scheme does not converge fast enough or at all, in which case we do not want to keep doing the scheme forever.

3.2 Treating different bus types

The power mismatch function (3.6) was constructed with the underlying assumption that all entries in $\mathbf{P}$ and $\mathbf{Q}$ are known and all values in $\mathbf{x}$ unknown. However, this is not the case. Recall table 2.4. Generators do not have a specified Q_i but do have a specified $|V_i|$, and for slack buses both the P_i and Q_i are unknown whereas the δ_i and $|V_i|$ are known.
If any P_i or Q_i is not known, we can quite easily just remove the entry corresponding to that quantity from the power mismatch function. In that case, we may simply try to find the root of the power mismatch function and compute their values from $\mathcal{P}_i(\mathbf{x})$ or $\mathcal{Q}_i(\mathbf{x})$.
Handling the entries in the vector $\mathbf{x}$ that are known is somewhat more difficult. Recall that we work with iteration schemes, for which we have an update vector $\mathbf{s}_i$ such that $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{s}_i$. If we, as an example, have a δ_j that is known, then the best corresponding entry for $\mathbf{s}_i$ is 0. To make sure this happens, we may simply drop these entries from the update vector and modify our scheme to reflect this change.

In total, we remove $N_G + 2$ equations and entries of $\mathbf{x}$, where N_G is the number of generators. This can be seen from the fact that for generators Q_i is unknown, and for the slack bus the P_s and Q_s are unknown, which amounts to $N_G + 2$ unknowns. Similarly, for generators the $|V_i|$ is known and both δ_s and $|V_s|$ is known for the slack bus, resulting in the same number of removed entries of $\mathbf{x}$. Since this is the case there are now as many equations as there are variables, namely $2N - N_G - 2$, which should allow for the system to be solved. An additional benefit of reducing the system is that we also reduce the amount of linear equations that need to be solved, which should speed up our numerical schemes. We shall refer to the reduced power mismatch function as $\hat{\mathbf{F}}(\hat{\mathbf{x}})$. Going on, we shall not specify if we are talking about $\mathbf{F}$ or $\hat{\mathbf{F}}$ unless stated otherwise. For the analysis this will not matter.

It should be noted that there are other methods for dealing with different bus types, but we shall not discuss these.

3.3 Jacobian of the power mismatch function

As we will see in the next section, for each iteration scheme we need to compute the Jacobian matrix once per iteration. For this reason, we want to discuss what this matrix looks like. We get that the Jacobian (see also equation (4.2)) of the power mismatch (3.6) function is given by:

$$J_{\mathbf{F}} = -\left[\begin{array}{ccc|ccc} \frac{\partial \mathcal{P}_1}{\partial \delta_1} & \cdots & \frac{\partial \mathcal{P}_1}{\partial \delta_N} & \frac{\partial \mathcal{P}_1}{\partial |V_1|} & \cdots & \frac{\partial \mathcal{P}_1}{\partial |V_N|} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{P}_N}{\partial \delta_1} & \cdots & \frac{\partial \mathcal{P}_N}{\partial \delta_N} & \frac{\partial \mathcal{P}_N}{\partial |V_1|} & \cdots & \frac{\partial \mathcal{P}_N}{\partial |V_N|} \\ \hline \frac{\partial \mathcal{Q}_1}{\partial \delta_1} & \cdots & \frac{\partial \mathcal{Q}_1}{\partial \delta_N} & \frac{\partial \mathcal{Q}_1}{\partial |V_1|} & \cdots & \frac{\partial \mathcal{Q}_1}{\partial |V_N|} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{Q}_N}{\partial \delta_1} & \cdots & \frac{\partial \mathcal{Q}_N}{\partial \delta_N} & \frac{\partial \mathcal{Q}_N}{\partial |V_1|} & \cdots & \frac{\partial \mathcal{Q}_N}{\partial |V_N|} \end{array}\right](\mathbf{x}) \tag{3.8}$$

For the Jacobian of the reduced power mismatch function $\hat{\mathbf{F}}$ as discussed in section 3.2, we may take the above matrix and remove a corresponding row for each removed P_i or Q_i in $\hat{\mathbf{F}}$ and a corresponding column for each removed δ_i or $|V_i|$ from $\mathbf{x}$ to get $\hat{\mathbf{x}}$. With proper ordering, it is possible to obtain the Jacobian of $\hat{\mathbf{F}}$ by splicing the Jacobian of $\mathbf{F}$.

Recall the expressions for $\mathcal{P}_i(\mathbf{x}) = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\cos(\delta_{ik}) + B_{ik}\sin(\delta_{ik}))$ and $\mathcal{Q}_i(\mathbf{x}) = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\sin(\delta_{ik}) - B_{ik}\cos(\delta_{ik}))$. It should be quite easy to compute all derivatives, the results are given below. However, when doing this one might notice that all derivatives are made up form the same 'building blocks' as the $\mathcal{P}_i$ and $\mathcal{Q}_i$ are. Since every time the Jacobian needs to be evaluated, the power mismatch function is also evaluated, linking their evaluation together provides a possible shortcut to speed up the evaluation process. We give the elements of the Jacobian here.

Define $\mathcal{P}_{ij}(\mathbf{x}) = |V_i||V_j|\,(G_{ij}\cos(\delta_{ij}) + B_{ij}\sin(\delta_{ij}))$ and $\mathcal{Q}_{ij} = |V_i||V_j|\,(G_{ij}\sin(\delta_{ij}) - B_{ij}\cos(\delta_{ij}))$[1]. The off-diagonal elements are given by $(i \neq j)$

$$\frac{\partial \mathcal{P}_i}{\partial \delta_j}(\mathbf{x}) = \mathcal{Q}_{ij}(\mathbf{x}) \qquad \frac{\partial \mathcal{Q}_i}{\partial \delta_j}(\mathbf{x}) = -\mathcal{P}_{ij}(\mathbf{x})$$
$$\frac{\partial \mathcal{P}_i}{\partial |V_j|}(\mathbf{x}) = \frac{\mathcal{P}_{ij}(\mathbf{x})}{|V_j|} \qquad \frac{\partial \mathcal{Q}_i}{\partial |V_j|}(\mathbf{x}) = \frac{\mathcal{Q}_{ij}(\mathbf{x})}{|V_j|}$$

The diagonal elements are given by

$$\frac{\partial \mathcal{P}_i}{\partial \delta_i}(\mathbf{x}) = -\mathcal{Q}_i(\mathbf{x}) - |V_i|^2 B_{ii} \qquad \frac{\partial \mathcal{Q}_i}{\partial \delta_i}(\mathbf{x}) = -\mathcal{P}_i(\mathbf{x}) - |V_i|^2 G_{ii}$$
$$\frac{\partial \mathcal{P}_i}{\partial |V_i|}(\mathbf{x}) = \frac{\mathcal{P}_i(\mathbf{x}) + |V_i|^2 G_{ii}}{|V_i|} \qquad \frac{\partial \mathcal{Q}_i}{\partial |V_i|}(\mathbf{x}) = \frac{\mathcal{Q}_i(\mathbf{x}) - |V_i|^2 B_{ii}}{|V_i|}$$

[1] Note that $\mathcal{P}_i = \sum_{j=1}^{N} \mathcal{P}_{ij}$ and $\mathcal{Q}_i = \sum_{j=1}^{N} \mathcal{Q}_{ij}$.

Chapter 4

Numerical solvers

As the zero of the power mismatch function is hard to find analytically, we have to use numerical solvers to approximate the answer. This approach is often found when dealing with non-linear equations. In this section we shall first discuss a standard method — the Newton-Raphson solver — and then discuss two possible methods that improve the convergence.

4.1 Newton-Raphson solver

One of the most-used numerical solvers is the Newton-Raphson method. We shall firstly derive the method, and then apply it to the power mismatch function.

Suppose we have a function $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n, \mathbf{F} = (F_1, \ldots, F_n)^T$ for which we want to find an $\mathbf{x}^\star$ such $\mathbf{F}(\mathbf{x}^\star) = 0$. Firstly, let us assume that this $\mathbf{x}^\star$ actually exists. Secondly, we assume that for all $i, j, k = 1, \ldots, n$, the second partial derivatives $\frac{\partial^2 f_i}{\partial x_j \partial x_k}$ exist and are continuous. In this case, we may expand $\mathbf{F}$ around a certain $\mathbf{x}$ as a Taylor polynomial and evaluate that in $\mathbf{x}^\star$:

$$\mathbf{F}(\mathbf{x}^\star) = \mathbf{F}(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})\,(\mathbf{x}^\star - \mathbf{x}) + \mathcal{O}\left(||\mathbf{x}^\star - \mathbf{x}||^2\right) \tag{4.1}$$

where $J_{\mathbf{F}}(\mathbf{x})$ is given by

$$J_{\mathbf{F}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \tag{4.2}$$

Now if $\mathbf{x}$ is close enough to $\mathbf{x}^\star$, we may neglect $\mathcal{O}\left(||\mathbf{x}^\star - \mathbf{x}||^2\right)$ to just obtain

$$0 = \mathbf{F}(\mathbf{x}^\star) = \mathbf{F}(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})\,(\mathbf{x}^\star - \mathbf{x}) \tag{4.3}$$

From this, we may derive an iteration scheme $(\mathbf{x}^k)_k$ such that

$$J_{\mathbf{F}}\left(\mathbf{x}^k\right)\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right) = -\mathbf{F}(\mathbf{x}^k) \tag{4.4}$$

Setting $\mathbf{s}^k = \mathbf{x}^{k+1} - \mathbf{x}^k$, this means that we have to solve the linear system $J_{\mathbf{F}}(\mathbf{x}^k)\mathbf{s}^k = -\mathbf{F}(\mathbf{x}^k)$ at each step. We then update the iterate by $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$.
If we assume that $J_{\mathbf{F}}(\mathbf{x}^\star)$ is continuous, bounded and non-singular, it may be shown that this scheme converges in a open set around $\mathbf{x}^\star$. However, it is also true that with a poorly-chosen initial value, the scheme may not converge. We can take steps to prevent the divergence of the scheme. The first we have already seen. This is the use of a per-unit system, the use of which makes it easier to give good initial values and helps with the speed of the convergence. The next is modifying our scheme. We shall discuss these modifications in the next section.

