Technische Universität München
Department of Electrical Engineering and Information Technology
Institute of Power Transmission Systems

Implementation and Evaluation of the Holomorphic Embedding Load Flow Method

Benedikt Schmidt, B.Sc.
benedikt.schmidt@tum.de

Supervisor: Markus Meyer, M.Sc.
Supervising Professor: Prof. Dr.-Ing. Rolf Witzmann
Date of submission: 31.03.2015

Abstract

A load-flow calculation is a key element in planning and running power nets. Over the past decades only iterative methods with insufficient convergence behaviour have been available for this task. Recent research has resulted in a new approach to this problem, the so-called *Holomorphic Embedding Load Flow Method* (*HELM*). This thesis explains how this method can be applied and compares it to the iterative methods. Furthermore, experimental results show that the superior convergence behaviour of *HELM* enables the load-flow calculation of nets closer to their border of stability than with any other iterative method. This is made possible by a trade-off with respect to runtime through special settings. With default settings *HELM* delivers already more accurate results in comparable runtime to the iterative methods. Therefore, in practice one has to evaluate the use of *HELM* with high accuracy settings but it can be used out-of-the-box for most cases.

Statutory declaration

I declare that I have authored this thesis independently, that I have not used any other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Innsbruck, 31.03.2015 *Benedikt Schmidt*

Contents

Introduction

The energy revolution, driven by consumers, politics and the society as a whole, is stressing our power supply system. For instance, the decision to shut down the atomic power plants in Germany in the foreseeable future will shift the main power sources from the South of Germany to the North. Historically, these sources were placed close to the main loads in the power net, but with the need to use renewable energy sources our power plants will be located where the energy sources are available and not where the big cities and the industry are located. Therefore, our power nets will have to change in order to be able to transport the energy from the sources to the loads.

One of the necessary steps for a change in the power net is a static load-flow calculation for which only algorithms with a bad convergence behaviour in the past have been available. These algorithms have one thing in common: they are iterative and can therefore not guarantee to find the physically correct solution. With this drawback in mind, a totally new approach called *Holomorphic Embedding Load Flow* (*HELM*) [1] was developed by Antonio Trias, described more detailed in Section 2.4.5. This approach guarantees to find a solution for a given load-flow problem if and only if the system is stable. Unfortunately, this method is so far only implemented in *HELM-Flow*[1] by Gridquant. The one tool commonly used in Europe *PSS SINCAL*[2] has not yet implemented this new approach. Therefore, I implemented a tool which can apply *HELM* to a power net stored in the file format of *PSS SINCAL*.

The main results of this thesis are:

- *HELM* has a better convergence behaviour than the iterative methods.
- The theoretically perfect behaviour of *HELM* can only be reached through a trade-off respecting its runtime.

[1] http://www.gridquant.com/solutions/helm-flow/
[2] http://www.simtec.cc/sites/sincal.asp

Load-Flow Calculation

To clarify the situation, I will start with the problem formulation in Section 2.1. Afterwards, I will go into more detail explaining the specific steps which are necessary to model a power net in Section 2.2 and Section 2.3. This chapter concludes in a description of the implemented calculation methods (Section 2.4). In order to make a fair comparison between the iterative methods and *HELM*, I implemented these iterative methods too. This enabled me to circumvent possible optimizations and modifications of the raw methods in their implementations in state-of-the-art tools. In order to guarantee completeness formulas and derivations of the iterative methods in the last section shall be included.

2.1 Problem Formulation

The aim of a load-flow analysis is to determine the voltages in the power net. Any further information, such as currents in connections or critically low voltages, can be derived from these voltages. Therefore, I will consider the problem solved if the voltages are determined.

The first step of a load-flow analysis is the modelling of the net elements, as it is way too complex to use a detailed description of, for instance, a power plant like in Figure 2.1. Therefore, all elements in a power net are modelled through busses, also called nodes, and admittances between them. To simplify the calculations, only single phase nets are being considered. Consequently, three phase systems have to be scaled down by the factor 3, respectively $\sqrt{3}$, to be represented by a single phase system. Asymmetric cases can be modelled with only single phase nets through symmetrical components [6, p. 399]. Besides that, only one voltage level exists in the final model. This means that all admittances, powers and voltages have to be scaled down to this voltage level.

The second step is the actual calculation of the node voltages, which is based on a nodal analysis. As a pure nodal analysis is only capable of current loads, it is extended to support more realistic load models which define, for instance, the power of a node.

Figure 2.1: Sir Adam Beck Hydroelectric Generating Stations (`http://en.wikipedia.org/wiki/Sir_Adam_Beck_Hydroelectric_Generating_Stations`)

2.1.1 Busses

The most basic mathematical description of an electric circuit, which a power net is, would be through admittances Y_{ki} between the nodes k and i, node voltages U_k and branch currents I_k. In this case I will use the element-based version

$$\sum_i Y_{ki} U_i = I_k \tag{2.1}$$

of

$$\underset{\sim}{\mathbf{Y}} \underline{U} = \underline{I}, \tag{2.2}$$

because, eventually, the right hand side of this equation can not be easily described with vector notation.

If the loads and inputs were current-controlled I would be able to stop at this point, solve the equation system and receive the node voltages as a result. Unfortunately, most elements in a power net are defined through power. This power is either fed in, in case of a generator, or drawn, in case of a load. Therefore, I have to extend Equation 2.1 with a term for a constant power $S_k = P_k + jQ_k = U_k I_k^\star$ at the node k and receive

$$\sum_i Y_{ki} U_i = I_k + \frac{S_k^\star}{U_k^\star}, \tag{2.3}$$

which is already the definition of a PQ-bus. As a side remark, a positive power in this formulation indicates that power is fed into the net. Consequently, real loads are modelled by using a negative sign and generators by using a positive one.

Another possible type of bus is a slack bus. The voltage of this bus is defined, therefore I do not have to add a line to the equation system. This kind of bus still occurs in the total equation system as a part of neighbour busses through the branch currents $Y_{ki}U_i$. These known branch currents can be represented on the right hand side of Equation 2.3 through

$$I_k = - \sum_{\text{slack busses}} Y_{ki}U_i. \tag{2.4}$$

Although slack busses appear only in the constant currents of the right hand side, there must always be at least one slack bus in the power net. This bus defines the rotation of the system and compensates mismatches in the total power sum. In practice, a major power plant is typically selected as a slack bus.

The third important type of bus is the PV-bus. At such a node the real power P_k and the voltage magnitude $|U_k|$ are defined. The implementation of this bus type depends on the algorithm, which is used to calculate the missing node voltages. Therefore, I will discuss this in Section 2.4.

2.1.2 Admittance Matrix

The admittance matrix $\underset{\sim}{\mathbf{Y}} = (Y_{ki})$ is filled with the admittances between the nodes. More complex elements, like controlled sources, have to be represented by an equivalent circuit, which is voltage-controlled, so that they can be modelled with an admittance matrix. Not voltage-controlled elements can be transformed through a gyrator, which itself can be modelled through voltage-controlled elements.

During the modelling several kinds of electric elements will be used. I will describe how each of them affects the admittance matrix. Each circuit element is defined by a partial admittance matrix $\underset{\sim}{\mathbf{Y}}_p$. Summing up, all these partial matrices result in the total admittance matrix

$$\underset{\sim}{\mathbf{Y}} = \sum_i \underset{\sim}{\mathbf{Y}}_{p,i}. \tag{2.5}$$

The same superposition applies for the current sources, if there are any:

$$\underline{I} = \sum_i \underline{I}_{p,i}. \tag{2.6}$$

The two most important elements are the admittance, which is part of nearly every model of a net element, and the ideal transformer, which is mainly used for modelling real transformers with non-nominal ratios. The controlled source and the gyrator are only used to model the ideal transformer.

Figure 2.2: Admittance G between the nodes α and β

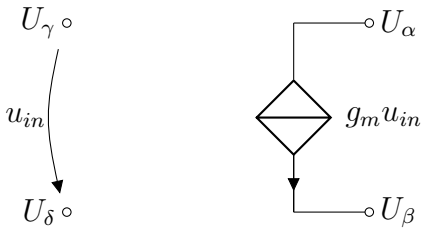

Figure 2.3: Voltage-controlled current source

Admittance

The admittance G between the nodes α and β (Figure 2.2) causes the currents

$$I_\alpha = (U_{k,\alpha} - U_{k,\beta})G \tag{2.7}$$

and

$$I_\beta = (U_{k,\beta} - U_{k,\alpha})G, \tag{2.8}$$

which have to be considered in the admittance matrix through

$$\underset{\sim}{\mathbf{Y}}_p = \begin{array}{c} \\ \\ \alpha \\ \\ \beta \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & \vdots & & \vdots & \\ \cdots & G & \cdots & -G & \cdots \\ & \vdots & & \vdots & \\ \cdots & -G & \cdots & G & \cdots \\ & \vdots & & \vdots & \end{array} \\ \begin{array}{cc} \alpha & \beta \end{array} \end{array} . \tag{2.9}$$

Voltage-Controlled Current Source

The voltage-controlled current source (Figure 2.3) is defined by the two branch currents

$$I_\alpha = (U_{k,\gamma} - U_{k,\delta})g_m \tag{2.10}$$

and

$$I_\beta = (U_{k,\delta} - U_{k,\gamma})g_m. \tag{2.11}$$

The difference to a simple admittance is that in this case the current is controlled by different nodes, which results in the asymmetric

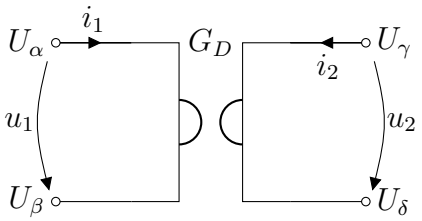

Figure 2.4: Gyrator

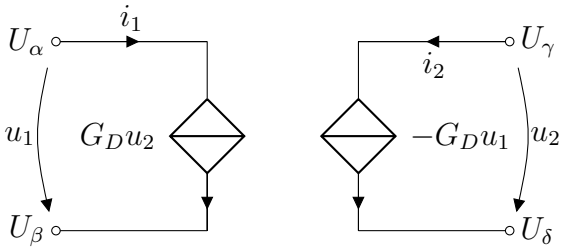

Figure 2.5: Equivalent circuit for a gyrator

admittance matrix

$$\underset{\sim}{\mathbf{Y}}_p = \begin{bmatrix} & \vdots & & \vdots & \\ \cdots & g_m & \cdots & -g_m & \cdots \\ & \vdots & & \vdots & \\ \cdots & -g_m & \cdots & g_m & \cdots \\ & \vdots & & \vdots & \\ & \gamma & & \delta & \end{bmatrix} \begin{matrix} \\ \alpha \\ \\ \beta \\ \\ \end{matrix} . \tag{2.12}$$

Gyrator

The gyrator (Figure 2.4), which is defined by

$$i_1 = G_D u_2 \tag{2.13}$$

and

$$i_2 = -G_D u_1, \tag{2.14}$$

can be replaced by two voltage-controlled current sources as illustrated in Figure 2.5.

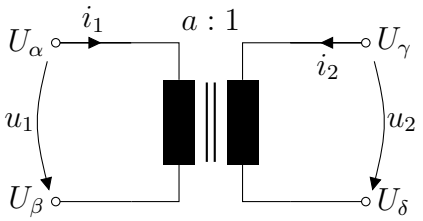

Figure 2.6: Ideal transformer

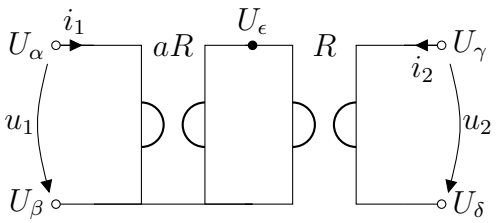

Figure 2.7: Equivalent circuit for an ideal transformer

Ideal Transformer

An ideal transformer (Figure 2.6) can be modelled using the circuit in Figure 2.7, consist of two gyrators. The gyrators themselves have to be replaced by the aforementioned voltage-controlled elements.

In theory, the parameter R in this model can be chosen freely, but for numerical reasons I recommend a scaling of the internal node so that its voltage is in the range of the nominal voltage. To be able to do so, an estimation of the load-flow over the transformer is needed, but it is sufficient to have a very rough estimate. If the scaling is chosen improperly the iterative methods may not converge.

2.2 Scaling

As long as the relations are kept the same, it is possible to change the scale base of all values in a system. For this purpose, from the set of voltage, current, impedance and power, two physical quantities can be chosen at will and the others are determined depending on this decision. This scaling is also referred to as a transformation into a per-unit system [6, p. 90].

For numerical stability, the voltages should be in the range of one, therefore for each voltage level in the system the nominal voltage is selected as the scale base U_B for the voltages. The second degree of freedom can be used to scale the powers down into said range, which can be achieved, for instance, roughly with the power scale base

$$P_B = \frac{1}{2n}\left(\sum_i^n |P_{load,i}| + \sum_i^n |Q_{load,i}|\right). \tag{2.15}$$

The other scale bases are then derived from these two chosen values:

$$I_B = \frac{P_B}{U_B} \tag{2.16}$$

$$Z_B = \frac{1}{Y_B} = \frac{U_B}{I_B} \tag{2.17}$$

The actual scaling is achieved through a division of the values by the scale bases:

$$U_{scaled} = \frac{U}{U_B} \tag{2.18}$$

$$I_{scaled} = \frac{I}{I_B} \tag{2.19}$$

$$P_{scaled} = \frac{P}{P_B} \tag{2.20}$$

$$Q_{scaled} = \frac{Q}{P_B} \tag{2.21}$$

$$Z_{scaled} = \frac{Z}{Z_B} \tag{2.22}$$

$$Y_{scaled} = \frac{Y}{Y_B} \tag{2.23}$$

The two big advantages of this scaling are the maximum numerical range for the voltage values and a simpler model for transformers with a nominal ratio. Because of the scaling these transformers would contain an ideal transformer with a ratio of one, which means that the ideal transformer can be left out.