4.2 Globally convergent Newton-Raphson methods

In the previous section, we mentioned that the standard Newton-Raphson method does not always converge. This section will be dedicated to two methods, which should always converge to a local minimum.

4.2.1 Line search

We have seen in the previous section that the Newton-Rahpson method does not always converge. In this section, we want to discuss a method to prevent divergence, at the cost of some additional computational power. This method is called a line search.
The method still starts with at each iteration solving the linear system $J_{\mathbf{f}}(\mathbf{x}^k)\mathbf{s}^k = -\mathbf{f}(\mathbf{x}^k)$ until for some K the norm of the power mismatch function drops below a certain tolerance, i.e. $||\mathbf{F}(\mathbf{x}^K)|| < \varepsilon$. We assume that we have not found this iterate yet. This method, just like the Newton method, relies upon the assumption that $\mathbf{s}^k$ is in the general direction of the root. Instead of updating with the full $\mathbf{s}^k$, we now update our iterate with

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \mathbf{s}^k \tag{4.5}$$

Where $\lambda^k \in (0, 1]$ is a parameter we have yet to determine. Before we discuss how we pick a λ^k, we firstly define some functions as groundwork.
Because the coming part of the section only talks about one iteration steps, we shall suppress the current iterate. That is, we write $\lambda, \mathbf{x}$ and $\mathbf{s}$ for $\lambda^k, \mathbf{x}^k$ and $\mathbf{s}^k$. This should also help with readability.

Firstly, let us define a new positive function f given by

$$f(\mathbf{x}) = \frac{1}{2}||\mathbf{F}(\mathbf{x})||^2 = \frac{1}{2}\mathbf{F}(\mathbf{x})^T\mathbf{F}(\mathbf{x}) \tag{4.6}$$

Next, define

$$g(\lambda) = f(\mathbf{x} + \lambda\mathbf{s}) \tag{4.7}$$

It can clearly be seen that if g is minimised, then $||\mathbf{F}(\mathbf{x} + \lambda\mathbf{s})||$ is too. Since we are interested in finding a zero of $\mathbf{F}$, minimising the norm in this way is a method that should lead us closer to finding that zero[1]. However, this minimisation process is difficult if not impossible to do analytically. For this reason, we try to find a λ that is 'good enough'. Let us examine the way in which we can find such a λ.
A vector $\mathbf{t}$ is called a descent direction of f in $\mathbf{x}$ if

$$\nabla f(\mathbf{x})^T\mathbf{t} < 0 \tag{4.8}$$

We may compute $\nabla f(\mathbf{x})$ by

$$\nabla f(\mathbf{x}) = \nabla \sum_{i=1}^{n} F_i^2(\mathbf{x}) = J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x}) \tag{4.9}$$

Furthermore, recall that we may use $\mathbf{s} = -\left(J_{\mathbf{F}}(\mathbf{x})\right)^{-1}\mathbf{F}(\mathbf{x})$ as $J_{\mathbf{F}}$ was non-singular. Then

$$\nabla f(\mathbf{x})^T\mathbf{s} = -\mathbf{F}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x}) J_{\mathbf{F}}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x}) = -||\mathbf{F}(\mathbf{x})||^2 < 0 \tag{4.10}$$

which gives us that the Newton step is a descent direction.
We now use this property to give a rule, called the Armijo rule, which will dictate if a λ is good enough. λ is sufficient if it satisfies

$$f(\mathbf{x} + \lambda\mathbf{s}) \leq f(\mathbf{x}) + \alpha\lambda\nabla f(\mathbf{x})^T\mathbf{s} \tag{4.11}$$

[1]Do note that it might still be possible that we find a local minimum this way, in which case we should pick another initial value.

where $\alpha \in (0,1)$ a constant we can still choose. One might think that just requiring $f(\mathbf{x}+\lambda\mathbf{s}) \leq f(\mathbf{x})$ should be sufficient. However, in that case examples may be constructed that do converge, but not to a zero of the function. A possible choice would be to $\alpha = 10^{-4}$.[4] Because we know that $\nabla f(\mathbf{x})^T\mathbf{s} < 0$, this means that this choice for λ should result in a new iterate for which f is decreased, which is what we want. If this is not immediately clear, using the definition of f and equation (4.10) to write the Armijo rule as

$$||\mathbf{F}(\mathbf{x}+\lambda\mathbf{s})||^2 \leq (1-2\alpha\lambda)||\mathbf{F}(\mathbf{x})||^2 \tag{4.12}$$

What we may now do is try multiple values for λ and see if they satisfy the Armijo rule. It is guaranteed that the method is satisfied when λ is very small[2], so trying values of λ close to 0 will quite often work. However, doing this also makes the next iterate close to the current iterate, which means that we eventually will need more iterations. This is not beneficial for the speed of the convergence of our scheme. Hence, we start with values close to 1. This approach is also called 'backtracking'.
There are multiple ways we can go about this approach. The first would be to start at $\lambda_0 = 1$ and each time a given λ_i does not satisfy (4.12), we subtract a fixed small number η to get $\lambda_{i+1} = \lambda_i - \eta$. We then try this until either the Armijo rule is satisfied, or until we get to $\lambda = 0$. In the latter case, we are left with two options. Either we try again with a smaller value of η, or we just make a standard Newton-Raphson step. This means that we have either 'wasted' a lot of computational power, or require a lot more. Either way, it would be more efficient to implement another method. Another approach would be to each time multiply λ with a set value $\rho \in (0,1)$: $\lambda_{i+1} = \rho\lambda_i = \rho^{i+1}$. Not every value of ρ is acceptable, however. Relatively high values of ρ results in the need to check more values λ_i, which in turn results in more function evaluations. On the other hand, while small values of ρ might result in a very rapid decrease in the iterates λ_i, this might also cause the algorithm to often find extremely small values for λ which is not desirable for the general convergence of the algorithm. For this reason, we pick $\rho \in [0.1, 0.5]$. This is called safeguarding.
We discuss one last method for finding λ. Rather than have ρ be a fixed number, we may change it per iterate, i.e. $\lambda_{i+1} = \rho_i\lambda_i$. We do, pick $\rho_i \in [0.1, 0.5]$ due to the reasons explained above. To get our choice for ρ_i, we recall the function g from (4.7). We know two things about this function:

$$g(0) = f(\mathbf{x}) \quad \text{and} \quad g'(0) = \nabla f(\mathbf{x})^T\mathbf{s} = -||\mathbf{F}(\mathbf{x})||^2 \tag{4.13}$$

Furthermore, we may also compute $g(\lambda_i)$. A possibility would be to make a quadratic model $\hat{g}_i$ of g and try to minimise that for each iteration i. In this case, $\hat{g}_i$ would be given by

$$\hat{g}_i(\lambda) = \left(\frac{g(\lambda_i) - g(0)}{\lambda_i^2} - \frac{g'(0)}{\lambda_i}\right)\lambda^2 + g'(0)\lambda + g(0) \tag{4.14}$$

This function is minimised at

$$\hat{\lambda}_i = \frac{-g'(0)}{2\left[\frac{g(\lambda_i)-g(0)}{\lambda_i} - g'(0)\right]}\lambda_i \tag{4.15}$$

This means that we want to pick $\lambda_{i+1} = \hat{\lambda}_i$, and thus $\rho_i = \frac{-g'(0)}{2\left[\frac{g(\lambda_i)-g(0)}{\lambda_i} - g'(0)\right]}$. However, if it is the case that now $\rho_i \notin [0.1, 0.5]$, we disregard that value and pick $\rho_i = \text{argmin}_{\lambda\in\{0.1,0.5\}}\hat{g}_i(\lambda)$ due to the safeguards imposed on ρ_i. Namely, this might indicate that g is poorly modelled by $\hat{g}_i$. This last approach can be implemented in such a way that it only requires one function evaluation per iteration of λ.