2.3 Modelling of Net Elements

The power net elements are modelled through admittances and busses. Therefore, I will use the previously discussed equivalent circuits to describe the behaviour of the net elements.

All external nodes, which exist in a power net, are by default PQ-busses with no load, therefore $P = 0$ and $Q = 0$. When two nodes are directly connected with an impedance $Z = 0$, they have to be merged. The direct combination of PQ-busses leads to a summation of the partial inputs (or loads, depending on the sign). On the opposite, the direct connection of a PV-bus to a PQ-bus has the result of that the bus being forced to become a PV-bus with the values $P_{total} = P_{PV} + P_{PQ}$ and $V_{total} = V_{PV}$. The reactive power of the PQ-node is assumed to be provided by the PV-bus. The third type of busses, a slack bus, can also be combined with a PQ-bus. In this case all loads are provided by the slack bus itself. Therefore, the total result is a slack bus.

As it would lead to an overspecified problem, PV-busses must not be connected to slack busses. Theoretically this is possible if the PV-bus has the same voltage magnitude as the slack bus, but in practice this case does not occur and can be neglected.

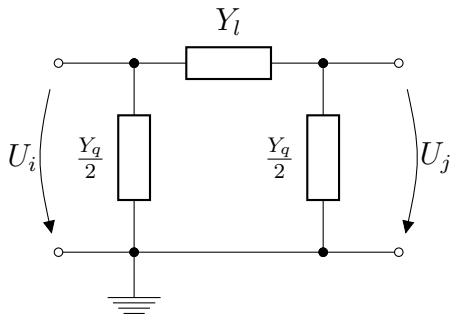

Figure 2.8: Equivalent circuit for a transmission line

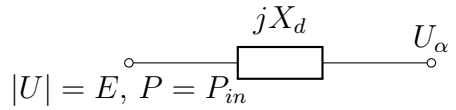

Figure 2.9: Equivalent circuit for a generator

2.3.1 Transmission Line

A transmission line can be modelled through admittances like in Figure 2.8. I can derive the values Y_q and Y_l with the wave impedance Z_W, the propagation constant γ, and the length of the transmission line through

$$Y_l = \frac{1}{Z_W \sinh(\gamma l)} \tag{2.24}$$

and

$$\frac{Y_q}{2} = \frac{1}{Z_W} \tanh\left(\frac{\gamma l}{2}\right), \tag{2.25}$$

as shown in [6, p. 155]. The wave impedance and the propagation constant can be calculated from the electrical characteristics of

$$Z_W = \sqrt{\frac{R' + j\omega L'}{G' + j\omega C'}} \tag{2.26}$$

and

$$\gamma = \sqrt{(R' + j\omega L')\,(G' + j\omega C')}, \tag{2.27}$$

also derived in [6, p. 153].

2.3.2 Load

A load does not affect the admittance matrix; it can be modelled solely through a PQ-bus. If there are several loads connected to one node, their values sum up.

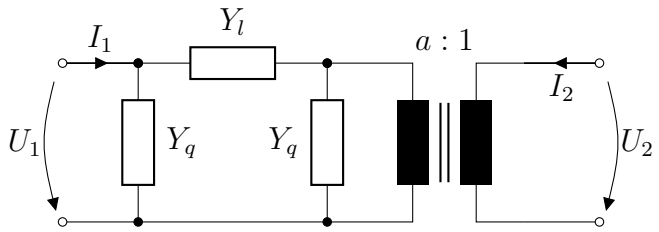

Figure 2.10: Equivalent circuit for a transformer

2.3.3 Generator

Generators are represented by a synchronous reactance X_d, which models internal losses, and a PV-bus [6, p. 55], as shown in Figure 2.9. The voltage magnitude at the internal PV-bus is the excitation voltage E and the real power input is determined by the mechanical power and some transformation losses.

If the synchronous reactance is not zero, the external node α is not forced to become any certain bus type, but if it is zero, the bus is forced to become a PV-bus.

2.3.4 Transformer

To model the transformer I chose to use the equivalent circuit in Figure 2.10 with a π-model and an ideal transformer. As all variables are scaled to the same nominal voltage, the ideal transformer is only needed if the real transmission ratio a is not the nominal transmission ratio

$$a_n = \frac{U_{1n}}{U_{2n}}. \tag{2.28}$$

In this case, the ratio of the ideal transformer is set to the relative ratio

$$a_r = \frac{a}{a_n}. \tag{2.29}$$

For transformers several different ways of specifing their electrical characeristics exist. As an input, I chose to use:

- S_n: nominal power
- $|u_r|$: relative short circuit voltage
- P_{Cu}: copper losses
- P_{Fe}: iron losses
- $\frac{I_0}{I_n}$: relative no-load current

The shunt admittance can then be derived directly from these values through

$$Y_q = \frac{1}{2}\left(P_{Fe} - j\sqrt{\left(\frac{I_0}{I_n}S_n\right)^2 - P_{Fe}^2}\right)\frac{1}{U_{1n}^2}. \tag{2.30}$$

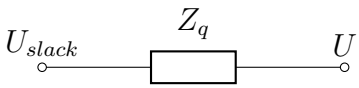

Figure 2.11: Equivalent circuit for a feed-in

For the length admittance it is necessary to calculate the complex relative short circuit voltage u_r. The real part

$$\mathrm{Re}\{u_r\} = \frac{P_{Cu}}{S_n} \tag{2.31}$$

can be calculated from the copper losses and the nominal power. At this point the magnitude and the real part of the relative short circuit voltage are known. Thus, it is possible to calculate the imaginary part

$$\mathrm{Im}\,\{u_r\} = \sqrt{|u_r|^2 - \mathrm{Re}\{u_r\}}. \tag{2.32}$$

Therefore, the total complex relative short circuit voltage

$$u_r = \mathrm{Re}\{u_r\} + j\,\mathrm{Im}\,\{u_r\} \tag{2.33}$$

is also known and the length admittance

$$Y_l = \frac{S_n}{U_{1n}^2 u_r} \tag{2.34}$$

can be determined.

2.3.5 Feed-In

A feed-in (Figure 2.11) is characterized by the voltage at the internal slack bus U_{slack}, the short circuit power S_k, the power factor c and the ratio of the real to the imaginary part $\frac{R}{X}$. From these values the magnitude of the input impedance

$$|Z_q| = c\frac{U_n}{S_k} \tag{2.35}$$

can be derived, which again enables the calculation of

$$X = \frac{\sqrt{\left(\frac{R}{X}\right) + 1}}{|Z_q|} \tag{2.36}$$

and

$$R = \frac{R}{X} \cdot X. \tag{2.37}$$

The combination of these two values gives the input impedance

$$Z_q = R + jX. \tag{2.38}$$

In case of a huge short circuit power compared to the actual load-flow, the input impedance may turn out to have a very small value. If this value is negligible, this results in a direct connection of the internal slack bus with the external node. As a consequence of this case, the external node is overriden and turns into a slack bus.