[2]The Taylor expansion of f around $\mathbf{x}$ is given by $f(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \mathcal{O}(||\mathbf{y}-\mathbf{x}||^2)$. This results in us needing to show that for some λ, $f(\mathbf{x}+\lambda\mathbf{s}) - f(\mathbf{x}) - \alpha\lambda\nabla f(\mathbf{x})\mathbf{s} = (1-\alpha)\lambda\nabla f(\mathbf{x})^T\mathbf{s} + \mathcal{O}(||\lambda\mathbf{s}||^2) \leq 0$. We may obtain this from the fact that $(1-\alpha)\nabla f(\mathbf{x})^T\mathbf{s} < 0 = \lim_{\lambda\to 0^+}\mathcal{O}\left(\frac{||\lambda\mathbf{s}||^2}{\lambda}\right)$. We conclude that such a λ exists (in fact, there is an entire interval of possible values of λ of the form $(0,\eta)$ for some small $\eta > 0$), but it might be very close to 0.

4.2.2 Trust regions

In the previous section, we assumed that we had a good direction for the new iterate, and wanted to compute the length of the step. In this section, we do the opposite: we give a step length and then try to find the best direction to go in.

We assume we have a trust region for the update vector of the form of a hypersphere, i.e.

$$||\mathbf{s}|| < \delta \tag{4.16}$$

for some $\delta > 0$. At this point, we would like to find

$$\hat{\mathbf{s}} = \text{argmin}_{||\mathbf{s}||<\delta} ||F(\mathbf{x}+\mathbf{s})|| \tag{4.17}$$

Unfortunately, solving this is just as hard as the original problem. The answer lies with returning to a quadratic model. For this, we realise that the Newton-Raphson step $s^N = -J_{\mathbf{F}}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$ is a root of $\mathbf{F}(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})\mathbf{s}$, which means that it will also be a root of

$$m(\mathbf{x}+\mathbf{s}) = \frac{1}{2}||\mathbf{F}(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})\mathbf{s}||^2 = f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\mathbf{s} \tag{4.18}$$

Furthermore, note that $\mathbf{s}^N$ is in descent direction for m, as the gradients of m and f evaluated in $\mathbf{x}$ are the same [3]. The problem we then have to solve is

$$\begin{aligned} \min \quad & m(\mathbf{x}+\mathbf{s}) \\ \text{subject to} \quad & ||\mathbf{s}|| \leq \delta \end{aligned}$$

If $||\mathbf{s}^N|| \leq \delta$, then this problem is solved by $\mathbf{s}^N$. In the case that $||\mathbf{s}^N|| > \delta$, it has been proven that this minimisation problem is solved by

$$\hat{\mathbf{s}}(\mu) = -\left(J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x}) + \mu I_n\right)^{-1} J_{\mathbf{F}}(\mathbf{x})^T \mathbf{F}(\mathbf{x}) \tag{4.19}$$

where I_n is the $n \times n$ identity matrix and μ is the (unique) number such that $||\hat{\mathbf{s}}(\mu)|| = \delta$. We point out that $\hat{\mathbf{s}}(0) = \mathbf{s}^N$ and that if μ is large, then $\hat{\mathbf{s}}(\mu) \approx -\frac{1}{\mu} J_{\mathbf{F}}(\mathbf{x})^T \mathbf{F}(\mathbf{x}) = \frac{1}{\mu}\left(-\nabla f(\mathbf{x})\right)$. We shall use these facts later.

We now need to explore two things: how to find μ if $\mathbf{s}^N$ does not suffice, and how to update δ each iteration. We firstly try to find an appropriate μ. For this, we want to solve

$$\chi(\mu) = ||\mathbf{s}(\mu)|| - \delta = 0 \tag{4.20}$$

This is an expensive problem to solve due to the fact that we need to compute an inverse of a matrix. However, it should be noted that an approximate solution for this problem will also suffice.

Double dogleg method

There are multiple ways to tackle problem (4.20). In the case that $||\mathbf{s}(0)|| \leq \delta$, we are already done. So, we shall assume that this is not the case. We shall highlight a modification of Powell's method, called the double dogleg step. In this method, we create a piecewise linear path $\boldsymbol{\alpha}(\theta), \theta \in [0,3]$ that approximates the curve of $\mathbf{s}(\mu)$, consisting of 3 pieces. The first part of the curve will be in the steepest-decent direction, while the last part will be in the direction of the Newton-Raphson step. The middle piece just connects the ends together. Hence, we already know that $\boldsymbol{\alpha}(0) = \mathbf{x}$ and $\boldsymbol{\alpha}(3) = \mathbf{x} + \mathbf{s}^N$. We give some additional requirements for the curve. Firstly, the distance between the curve and the centre of the trust region should increase strictly monotonically, i.e. $||\mathbf{x} - \boldsymbol{\alpha}(\theta)|| < ||\mathbf{x} - \boldsymbol{\alpha}(\vartheta)||$ if $\theta < \vartheta$. The reason we require this is that there can be at most one

[3] Do note, however, that m is not the quadratic Taylor expansion of f. This can be seen from the fact that $J_{\mathbf{F}}^T J_{\mathbf{F}} = H_m \neq H_f$, where H denotes the Hessian matrix.

point for which $\boldsymbol{\alpha}$ is exactly δ from $\mathbf{x}$. Secondly, along the curve m should decrease monotonically, i.e. $m(\mathbf{x}+\boldsymbol{\alpha}(\theta)) \geq m(\mathbf{x}+\boldsymbol{\alpha}(\vartheta))$ if $\theta < \vartheta$.
We create the first piece. For this we need the Cauchy point $\mathbf{c}$. This is the point that minimises the quadratic model m in the direction of steepest-decent, given by $-\nabla f(\mathbf{x})$. That is, it can be found by minimising

$$m(\mathbf{x}-\lambda\nabla f(\mathbf{x})) = f(\mathbf{x}) - \lambda||\nabla f(\mathbf{x})||^2 + \frac{\lambda^2}{2}\nabla f(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\nabla f(\mathbf{x}) \tag{4.21}$$

over $\lambda \in \mathbb{R}$. This has the solution

$$\hat{\lambda} = \frac{||\nabla f(\mathbf{x})||^2}{||J_{\mathbf{F}}(\mathbf{x})\nabla f(\mathbf{x})||^2} \tag{4.22}$$

Therefore the Cauchy point $\mathbf{c}$ is given by $\mathbf{c} = \mathbf{x} - \hat{\lambda}\nabla f(\mathbf{x})$. In case that the Cauchy point lies on the boundary or outside of the trust region, meaning $\hat{\lambda}||\nabla f(\mathbf{x})|| \geq \delta$, then we take a step of length δ in the steepest-decent direction. That is,

$$\mathbf{s} = \delta\frac{-\nabla f(\mathbf{x})}{||-\nabla f(\mathbf{x})||} \tag{4.23}$$

Regardless of the distance of $\mathbf{c}$ from $\mathbf{x}$, our first piece will now be given by the line between $\mathbf{x}$ to $\mathbf{c}$. We may summarise this as $\boldsymbol{\alpha}(\theta) = (1-\theta)\mathbf{x} + \theta\mathbf{c}$ for $\theta \in [0,1]$.
Note that this first piece satisfies both the requirements. Clearly, the distance of $\boldsymbol{\alpha}$ increases as θ increases, and because $\mathbf{c}$ is a minimiser of equation (4.21) it satisfies the second requirement.

Our middle piece needs to connect $\mathbf{c}$ to the line in the direction of the Newton-Raphson step $\mathbf{s}^N$. For this step to make sense, we do need to make sure that $\mathbf{c}$ is not further away from $\mathbf{x}$ than $\mathbf{x}+\mathbf{s}^N$. This is shown by the following computation:

$$\begin{aligned}
||\mathbf{c}-\mathbf{x}|| &= \frac{||\nabla f(\mathbf{x})||^3}{\nabla f(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\nabla f(\mathbf{x})} \\
&\leq \frac{||\nabla f(\mathbf{x})||^3}{\nabla f(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\nabla f(\mathbf{x})} \frac{||\nabla f(\mathbf{x})||\ ||(J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x}))^{-1}\nabla f(\mathbf{x})||}{\nabla f(\mathbf{x})^T (J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x}))^{-1}\nabla f(\mathbf{x})} \\
&= \frac{||J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x})||^4}{||J_{\mathbf{F}}(\mathbf{x})J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x})||^2||\mathbf{F}(\mathbf{x})||^2}||\mathbf{s}^N|| = \gamma||\mathbf{s}^N||
\end{aligned}$$

From $||J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x})||^2 = \mathbf{F}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x}) \leq ||\mathbf{F}(\mathbf{x})||\ ||J_{\mathbf{F}}(\mathbf{x})J_{\mathbf{F}}(\mathbf{x})^T\mathbf{F}(\mathbf{x})||$ it may be concluded that $\gamma \leq 1$. Equality only holds if $\mathbf{x}-\mathbf{c} = \mathbf{s}^N$ [4] , in which case the construction of this path is not necessary [4]. We therefore exclude this possibility.