2.4 Calculation Methods

The task for the following calculation methods is to determine the node voltages from an admittance matrix, a list of PQ- and PV-busses, and a vector of constant currents. In this section, I will describe in total four different algorithms for this problem:

- *Current Iteration* [6, p. 209]
- *Newton-Raphson* [6, p. 232]
- *Fast-decoupled-load-flow* [6, p. 240]
- *Holomorphic Embedding Load Flow* [1, 3, 4]

As an introduction, I will classify these methods based on their properties.

2.4.1 Classification of Methods

The first three methods, the *Current Iteration*, *Newton-Raphson* and the *FDLF*, fall into the category of iterative methods. These algorithms have been well-known for decades now but have two major drawbacks. Firstly, they are iterative and need some seed values for the voltages. This circumstance leads directly to the second drawback: The iterative methods can not guarantee to find a solution and their convergence depends heavily on the initial voltages. It may even happen that the iterative methods produce results which are physically incorrect because they do not represent a stable operating point. This typically happens only for certain constructed power nets. In practice, the main problem is that these methods often do not converge at all, although the power net is in a stable condition.

To circumvent these drawbacks a new approach to the load-flow problem was developed, the *Holomorphic Embedding Load Flow*. This algorithm guarantees to converge in theory if and only if the system is stable. Therefore, *HELM* would be superior to the iterative methods, but it has some practical drawbacks, which I will discuss in Section 4.1.

2.4.2 Current Iteration

The basic problem of the load-flow calculation is that Equation 2.3 can not be explicitly solved. The iterative approach of the Current Iteration works around this problem through a selection of initial voltages and the successive solving of one line of Equation 2.3 after another. To do so, on the left hand side of the equation the current voltage is separated from the sum

$$\sum_{i \neq k} Y_{ki} U_i + Y_{kk} U_k = I_k + \frac{S_k^\star}{U_k^\star} \tag{2.39}$$

and then the rest is moved to the right hand side

$$U_k = \frac{1}{Y_{kk}} \left(I_k + \frac{S_k^\star}{U_k^\star} - \sum_{i \neq k} Y_{ki} U_i \right). \tag{2.40}$$

The resulting equation is, again, not explicitly solved for U_k, as this variable is still found on the right hand side. In this occurence the old value of the previous iteration can be used. This leads to

$$U_k^{(j+1)} = \frac{1}{Y_{kk}} \left(I_k + \frac{S_k^\star}{U_k^{(j)\star}} - \sum_{i \neq k} Y_{ki} U_i^{(j)} \right), \tag{2.41}$$

where the superindex j denotes the iteration step. For the sake of legibility, this formula would cause the node voltages to be updated after every iteration. In fact, they can be updated every time a new value is determined.

Another approach, which is very similar to the aforementioned presented one, is to leave the admittance matrix on the left hand side. By doing so, it is possible to calculate all new voltages in one step by solving

$$\sum_i Y_{ki} U_i^{(j+1)} = I_k + \frac{S_k^\star}{U_k^{(j)\star}}. \tag{2.42}$$

So far, I discussed only the PQ-busses, but PV-busses can be handled too. The PV-bus is considered as a PQ-bus in the first step and afterwards the values are corrected to match the needs of the PV-bus [6, p. 211]. One possible way to calculate the updated voltage U_k' is to combine the specified voltage magnitude $|U_{k,PV}|$ with the newly calculated U_k like

$$U_k' = |U_{k,PV}| e^{j \angle U_k}. \tag{2.43}$$

2.4.3 Newton-Raphson

In general, *Newton-Raphson* is a method of finding roots of a non-linear function. In the area of load-flow calculation the idea is the same: The basic problem is transformed into finding voltages $\underline{x}$ so that the loads driven by these voltages $\underline{S}(\underline{x})$ are the same as the specified loads S_{spec}:

$$\underline{S}(\underline{x}) = S_{spec} \tag{2.44}$$

With a small transformation this leads to the problem of finding the roots of

$$\underline{S}(\underline{x}) - S_{spec} = 0, \tag{2.45}$$

which is exactly what *Newton-Raphson* does. The whole left part is considered as a function

$$\underline{f}(\underline{x}) = \underline{S}(\underline{x}) - S_{spec}, \tag{2.46}$$

which is developed into the Taylor-series

$$\underline{f}(\underline{x}) = \sum_{k=0}^{\infty} \frac{\underline{f}^{(k)}(\underline{x}_0)}{k!} (\underline{x} - \underline{x}_0) . \tag{2.47}$$

From this series only the linear term is used as an approximation, which leads to

$$\underline{f}(\underline{x}) \approx \underline{f}(\underline{x}_0) + \underline{f}'(\underline{x}_0) (\underline{x} - \underline{x}_0) . \tag{2.48}$$

As I want to find the root of $\underline{f}(\underline{x})$, I set the approximation to 0 and replace the occurence of $\underline{f}(\underline{x})$ with its definition in the result

$$0 = \underline{f}(\underline{x}^{(k)}) + \underline{f}'(\underline{x}^{(k)}) \left(\underline{x}^{(k+1)} - \underline{x}^{(k)}\right) \tag{2.49}$$

to get

$$\underset{\sim}{\mathbf{S}}'(\underline{x}^{(k)}) \left(\underline{x}^{(k+1)} - \underline{x}^{(k)}\right) = \underline{S}(\underline{x}^{(k)}) - \underline{S}_{spec}. \tag{2.50}$$

In this formula we have as a matrix the derivative of the power function with respect to the voltages $\underset{\sim}{\mathbf{S}}'(\underline{x}_k)$, the voltage changes $\Delta\underline{x}^{(k)} = \left(\underline{x}^{(k+1)} - \underline{x}^{(k)}\right)$ on the left side and on the right side the current power mismatch

$$\Delta\underline{S}(\underline{x}^{(k)}) = \underline{S}(\underline{x}^{(k)}) - \underline{S}_{spec}. \tag{2.51}$$

The so far unsolved question how to represent the voltages, as they are complex variables. The two possible approaches are a representation of magnitude and phase and a representation of real and imaginary parts. As the version with magnitudes and phases is more common, I shall start with this one.

Magnitude and Phase

In real power nets the voltage magnitude $|U_i|$ of a node mainly depends on the reactive power and the angle of the voltage δ_i on the real power. Consequently, voltages are usually represented in polar coordinates. If I consider n PQ-busses and m PV-busses, I have as voltage vector

$$\underline{x} = \begin{bmatrix} |U_1| \\ \vdots \\ |U_n| \\ \delta_1 \\ \vdots \\ \delta_n \\ \delta_{n+1} \\ \vdots \\ \delta_{n+m} \end{bmatrix} = \begin{bmatrix} |U_1| \\ \vdots \\ |U_n| \\ \delta_1 \\ \vdots \\ \delta_{n+m} \end{bmatrix} \tag{2.52}$$

and for the specified powers

$$\underline{S}_{spec} = \begin{bmatrix} P_1 \\ \vdots \\ P_n \\ P_{n+1} \\ \vdots \\ P_{n+m} \\ Q_1 \\ \vdots \\ Q_n \end{bmatrix} = \begin{bmatrix} P_1 \\ \vdots \\ P_{n+m} \\ Q_1 \\ \vdots \\ Q_n \end{bmatrix}. \tag{2.53}$$

At this point in order to be able to apply *Newton-Raphson*, I need the derivative of the power function with respect to the entries in the voltage change vector. I will start with a separation of the power at each node into its real and imaginary part and then derive these functions with respect to the voltage magnitudes and angles.