Using this, we construct a point $\hat{\mathbf{N}}$ of the form

$$\hat{\mathbf{N}} = \mathbf{x} + \eta\mathbf{s}^N \tag{4.24}$$

for some η. This η should satify:

- $\gamma \leq \eta \leq 1$
- m decreases monotonically along the line between $\mathbf{c}$ and $\hat{\mathbf{N}}$.

To show that this holds, parametrise the line segment by $\mathbf{y}_\eta(\tau) = \mathbf{s}^{\mathbf{c}} + \tau\left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right)$, $\tau \in [0,1]$, where $\mathbf{s}^{\mathbf{c}} = \mathbf{c}-\mathbf{x} = -\hat{\lambda}\nabla f(\mathbf{x})$. The derivative of m along $\mathbf{y}_\eta$ is the given by

[4] If both are inside, then $\mathbf{s}^N$ is chosen. Otherwise, we follow equation (4.23).

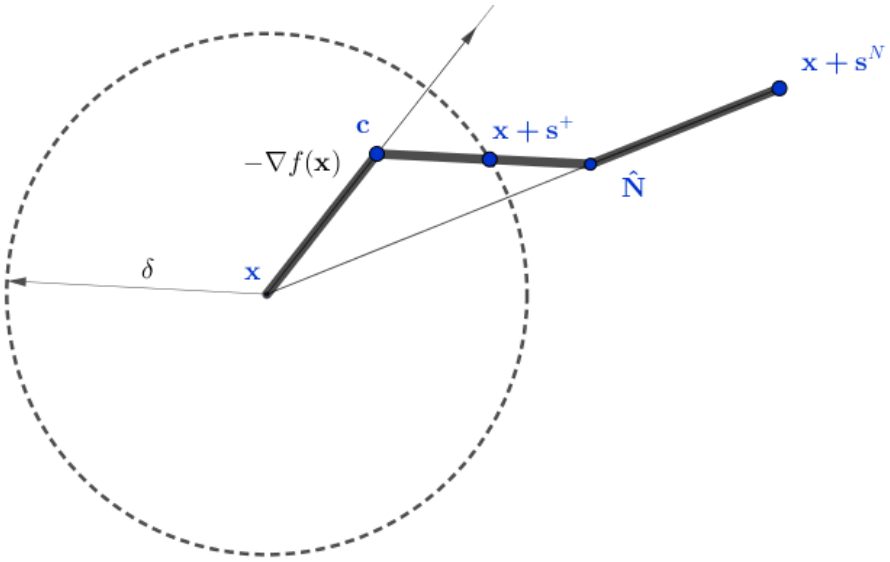


Figure 4.1: A visualisation of the double dogleg curve $\boldsymbol{\alpha}(\theta)$ with the points $\mathbf{x}, \mathbf{c}, \hat{\mathbf{N}}$ and $\mathbf{x}+\mathbf{s}^N$ indicated. The point where this curve goes outside the trust region is the candidate for our next iterate.

$$\begin{aligned}
&\left(\nabla_{\mathbf{s}}\, m(\mathbf{x}+\mathbf{s})|_{\mathbf{s}=\mathbf{y}(\tau)}\right)^T \left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right) \\
&= \left(\nabla f(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x}) \left(\mathbf{s}^{\mathbf{c}} + \tau\left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right)\right)\right)^T \left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right) \\
&= \left(\nabla f(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\mathbf{s}^{\mathbf{c}}\right)^T \left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right) + \tau\left|\left|J_{\mathbf{F}}(\mathbf{x})(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}})\right|\right|^2 \qquad (4.25)
\end{aligned}$$

We must have that (4.25) is negative for all τ. As it is increasing in τ, we only need to verify $\tau = 1$.

That is, we need to find η that satisfy

$$\begin{aligned}
0 &\geq \left(\nabla f(\mathbf{x}) + J_{\mathbf{F}}(\mathbf{x})^T J_{\mathbf{F}}(\mathbf{x})\mathbf{s}^{\mathbf{c}}\right)^T \left(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}}\right) + \tau\left|\left|J_{\mathbf{F}}(\mathbf{x})(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}})\right|\right|^2 \\
&= (1-\eta)\left(\nabla f(\mathbf{x})^T(\eta\mathbf{s}^N - \mathbf{s}^{\mathbf{c}})\right) \\
&= (1-\eta)(\gamma-\eta)||\mathbf{F}||^2
\end{aligned}$$

The above equation is satisfied by $\eta \in [\gamma, 1]$, meaning that both above requirements for η are equivalent. As such, $\hat{\mathbf{N}} = \mathbf{x} + \eta\mathbf{s}^N$ for some $\eta \in [\gamma, 1]$. If $\eta = 1$, then we obtain the (single) dogleg method. The single dogleg method was Powell's original method. For best performance of the algorithm, $\eta = 0.8\gamma + 0.2$ is suggested.[10]

The third and last part of $\boldsymbol{\alpha}$ is now just given by the line between $\hat{\mathbf{N}}$ and $\mathbf{x}+\mathbf{s}^N$. It should be clear that m does decrease along this line, as m is the quadratic model with root $\mathbf{s}^N$.
We may now define $\boldsymbol{\alpha}$ by

$$\boldsymbol{\alpha}(\theta) = \begin{cases} (1-\theta)\mathbf{x} + \theta\mathbf{c} & \theta \in [0,1] \\ (2-\theta)\mathbf{c} + (\theta-1)\hat{\mathbf{N}} & \theta \in [1,2] \\ (3-\theta)\hat{\mathbf{N}} + (\theta-2)(\mathbf{x}+\mathbf{s}^N) & \theta \in [2,3] \end{cases} \qquad (4.26)$$

This curve is more for the mathematical properties than for a computer implementation, however. Once $\mathbf{c}, \hat{\mathbf{N}}$ and $\mathbf{s}^N$ are computed, we may compute $\boldsymbol{\alpha}(\theta)$ with relatively little extra computational expense. Furthermore, note that the points $\mathbf{c}$ and $\hat{\mathbf{N}}$ can be computed with just $J_{\mathbf{F}}(\mathbf{x})$ and $\mathbf{F}(\mathbf{x})$, which is again relatively cheap as these have already been computed for $\mathbf{s}^N$.

Updating the trust region

Up to this point, we have not discussed the size of the trust region. We shall do this now.

We always start of with a guess for a good δ. However, this δ is by no means fixed. After we construct $\boldsymbol{\alpha}(\theta)$ as above, we find the new update vector $\mathbf{s}^+$ such that $||\mathbf{s}^+|| \approx \delta$. After that, we simply check if this update vector is good enough, in the sense that it satisfies the Armijo rule from equation (4.12) modified to our problem:

$$f(\mathbf{x} + \mathbf{s}^+) \leq f(\mathbf{x}) + \alpha g(\mathbf{x})^T \mathbf{s}^+ \tag{4.27}$$

where $\alpha \in (0, 1)$ and $g(\mathbf{x})$ is given by $\nabla f(\mathbf{x}) = J_{\mathbf{F}}(\mathbf{x})\mathbf{F}(\mathbf{x})$ or some approximation thereof. The approximation is mostly used in the more general problem, but can still be utilised. However, we shall use $g = \nabla f$. For this section we will just use Just as in the previous section, we pick $\alpha = 10^{-4}$. If $\mathbf{s}^+$ satisfies the Armijo rule, then that is our new update vector. Otherwise, we decrease δ similarly to how we updated λ in the line-search algorithm[5].
We define the iterates δ_i for this process. If δ_i does not satisfy equation (4.27), then we multiply it with some constant ρ_i, i.e. $\delta_{i+1} = \rho_i \delta_i$. We safeguard this $\rho_i \in \left[\frac{1}{10}, \frac{1}{2}\right]$. A first choice would be to have all ρ_i be the same number, for example $\rho_i = \rho = \frac{1}{2}$ for all i.
Another choice would be minimise $f(\mathbf{x} + \beta \mathbf{s}^+)$ for β using a quadratic model. We spare the details and give the answer, as this is much the same as explained in section 4.2.1.

$$\hat{\beta} = \frac{-\nabla f(\mathbf{x})^T \mathbf{s}^+}{2\left[f(\mathbf{x} + \mathbf{s}^+) - f(\mathbf{x}) - \nabla f(\mathbf{x})^T \mathbf{s}^+\right]} \tag{4.28}$$

We then pick $\rho_i = \hat{\beta}$ if $\hat{\beta} \in \left[\frac{1}{10}, \frac{1}{2}\right]$, otherwise we pick ρ_i to be $\frac{1}{10}$ or $\frac{1}{2}$ depending on which minimises the model the most.