As a first step, Equation 2.3 is transformed into an explicit calculation of the power at the bus k

$$S_k = \left(U_k^\star \left(\sum_{i \neq k} Y_{ki} U_i + Y_{kk} U_k - I_k \right) \right)^\star \tag{2.54}$$

$$= U_k \left(\sum_{i \neq k} Y_{ki}^\star U_i^\star + Y_{kk}^\star U_k^\star - I_k^\star \right) \tag{2.55}$$

$$= \sum_{i \neq k} U_k Y_{ki}^\star U_i^\star + Y_{kk}^\star |U_k|^2 - I_k^\star U_k. \tag{2.56}$$

Next, the variables are split up into magnitude and angle ($U_k = |U_k| e^{j\delta_k}$, $I_k = |I_k| e^{j\gamma_k}$, $Y_{ki} = |Y_{ki}| e^{j\theta_{ki}}$), which leads to the expression

$$\begin{aligned} S_k =& \sum_{i \neq k} |U_k| e^{j\delta_k} |Y_{ki}| e^{-j\theta_{ki}} |U_i| e^{-j\delta_i} + |Y_{kk}| e^{-j\theta_{kk}} |U_k|^2 \\ & - |I_k| e^{-j\gamma_k} |U_k| e^{j\delta_k} \\ =& \sum_{i \neq k} |U_k| |Y_{ki}| |U_i| e^{j(\delta_k - \theta_{ki} - \delta_i)} + |Y_{kk}| |U_k|^2 e^{-j\theta_{kk}} \\ & - |I_k| |U_k| e^{j(\delta_k - \gamma_k)} \end{aligned} \tag{2.57}$$

for the load at bus k. This load can be separated into its real and imaginary parts

$$S_k = P_k + jQ_k, \tag{2.58}$$

as well as the part on the right hand side of Equation 2.57 to receive

$$\begin{aligned} P_k =& \sum_{i \neq k} |U_k| |Y_{ki}| |U_i| \cos(\delta_k - \theta_{ki} - \delta_i) \\ & + |Y_{kk}| |U_k|^2 \cos(\theta_{kk}) - |I_k| |U_k| \cos(\delta_k - \gamma_k) \end{aligned} \tag{2.59}$$

and

$$\begin{aligned} Q_k =& \sum_{i \neq k} |U_k| |Y_{ki}| |U_i| \sin(\delta_k - \theta_{ki} - \delta_i) \\ & - |Y_{kk}| |U_k|^2 \sin(\theta_{kk}) - |I_k| |U_k| \sin(\delta_k - \gamma_k). \end{aligned} \tag{2.60}$$

I can then differentiate these formulas with respect to U_k, U_i $(i \neq k)$, δ_k and δ_i $(i \neq k)$:

$$\begin{aligned}\frac{\partial P_k}{\partial |U_k|} =& \sum_{i \neq k} |Y_{ki}||U_i| \cos\left(\delta_k - \theta_{ki} - \delta_i\right) + 2|Y_{kk}||U_k| \cos\left(\theta_{kk}\right) \\ & - |I_k| \cos\left(\delta_k - \gamma_k\right)\end{aligned} \tag{2.61}$$

$$\begin{aligned}\frac{\partial Q_k}{\partial |U_k|} =& \sum_{i \neq k} |Y_{ki}||U_i| \sin\left(\delta_k - \theta_{ki} - \delta_i\right) - 2|Y_{kk}||U_k| \sin\left(\theta_{kk}\right) \\ & - |I_k| \sin\left(\delta_k - \gamma_k\right)\end{aligned} \tag{2.62}$$

$$\frac{\partial P_k}{\partial |U_i|} = |U_k||Y_{ki}| \cos\left(\delta_k - \theta_{ki} - \delta_i\right) \tag{2.63}$$

$$\frac{\partial Q_k}{\partial |U_i|} = |U_k||Y_{ki}| \sin\left(\delta_k - \theta_{ki} - \delta_i\right) \tag{2.64}$$

$$\frac{\partial P_k}{\partial \delta_k} = -\sum_{i \neq k} |U_k||Y_{ki}||U_i| \sin\left(\delta_k - \theta_{ki} - \delta_i\right) + |I_k||U_k| \sin\left(\delta_k - \gamma_k\right) \tag{2.65}$$

$$\frac{\partial Q_k}{\partial \delta_k} = \sum_{i \neq k} |U_k||Y_{ki}||U_i| \cos\left(\delta_k - \theta_{ki} - \delta_i\right) - |I_k||U_k| \cos\left(\delta_k - \gamma_k\right) \tag{2.66}$$

$$\frac{\partial P_k}{\partial \delta_i} = |U_k||Y_{ki}||U_i| \sin\left(\delta_k - \theta_{ki} - \delta_i\right) \tag{2.67}$$

$$\frac{\partial Q_k}{\partial \delta_i} = -|U_k||Y_{ki}||U_i| \cos\left(\delta_k - \theta_{ki} - \delta_i\right) \tag{2.68}$$

With these derivatives I can calculate the elements of the Jacobian matrix

$$\underset{\sim}{\mathbf{S}}_1'(\underline{x}^{(k)}) = \begin{bmatrix} \frac{\partial P_1}{\partial |U_1|} & \cdots & \frac{\partial P_1}{\partial |U_n|} & \frac{\partial P_1}{\partial \delta_1} & \cdots & \frac{\partial P_1}{\partial \delta_{n+m}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_{n+m}}{\partial |U_1|} & \cdots & \frac{\partial P_{n+m}}{\partial |U_n|} & \frac{\partial P_{n+m}}{\partial \delta_1} & \cdots & \frac{\partial P_{n+m}}{\partial \delta_{n+m}} \\ \frac{\partial Q_1}{\partial |U_1|} & \cdots & \frac{\partial Q_1}{\partial |U_n|} & \frac{\partial Q_1}{\partial \delta_1} & \cdots & \frac{\partial Q_1}{\partial \delta_{n+m}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial |U_1|} & \cdots & \frac{\partial Q_n}{\partial |U_n|} & \frac{\partial Q_n}{\partial \delta_1} & \cdots & \frac{\partial Q_n}{\partial \delta_{n+m}} \end{bmatrix}. \tag{2.69}$$

Real and Imaginary Part

For the representation of the voltages through real and imaginary parts, one problem remains: PV-busses. For these busses the voltage magnitude is already set and I only have to calculate the phases. Therefore, these busses will still be represented in polar coordinates, but all PQ-busses can be described in a cartesian notation. The advantage of this notation is less computational complexity in the formulas, as I can avoid the sines and cosines which occured in the derivatives of P and Q. For huge load-flow problems this change has

a significant impact on the performance of the overall algorithm, although I can only use this improvement for PQ-busses.