4.3 Implementation

Now that we have seen the power flow equations and possible algorithms to solve them, we quickly discuss how to implement these. As is usual with these types of problems, it is extremely labour intensive to do them by hand, hence why we implement them with the help of computers. This was done in python.
There are several notable packages used, which we briefly want to highlight. Firstly, we made use of PandaPower ,[7] which has data structures in place to model grids, as well as multiple examples. It makes use of a standard Newton-Raphson algorithm (see section 4.1). For basic examples, this package was used to verify the implementations of other algorithms.
Next, we made use of SciPy's [8] fsolve. This function, which is based on the work of many people, makes use of the modified Powell algorithm from section 4.2.2. It makes use of many more algorithms (such as the Levenberg–Marquardt algorithm, Jacobian-approximating schemes), but discussing all these is not the aim of this document.
Lastly, the power mismatch function and its Jacobian were implemented with the use of SymPy .[9] This is a library for symbolic mathematics, which allows us to construct the function entirely symbolically before evaluating it. This was done to prevent having to construct the function each time we want to evaluate it, to speed up the algorithm. However, it should be noted that this approach has a significant initial set-up time. As this is the case, it should only be used on small examples or if the same grid gets examined multiple times with possibly different values.

Lastly, the methods described above are all iterative methods. To ensure that the algorithms halt, we need to have some stopping condition on the number of iterations. In this case, we may give the user a warning that the method has not converged, and recommend either choosing a better initial value or one of the other methods. For solving large problems, it might be that an increased maximum number of iterations helps but it is often not recommended.

[5] Do note that the double dogleg curve $\boldsymbol{\alpha}$ does not depend on δ and can be kept the same throughout the whole process of finding a good δ.

Chapter 5

Examples and results

Now that we have our model and methods to solve it, we want to derive some results from small examples and examine the behaviour of the numerical schemes.

5.1 Small examples of load flows

We start with a very simple grid with just one generating buses and two load buses, all in a row. This might look something like this:

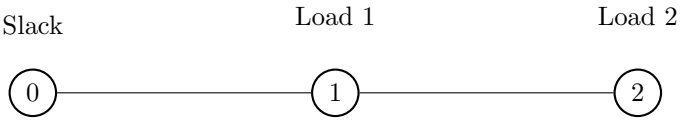


Figure 5.1: A graph representing a grid with one slack bus and two loads, connected in a line.

Now because there is just one generating bus, this is automatically the slack bus. There are no shunts or transformers. We give some default values for this example. Both lines are 1 kilometre long and of a standard type called 'NAYY 4x50 SE'[1]. Both loads ask for $P_1 = P_2 = 1$ MW of active power and $Q_1 = Q_2 = 0.25$ MVAr of reactive power (which results in a power factor of around 0.97). Furthermore, the system frequency is $\omega = 50$ Hz and a grid base voltage level of 50 kV, which are default values for a low voltage grid.
Solving this system, we obtain the values from figure 5.2.

	$\lvert V_i \rvert$ (p.u.)	δ_i (rad)	P_i (MW)	Q_i (MVAr)
Bus 0	1.000000	0.000000	2.001309	0.170604
Bus 1	0.999478	-0.000083	-1.000000	-0.250000
Bus 2	0.999215	0.000482	-1.000000	-0.250000

Figure 5.2: A table of the example of the grid from figure 5.1, with $P_1 = P_2 = 1$ MW and $Q_1 = Q_2 = 0.25$ MVAr.

The results from figure 5.2 were obtained with the standard Newton-Raphson method with a power mismatch tolerance $\varepsilon = 10^{-10}$. For such small examples, the Newton-Raphson method is sufficient. In this case, it only took 3 iterations to converge below the required tolerance.
From the results we can see that the voltage drops a bit after each line, as well as a change in the voltage angle. Furthermore, from P_1 we can see that there is a small system loss. Note that because supplying and absorbing power are opposite, one of the two has a different sign in P and Q. In this case, the power flow is computed from the perspective of generating buses, so the

[1] To view its and others' parameters, visit https://pandapower.readthedocs.io/en/v2.2.2/std_types/basic.html.

supplied load from the generator is positive.

Now that we have seen a small example, let us see if we can make general predictions for a network when one of the loads changes. We still use the same grid from figure 5.1. First, let us see what happens if load 1 suddenly requires different amount of power. We vary P_1 between 0 and 20 MW and say $Q_1 = 0.25P_1$. This results in figure 5.3.

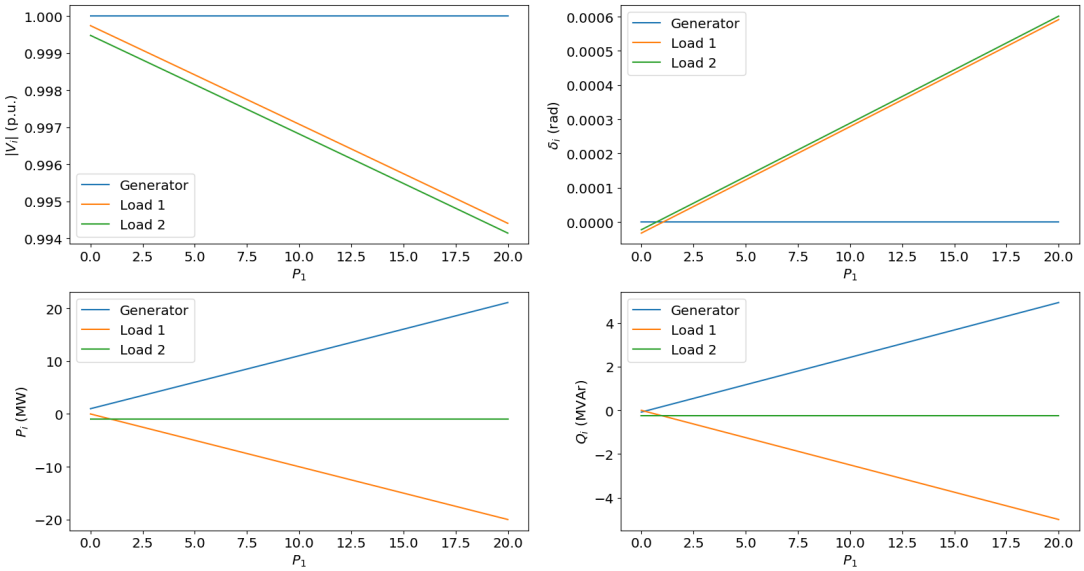


Figure 5.3: A graph of the example from figure 5.1, increasing the power demand from load 1.

From this figure we can clearly see that the bigger the power demand, the larger the voltage drop and change in voltage angles are. Furthermore, we can see that the voltage angle of bus 2 stays the same relative to that of bus 1.

We may work out the same example, but this time we change P_2 and $Q_2 = 0.25P_2$. This results in figure 5.4. We can observe a greater effect on the voltage magnitude and voltage angle, interestingly. Furthermore, also note that even though we only change the demand of the second bus, the first bus is also affected.

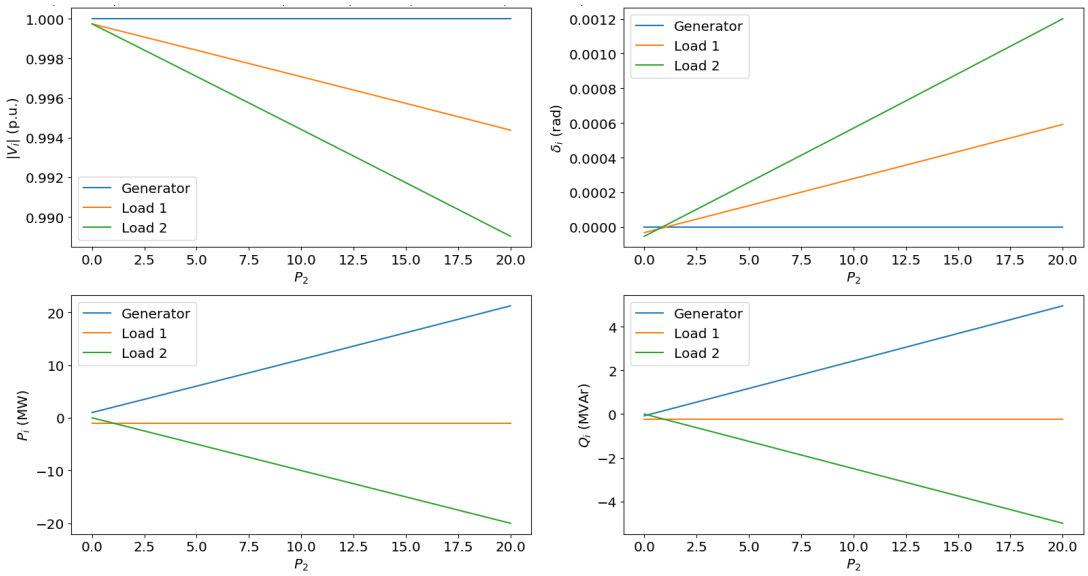


Figure 5.4: A graph of the example from figure 5.1, increasing the power demand from load 2.