Through a representation in cartesian coordinates I end up with the voltage vector

$$\underline{x} = \begin{bmatrix} U_1^r \\ \vdots \\ U_n^r \\ U_1^i \\ \vdots \\ U_n^i \\ \delta_{n+k} \\ \vdots \\ \delta_{n+m} \end{bmatrix}, \tag{2.70}$$

where the entries $U_k^r = \mathrm{Re}\{U_k\}$ and $U_k^i = \mathrm{Im}\{U_k\}$ are the real and imaginary parts of the complex node voltages of the n PQ-busses, and δ_{n+k} are the voltage phases of the m PV-busses.

In this case, for the Jacobian matrix I need the derivatives of P_k and Q_k with respect to U_{kr} and U_{ki}. To accomplish this I will start with Equation 2.56 and substitute the voltages, admittances and currents with their cartesian representations $Y_{ki}^r = \mathrm{Re}\{Y_{ki}\}$, $Y_{ki}^i = \mathrm{Im}\{Y_{ki}\}$, $I_k^r = \mathrm{Re}\{I_k\}$ and $I_k^i = \mathrm{Im}\{I_k\}$ to end up with

$$\begin{aligned} S_k = &\sum_{i \neq k} (U_k^r + jU_k^i)(Y_{ki}^r - jY_{ki}^i)(U_i^r - jU_i^i) + \\ &(Y_{kk}^r - jY_{kk}^i)((U_k^r)^2 + (U_k^i)^2) - (I_k^r - jI_k^i)(U_k^r + jU_k^i) \end{aligned}. \tag{2.71}$$

As a next step, I will need the separation of this into its real and imaginary parts. Therefore, I expand the summands

$$\begin{aligned} S_k = &\sum_{i \neq k} \left(U_k^r Y_{ki}^r U_i^r + U_k^i Y_{ki}^i U_i^r + U_k^i Y_{ki}^r U_i^i - U_k^r Y_{ki}^i U_i^i \right) + \\ &\sum_{i \neq k} j \left(U_k^i Y_{ki}^r U_i^r - U_k^r Y_{ki}^i U_i^r - U_k^r Y_{ki}^r U_i^i - U_k^i Y_{ki}^i U_i^i \right) + \\ &Y_{kk}^r((U_k^r)^2 + (U_k^i)^2) - jY_{kk}^i((U_k^r)^2 + (U_k^i)^2) - \\ &I_k^r U_k^r - I_k^i U_k^i + j(I_k^i U_k^r - I_k^r U_k^i) \end{aligned} \tag{2.72}$$

and separate this to get

$$\begin{aligned} P_k = &\sum_{i \neq k} \left(U_k^r Y_{ki}^r U_i^r + U_k^i Y_{ki}^i U_i^r + U_k^i Y_{ki}^r U_i^i - U_k^r Y_{ki}^i U_i^i \right) + \\ &Y_{kk}^r((U_k^r)^2 + (U_k^i)^2) - I_k^r U_k^r - I_k^i U_k^i \end{aligned} \tag{2.73}$$

and

$$\begin{aligned} Q_k = &\sum_{i \neq k} \left(U_k^i Y_{ki}^r U_i^r - U_k^r Y_{ki}^i U_i^r - U_k^r Y_{ki}^r U_i^i - U_k^i Y_{ki}^i U_i^i \right) - \\ &Y_{kk}^i((U_k^r)^2 + (U_k^i)^2) + I_k^i U_k^r - I_k^r U_k^i \end{aligned}. \tag{2.74}$$

These formulas must be differentiated with respect to U_k^r, U_k^i, U_i^r $(i \neq k)$ and U_i^i $(i \neq k)$:

$$\frac{\partial P_k}{\partial U_k^r} = \sum_{i \neq k} \left(Y_{ki}^r U_i^r - Y_{ki}^i U_i^i \right) + 2 Y_{kk}^r U_k^r - I_k^r \tag{2.75}$$

$$\frac{\partial P_k}{\partial U_k^i} = \sum_{i \neq k} \left(Y_{ki}^i U_i^r + Y_{ki}^r U_i^i \right) + 2 Y_{kk}^r U_k^i - I_k^i \tag{2.76}$$

$$\frac{\partial Q_k}{\partial U_k^r} = \sum_{i \neq k} \left(-Y_{ki}^i U_i^r - Y_{ki}^r U_i^i \right) - 2 Y_{kk}^i U_k^r + I_k^i \tag{2.77}$$

$$\frac{\partial Q_k}{\partial U_k^i} = \sum_{i \neq k} \left(Y_{ki}^r U_i^r - Y_{ki}^i U_i^i \right) - 2 Y_{kk}^i U_k^i - I_k^r \tag{2.78}$$

$$\frac{\partial P_k}{\partial U_i^r} = U_k^r Y_{ki}^r + U_k^i Y_{ki}^i \tag{2.79}$$

$$\frac{\partial P_k}{\partial U_i^i} = U_k^i Y_{ki}^r - U_k^r Y_{ki}^i \tag{2.80}$$

$$\frac{\partial Q_k}{\partial U_i^r} = U_k^i Y_{ki}^r - U_k^r Y_{ki}^i \tag{2.81}$$

$$\frac{\partial Q_k}{\partial U_i^i} = -U_k^r Y_{ki}^r - U_k^i Y_{ki}^i \tag{2.82}$$

This, in summary, is then called the Jacobian matrix

$$\underset{\sim}{\mathbf{S}}_2'(\underline{x}^{(k)}) = \begin{bmatrix} \frac{\partial P_1}{\partial U_1^r} & \cdots & \frac{\partial P_1}{\partial U_n^r} & \frac{\partial P_1}{\partial U_1^i} & \cdots & \frac{\partial P_1}{\partial U_n^i} & \frac{\partial P_1}{\partial \delta_{n+1}} & \cdots & \frac{\partial P_1}{\partial \delta_{n+m}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_{n+m}}{\partial U_1^r} & \cdots & \frac{\partial P_{n+m}}{\partial U_n^r} & \frac{\partial P_{n+m}}{\partial U_1^i} & \cdots & \frac{\partial P_{n+m}}{\partial U_n^i} & \frac{\partial P_{n+m}}{\partial \delta_{n+1}} & \cdots & \frac{\partial P_{n+m}}{\partial \delta_{n+m}} \\ \frac{\partial Q_1}{\partial U_1^r} & \cdots & \frac{\partial Q_1}{\partial U_n^r} & \frac{\partial Q_1}{\partial U_1^i} & \cdots & \frac{\partial Q_1}{\partial U_n^i} & \frac{\partial Q_1}{\partial \delta_{n+1}} & \cdots & \frac{\partial Q_1}{\partial \delta_{n+m}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial U_1^r} & \cdots & \frac{\partial Q_n}{\partial U_n^r} & \frac{\partial Q_n}{\partial U_1^i} & \cdots & \frac{\partial Q_n}{\partial U_n^i} & \frac{\partial Q_n}{\partial \delta_{n+1}} & \cdots & \frac{\partial Q_n}{\partial \delta_{n+m}} \end{bmatrix}. \tag{2.83}$$

Summary

In summary, the iterative process to improve the voltages is:

1. Calculate the current power mismatch $\Delta \underline{S}(\underline{x}^{(k)})$ with Equation 2.51
2. Calculate the Jacobian matrix $\underline{S}'(\underline{x}^{(k)})$ with Equation 2.69 or Equation 2.83
3. Calculate the voltage changes $\Delta \underline{x}^{(k)}$ through solving the linear equation system in Equation 2.50
4. Calculate the improved voltages through $\underline{x}^{(k+1)} = \underline{x}^{(k)} + \Delta \underline{x}^{(k)}$
5. Repeat these steps until the voltage change is small enough *or* until the power mismatch is small enough

2.4.4 Fast-decoupled-load-flow

The *FDLF* [10] is a modification of the *Newton-Raphson* method. This modification is based on a previously discussed observation: The change in voltage magnitudes is mostly driven by the reactive power flow and the change in the voltage angles by the real power flow. With this circumstance in mind I can neglect two quarters of the Jacobian matrix in Equation 2.69 and reduce it to

$$\underset{\sim}{\mathbf{S}}'(\underline{x}^{(k)}) \approx \begin{bmatrix} 0 & \dots & 0 & \frac{\partial P_1}{\partial \delta_1} & \dots & \frac{\partial P_1}{\partial \delta_{n+m}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \frac{\partial P_{n+m}}{\partial \delta_1} & \dots & \frac{\partial P_{n+m}}{\partial \delta_{n+m}} \\ \frac{\partial Q_1}{\partial |U_1|} & \dots & \frac{\partial Q_1}{\partial |U_n|} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial |U_1|} & \dots & \frac{\partial Q_n}{\partial |U_n|} & 0 & \dots & 0 \end{bmatrix} = \begin{bmatrix} 0 & \frac{\partial \underline{P}}{\partial \underline{\delta}} \\ \frac{\partial \underline{Q}}{\partial \underline{|U|}} & 0 \end{bmatrix}. \tag{2.84}$$

With this simplified system matrix I can separate the equation system in Equation 2.50 into

$$\frac{\partial \underline{P}}{\partial \underline{\delta}} \cdot \Delta \underline{\delta}^{(k)} = \Delta \underline{P}^{(k)} \tag{2.85}$$

and

$$\frac{\partial \underline{Q}}{\partial \underline{|U|}} \cdot \Delta \underline{|U|}^{(k)} = \Delta \underline{Q}^{(k)}. \tag{2.86}$$

The steps for solving the load-flow problem are nearly the same as for the *Newton-Raphson*. The only difference lies within the smaller linear equation system that has to be solved. This is already the reason why one may choose the *FDLF* instead of *Newton-Raphson*: the calculation is faster. The disadvantage of this simplification is a worse convergence behaviour. Consequently, the *FDLF* is less probable to be actually capable of solving a load-flow problem.

2.4.5 Holomorphic Embedding Load Flow

HELM is a comparatively young algorithm dealing with the load-flow problem, as it was developed in 2012 by Antonio Trias [1]. The basic idea is to replace the voltages with voltage functions and their respective Laurent series. Through this approach, it is possible to guarantee that the algorithm will converge if and only if the power net is stable. With this advantage in mind, I will cover the theory behind *HELM* in the upcoming section, the practical implications will be discussed later in Section 4.

Calculation of Coefficients

The starting point for *HELM* is Equation 2.3, where the slack busses are considered through constant currents. Because of its structure it is not possible to explicitly solve these equations, as mentioned

before. Therefore, the voltages U_i are replaced with the voltage functions $U_i(s)$. As these functions are constrained to be holomorph [4] Equation 2.3 is extended to

$$\sum_i Y_{ki} U_i(s) = sI_k + \frac{sS_k^\star}{U_k^\star(s^\star)} + (1-s)Y_k. \tag{2.87}$$

The additional parameter s is embedded in such a way that setting this parameter to one would result in the original formulation. Consequently, the target is to evaluate the functions $U_i(s)$ at the point $s = 1$.

The additional term $(1-s)Y_{total}$ vanishes at the point $s = 1$ and is needed only to avoid that the first coefficients of a Laurent series become zero. In fact, the value Y_{total} can be chosen arbitrarily. [4] suggests the expression

$$Y_k = \sum_i Y_{ki}, \tag{2.88}$$

where the sum is calculated over all nodes, including the slack busses. This approach turns out to work well for practical problems, although in certain scenarios this may still produce coefficients with the value zero. In these cases a random modification to this term can be applied to force the coefficient to a value different from zero.

To get an explicit formula for the functions $U_i(s)$ I replace them with their Laurent series

$$U_i(s) = \sum_{j=-\infty}^{\infty} c_{i,j}(s - s_0)^j. \tag{2.89}$$

As the functions are holomorph, the prinicipal part of the series vanishes, which turns the series into

$$U_i(s) = \sum_{j=0}^{\infty} c_{i,j}(s - s_0)^j. \tag{2.90}$$

To be able to calculate the coefficients I develop these series around the point $s_0 = 0$ and insert

$$U_i(s) = \sum_{j=0}^{\infty} c_{i,j} s^j. \tag{2.91}$$

into Equation 2.87 to get

$$\sum_i \left(Y_{ki} \sum_{j=0}^{\infty} c_{i,j} s^j \right) = sI_k + \frac{sS_k^\star}{\sum_{j=0}^{\infty} c_{k,j}^\star s^j} + (1-s)Y_k. \tag{2.92}$$

At this point I would actually like to make a coefficient comparison, but on the right hand side the inverse of a series is still given. To circumvent this, I need a power series for the inverse of a series. I can get the series

$$W_i(s) = \sum_{j=0}^{\infty} w_{i,j} s^j, \tag{2.93}$$

which satisifies

$$1 = W_i(s)U_i^\star(s^\star) \tag{2.94}$$

through a coefficient comparison of

$$1 = \left(\sum_{j=0}^{\infty} w_{i,j}s^j\right)\left(\sum_{l=0}^{\infty} c_{i,l}^\star s^l\right). \tag{2.95}$$

To do so, the equation is rewritten with the discrete convolution formula to

$$1 = \sum_{j=0}^{\infty}\left(\sum_{l=0}^{j} w_{i,l}c_{i,j-l}^\star\right)s^j, \tag{2.96}$$

where I have to consider the power zero and the rest separately. For s^0 the outcome of the coefficient comparison is

$$1 = w_{i,0}c_{i,0}^\star, \tag{2.97}$$

from which I can derive the first inverse coefficient to be

$$w_{i,0} = \frac{1}{c_{i,0}^\star}. \tag{2.98}$$

For all other powers of s the coefficient comparison delivers

$$0 = \sum_{l=0}^{j} w_{i,l}c_{i,j-l}^\star, \tag{2.99}$$

where I can assume that the previous inverse coefficients are already known. Hence, I can extract the last summand

$$0 = \sum_{l=0}^{j-1} w_{i,l}c_{i,j-l}^\star + w_{i,j}c_{j,0}^\star \tag{2.100}$$

and transform the result into the explicit formula

$$w_{i,j} = -\frac{\sum_{l=0}^{j-1} w_{i,l}c_{i,j-l}^\star}{c_{i,0}^\star}. \tag{2.101}$$