Next, we investigate what happens if only the reactive power is increased (and therefore a decreased power factor). The results are given in figure 5.5. In this figure, we can clearly see that again, the voltage drops. This drop is not as significant as for example in figure 5.4, but for that figure the active power is also increased. Furthermore, from P_0 we notice that there are now measurable system losses. Because this example is extremely small, these losses are not really impressive. However, keep in mind that these losses really add up in a larger grid. Seeing

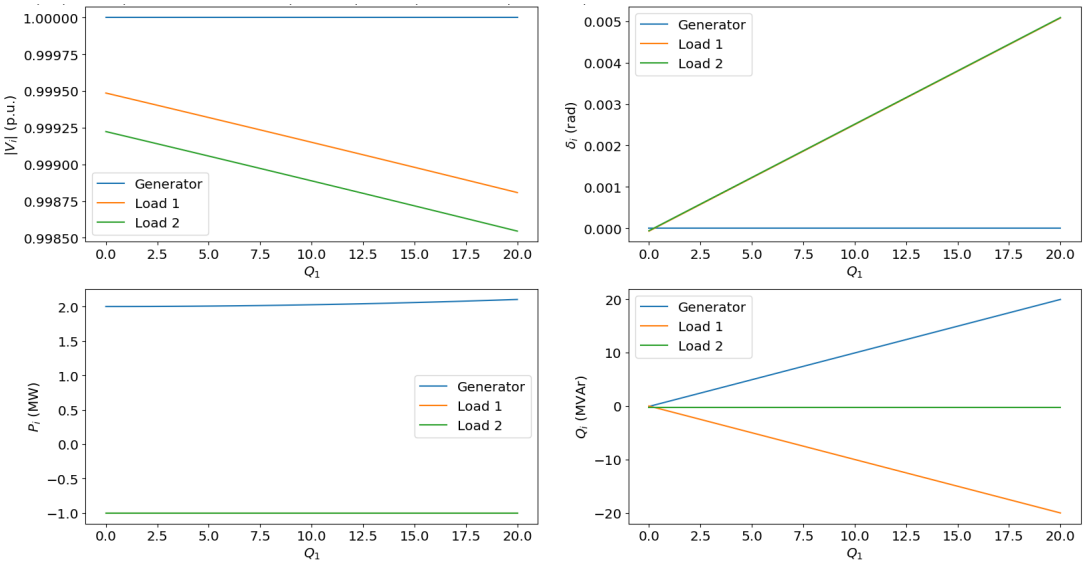


Figure 5.5: A graph of the example from figure 5.1, increasing the reactive power demand from load 1.

what happens when the reactive power demand is increased, leads us to investigate what happens when the power phase angle is changed. For bus 1, we take $|S|$ to be constant and then vary ϕ. We pick $|S| = 1$ MW and then vary $P = |S|\cos(\phi), Q = |S|\sin(\phi)$. This results in figure 5.6. Similarly, we may do this for bus 2. We find these results in figure 5.7. Interestingly, the voltage magnitude actually goes up when ϕ increases in both cases. One might think that this contradicts our statement from before, where we said that a low power factor (and hence a larger

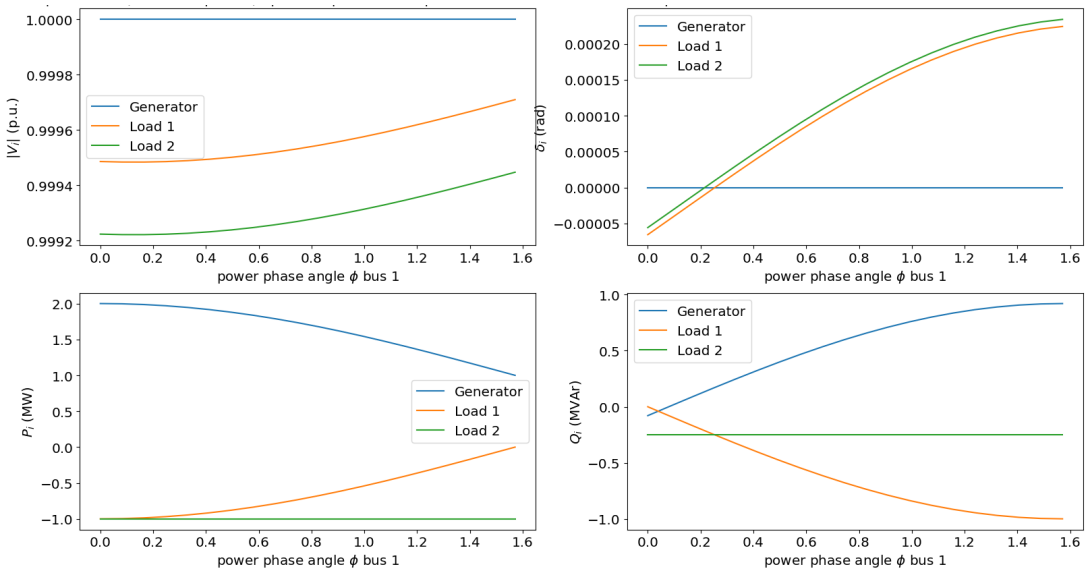


Figure 5.6: A graph of the example from figure 5.1, increasing power phase angle in load 1.

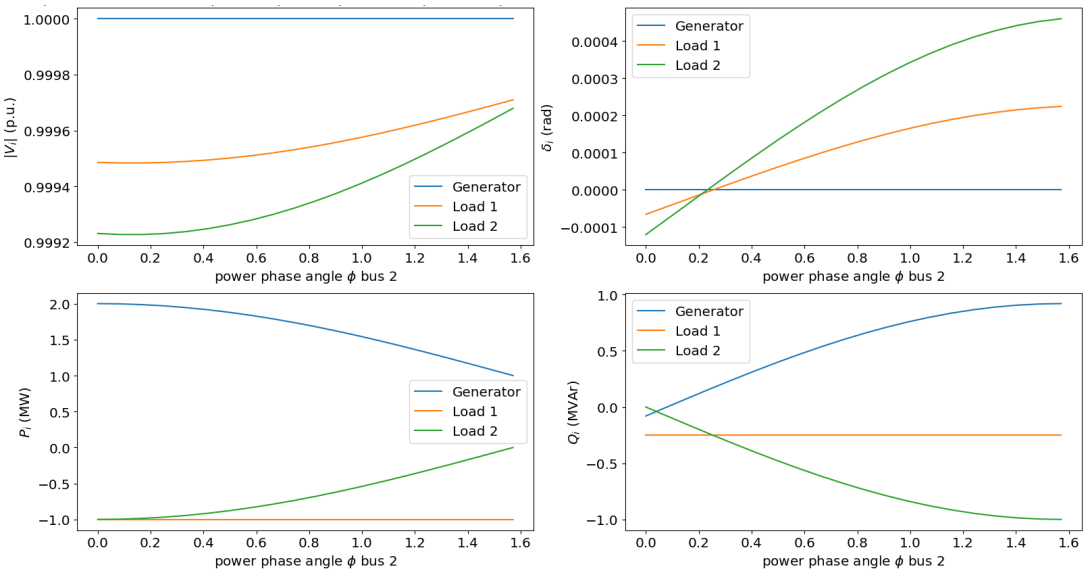


Figure 5.7: A graph of the example from figure 5.1, increasing power phase angle in load 2.

absolute power phase angle) causes voltage drops. This is not true, however, as assumes that the required active power is still the same. In the case of figure 5.6 and 5.7, this is not the case.

Lastly, we want to see what happens when we directly connect bus 2 to the slack bus, resulting in a grid that looks something like figure 5.8. We then increase P_1 and $Q_1 = 0.25P_1$ as above. All lines are kept at 1 km for simplicity and symmetry.

When we do this, we get figure **??** as a result. The most important thing we can get from this picture is that now, the effects of the increased power demands to the voltage magnitudes and angels is less than we have seen in the previous figures. This leads us to the conclusion that generally, it is best to keep the amount of loads between one load and a generator as small as possible, as each load will have an effect on the voltage magnitude and angle of the next.

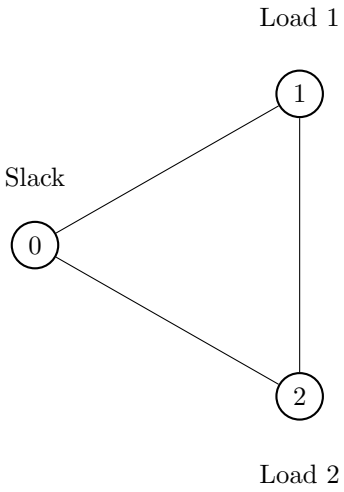


Figure 5.8: A graph representing a grid with one slack bus and two loads, both connected to the slack bus and each other.

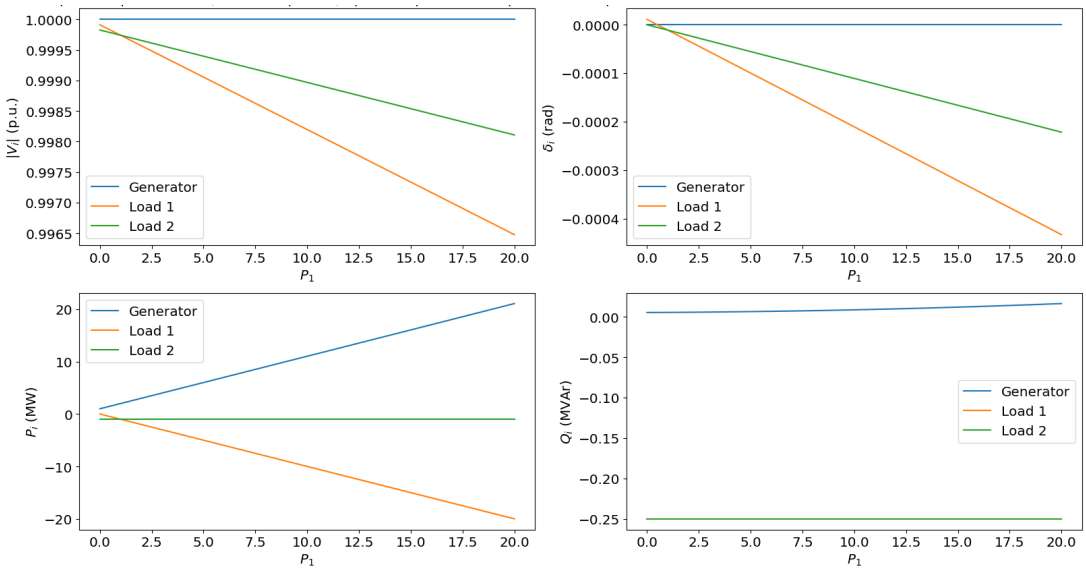


Figure 5.9: The result from a grid with two load buses connected to a slack bus and each other, where we increase the power demand of one of the two.

5.2 Convergence

Let us examine the way these methods converge. To do this we create examples that require more from our numerical schemes than the simple examples we have seen already. However, we do have to take care that the examples we create actually have a solution. Namely, it might well be the case that there is simply no solution to an arbitrarily constructed example.

Firstly, we shall show why we want to formulate our problem in a per-unit system rather than in the actual units. We look at the convergence of the problem as posed in figure 5.3, using the standard Newton-Raphson iteration scheme. We vary P_1 between 0 and 2 MW in 40 steps, so $P_1^i = \frac{i}{20}$ MW for $i = 0, \ldots, 40$ and then take $Q_1^i = 0.25 P_1^i$. For each i we check the convergence

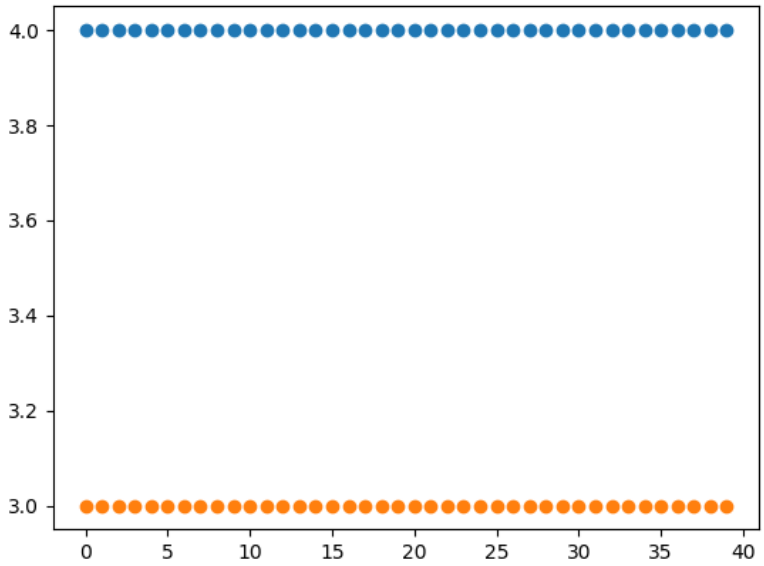


Figure 5.10: A graph on the number of iterations required for the standard Newton-Raphson iteration scheme to converge with a tolerance of $\varepsilon = 10^{-5}$ versus i. The blue dots represent the standard problem, the orange the problem converted to a per-unit system.

of the method, using a tolerance of $\varepsilon = 10^{-5}$. As initial value we take voltage magnitude to be the base voltage level of the grid, 50 kV, and the voltage angles to be 0. We do this for both the problem normally as well as converted to a per-unit system. In the per-unit system, we take $V_{\text{base}} = 50$ kV and $S_{\text{base}} = 1$ MW. All other base values are computed according to section 2.11. This results in figure 5.10.

Noteworthy of the figure is that the per-unit system requires only 3 iterations, while the system with normal units requires 4. This becomes more apparent when we decrease the tolerance to $\varepsilon = 10^{-7}$. The results for this can be found in figure 5.11. The per-unit system still consistently only requires 3 iterations, but the system with the normal units requires a lot more. Some do not even converge beneath the required tolerance, which caused the algorithm to be stopped by the maximum number of iterations. This was set to 100. When this happened, the norm of the power mismatch function $\mathbf{F}$ was around the order of 10^{-6}, meaning that the iterates at some point started to overshoot[2] the solution.
However, to say that they diverge would not be accurate. We have already seen that all points converge with a tolerance of $\varepsilon = 10^{-5}$.

This is important to take into account when implementing the standard Newton-Raphson method. However, the more essential essential point is that this gives us a reason to believe that formulating our problems in a per-unit system results in more accurate solutions.

If we now use one of the other algorithms described above, we may see that their global convergence does allow both methods to converge for a tolerance of $\varepsilon = 10^{-7}$. For this example, we take the line-search algorithm, where we update λ_i with a quadratic model. However, we may also look at the number of function evaluations. These are a lot higher than that of the standard Newton-Raphson solver.

[2]In figure A.1 (appendix) we have done this exact same procedure, but with a maximum of 10000 iterates.

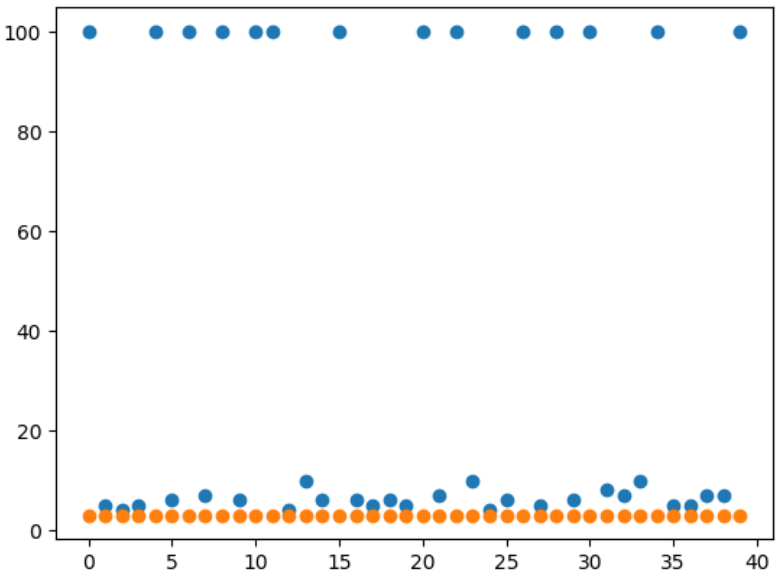
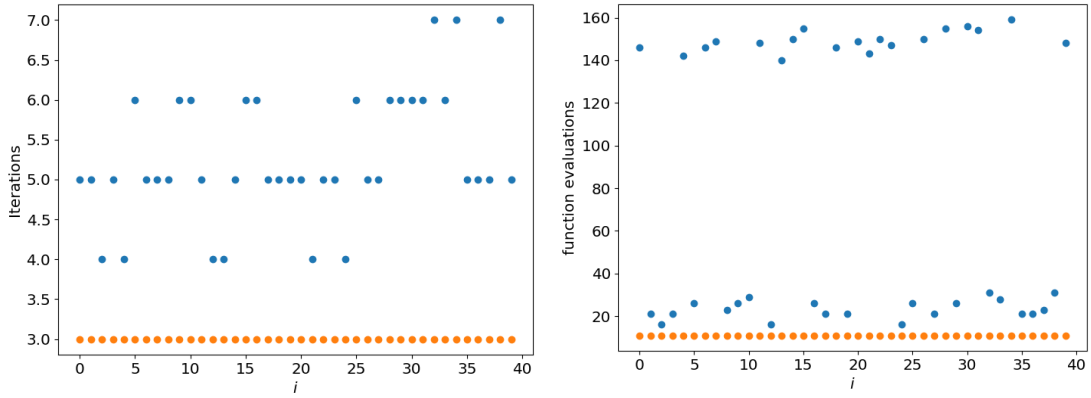


Figure 5.11: A graph on the number of iterations required for the standard Newton-Raphson iteration scheme to converge with a tolerance of $\varepsilon = 10^{-7}$ versus i. The blue dots represent the standard problem, the orange the problem converted to a per-unit system.



(a) Iterations of the scheme

(b) Function evaluations used

Figure 5.12: Iterations and number of function evaluations used when applying the line-search algorithm to the same problem as described in figure 5.10 and 5.11. The orange dots represent the results of the per-unit system, the blue dots that of the standard problem.

Secondly, we look at some some examples where the standard Newton-Raphson method really does diverge. We take the same concept as above, but chain more buses together.

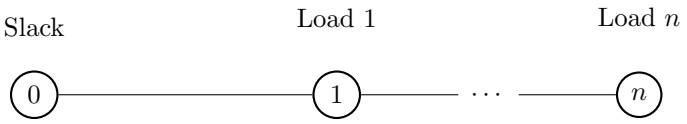


Figure 5.13: A graph representing a grid with one slack bus and n loads, connected in a line.

We will take 0 to be the slack bus, and $1 \dots n$ to each demand 1 p.u. of active power and 0.1 p.u. of reactive power. Otherwise, all parameters are the same as in figure 5.1.
If we apply the standard Newton-Raphson method for increasing n, eventually n will be so great that the scheme diverges. Numerical testing shows that this happens at $n = 18$. This can also be seen in figure 5.14.

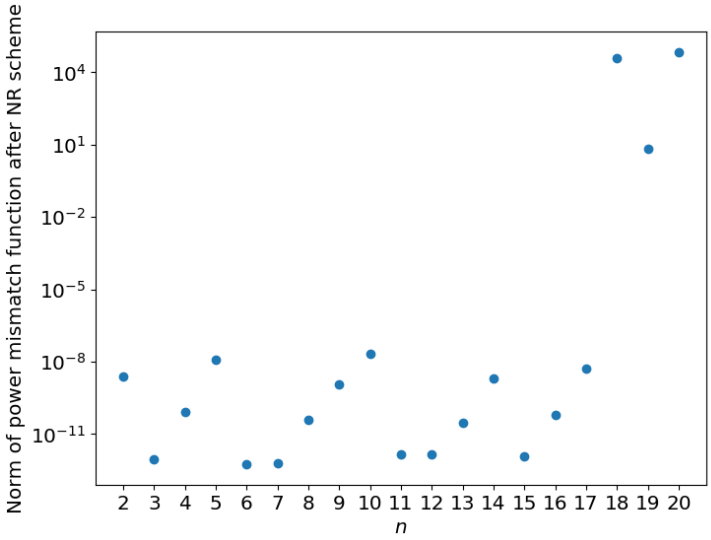


Figure 5.14: The residual norm of the Newton-Raphson scheme after 10000 iterations (or until converged) versus the number of buses in example 5.13.

If we now use one of the globally convergent methods, we do find a solution. However, this is only after a large number of iterations and function evaluations ($\geq$ 100.000 in the case of the line search algorithm).

Lastly, we want to compare our line-search method to the 'professional' fsolve. We take a similar setup as in figure 5.13, but decrease the reactive power to 0.01 p.u.. We then test how many function evaluation each method needs. The results can be found in figure 5.15. In this figure we can clearly see that fsolve usually requires less function evaluations to converge, as was to be expected.

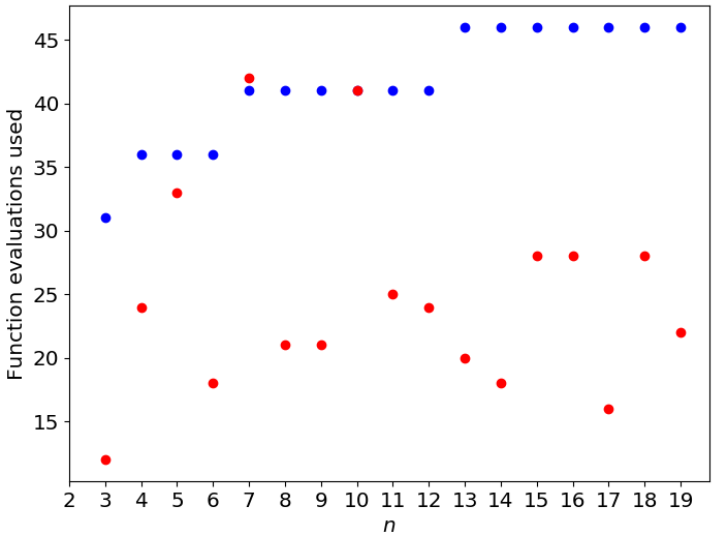


Figure 5.15: Number of function evaluations needed for the line-search method (blue) and fsolve (based on trust region, in red).

Chapter 6

Conclusion

In this report, we set out to construct a model to help us predict the flow of power throughout a network and methods to solve these.
For our model, we took several concepts of circuit analysis and electrical engineering to obtain a set of equations called the power flow equations, which related the power demand or generation to concepts as the admittance and the voltage. To make this model, we did have to make a few assumptions such as the ones that allowed us to lump our elements together. These assumptions are generally accepted as reasonable to make, and this model is widely used. However, any model may still be improved. As an example for a concept that could be added is three-phase power. The addition of three-phase power can add a whole new layer to our model, but it does require quite advanced circuit analysis. In some special cases, however, one might use 3 single phase grids (as explained in this report) to approximate the three-phase power grid.
The power flow equations are unfortunately non-linear equations, meaning that it is very difficult — if not impossible in some cases — to find a solution. To still approximate the solutions, we discussed multiple iteration schemes to help find this solution. We firstly started with the standard Newton-Raphson method, based upon which we also explained two methods that should have better convergence, the line search method and the trust region method. The latter two methods did come at a higher computational cost, however. We then used the model and the numerical schemes to compute the solution to some small examples. While the chosen examples are just that — examples — the methods used may be used on any grid to make predictions about it. Lastly, we examined the convergence of the explained methods. While the expanded methods did converge better (in the sense that they converge for stricter tolerances in more cases), they do require a lot more computational power. If one wants to use these methods, they must keep this into consideration.

Bibliography

[1] R. Idema and D.J.P. Lahaye, *Compuational Methods in Power System Analysis*, Atlantis Press, 2013.

[2] J. J. Grainger and W. D. Stevenson jr. *Power System Analysis*, McGraw-Hill, 1994.

[3] C. K. Alexander and M.N.O.Sadiku, *Fundamentals of Electric Circuits*, 6th ed. McGraw-Hill, 2017.

[4] J. E. Dennis jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996.

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region methods*, SIAM, 2000.

[6] J. van Kan, A. Segal, F. Vermolen, and H. Kraaijevanger, *Numerical Methods for Partial Differential Equations*, Delft Academic Press, 2019.

[7] L. Thurner and A. Scheidler and F. Schäfer and J. Menke and J. Dollichon and F. Meier and S. Meinecke and M. Braun, *pandapower — An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems*, IEEE Transactions on Power Systems **33** (2018), no. 6, 6510-6521, DOI 10.1109/TPWRS.2018.2829021.

[8] Pauli and Gommers Virtanen Ralf and Oliphant, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods **17** (2020), 261–272, DOI https://doi.org/10.1038/s41592-019-0686-2.

[9] Aaron and Smith Meurer Christopher P. and Paprocki, *SymPy: symbolic computing in Python*, PeerJ Computer Science **3** (2017), e103, DOI 10.7717/peerj-cs.103.

[10] J. E. Dennis and H. H. W. Mei, *Two new unconstrained optimization algorithms which use function and gradient values*, Journal of Optimization Theory and Applications **28** (1979), no. 4, 453-482, DOI 10.1007/BF00932218.

Appendix

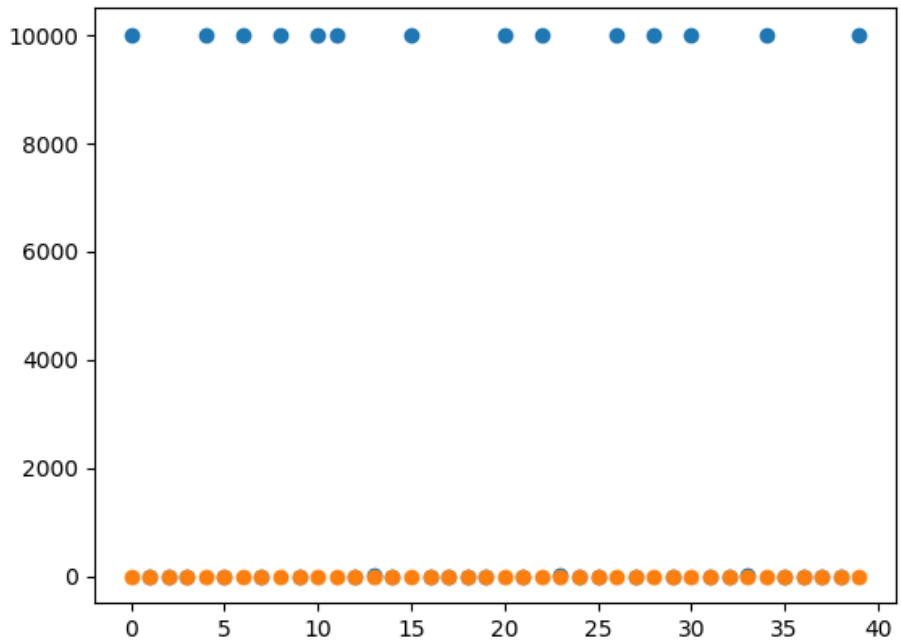


Figure A.1: A graph on the number of iterations required for the standard Newton-Raphson iteration scheme to converge with a tolerance of $\varepsilon = 10^{-7}$ versus i. The blue dots represent the standard problem, the orange the problem converted to a per-unit system. In this figure, the Newton-Raphson iteration scheme took at most 10000 steps.