With the power series of the inverse voltage functions I am able to reformulate Equation 2.92 into

$$\sum_i\left(Y_{ki}\sum_{j=0}^{\infty} c_{i,j}s^j\right) = sI_k + sS_k^\star\sum_{j=0}^{\infty} w_{i,j}s^j + (1-s)Y_k. \tag{2.102}$$

Here, I again apply a coefficient comparison. For the power s^0 I get the linear equation system

$$\sum_i Y_{ki}c_{i,0} = Y_k, \tag{2.103}$$

which I can solve to get the coefficients $c_{i,0}$. The second coefficients $c_{i,1}$ can be calculated with

$$\sum_i Y_{ki}c_{i,1} = I_k + S_k^\star w_{k,0} - Y_k, \tag{2.104}$$

as the coefficients $w_{i,j}$ depend only on the previous coefficients $c_{i,j}$. All other coefficients can be determined through

$$\sum_i Y_{ki}c_{i,j} = S_k^\star w_{k,j-1}. \tag{2.105}$$

PV-busses The modelling of PV-busses in *HELM* was first mentioned in [2]. As the notation and handling of slack busses is different to the formulas in this thesis, I will reformulate them to fit to the other equations here.

Starting with the definition of a PQ-bus (Equation 2.3) I transform this equation into the explicit definition of the power

$$S_k = U_k \left(\sum_i Y_{ki} U_i - I_k \right)^\star . \tag{2.106}$$

For the PV-bus only the real power is defined, therefore I do not have to consider the reactive power and can reduce this equation to

$$P_k = \mathrm{Re}\{U_k \left(\sum_i Y_{ki}^\star U_i^\star - I_k^\star \right)\}. \tag{2.107}$$

This way I end up with only one equation for the bus, although I introduce two unknowns. Thus, I can not choose the direct approach (Equation 2.107), but have to use

$$2P_k = S_k + S_k^\star \tag{2.108}$$

instead. Inserting the definition of the bus power (Equation 2.106) results in

$$\begin{aligned} 2P_k &= U_k \left(\sum_i Y_{ki} U_i - I_k \right)^\star + \left(U_k \left(\sum_i Y_{ki} U_i - I_k \right)^\star \right)^\star \\ &= U_k \left(\sum_i Y_{ki} U_i - I_k \right)^\star + U_k^\star \left(\sum_i Y_{ki} U_i - I_k \right) \end{aligned} . \tag{2.109}$$

Multiplying this with U_k gives

$$\begin{aligned} 2P_k U_k &= U_k^2 \left(\sum_i Y_{ki} U_i - I_k \right)^\star + |U_k|^2 \left(\sum_i Y_{ki} U_i - I_k \right) \\ &= U_k^2 \sum_i Y_{ki}^\star U_i^\star - U_k^2 I_k^\star + |U_k|^2 \sum_i Y_{ki} U_i - |U_k|^2 I_k \end{aligned} , \tag{2.110}$$

where I have introduced the voltage magnitude

$$|U_k| = \sqrt{U_k U_k^\star}, \tag{2.111}$$

which is also defined at the PV-bus. Through moving a few summands to the other side of the equation I get

$$|U_k|^2 \sum_i Y_{ki} U_i = 2P_k U_k - U_k^2 \sum_i Y_{ki}^\star U_i^\star + U_k^2 I_k^\star + |U_k|^2 I_k, \tag{2.112}$$

where I can embed a complex parameter s, so that the voltages U_i turn into holomorphic voltage functions $U_i(s)$. This way I end up with

$$\begin{aligned} |U_k|^2 \sum_i Y_{ki} U_i(s) =& s 2P_k U_k(s) - s U_k^2(s) \sum_i Y_{ki}^\star U_i^\star(s) \\ &+ s U_k^2(s) I_k^\star + |U_k|^2 I_k + (1-s)|U_k|^2 Y_k \end{aligned} , \tag{2.113}$$

which, after the division by $|U_k|^2$, yields

$$\begin{aligned}\sum_i Y_{ki}U_i(s) =& s\frac{2P_k}{|U_k|^2}U_k(s) - s\frac{1}{|U_k|^2}U_k^2(s)\sum_i Y_{ki}^\star U_i^\star(s) \\ &+ sU_k^2(s)\frac{I_k^\star}{|U_k|^2} + I_k + (1-s)Y_k\end{aligned}. \tag{2.114}$$

This line has to be inserted into the equation system for a PV-bus. The total system contains lines in the form of Equation 2.114 and Equation 2.87. In fact, the only difference is the right hand side of the equation, which I want to abbreviate with

$$\begin{aligned}F_{PV}(s) =& s\frac{2P_k}{|U_k|^2}U_k - s\frac{1}{|U_k|^2}U_k^2\sum_i Y_{ki}^\star U_i^\star \\ &+ sU_k^2\frac{I_k^\star}{|U_k|^2} + I_k + (1-s)Y_k\end{aligned}. \tag{2.115}$$

Consequently, the counter part for a PQ-bus is

$$\begin{aligned}F_{PQ}(s) &= sI_k + \frac{sS_k^\star}{U_k^\star(s^\star)} + (1-s)Y_k \\ &= sI_k + sS_k^\star W_k(s) + (1-s)Y_k\end{aligned}. \tag{2.116}$$

It is only necessary to calculate the elements of the right hand side for the next set of coefficients. The applicable formula is determined by the type of the bus.

For the formulation of the busses it is important that all elements in $F_{PQ}(s)$ and $F_{PV}(s)$ are either constant or depend on previous coefficients. This is ensured by the embedding, where all summands that are functions in the parameter s are multiplied with s, which shifts the coefficients of the series.

Next, I will go into more details for $F_{PV}(s)$. First, it contains the summand

$$U_k^2\sum_i Y_{ki}^\star U_i^\star, \tag{2.117}$$

which, for the implementation, is split up like

$$U_k^2\sum_{i\neq k} Y_{ki}^\star U_i^\star + U_k^2 U_k^\star = U_k^2\sum_{i\neq k} Y_{ki}^\star U_i^\star + |U_k|^2 U_k. \tag{2.118}$$

The problem here lies within the first part, which contains the multiplication of three power series. The multiplication of the two power series $\sum_i a_i s^i$ and $\sum_i b_i s^i$ results in the power series $\sum_i c_i s^i$ with the coefficients

$$c_i = \sum_{j=0}^{i} a_j b_{n-j}. \tag{2.119}$$

The multiplication of three power series can be built upon this formula.

To simplify Equation 2.115 I start with the two substitutions

$$V_k(s) = U_k(s)^2 = \sum_j v_{k,j}s^j \tag{2.120}$$

and

$$X_k(s) = U_k(s)^2 \sum_{i \neq k} Y_{ki}^\star U_i(s)^\star = \sum_j x_{k,j} s^j. \tag{2.121}$$

The coefficients for $V(s)$ can be calculated with the convolution formula

$$v_{k,j} = \sum_{l=0}^{j} c_{k,l} c_{k,j-l}, \tag{2.122}$$

and for $X(s)$ I can convolute and weigh these coefficients to calculate

$$x_{k,j} = \sum_{i \neq k} Y_{ki}^\star \sum_{l=0}^{j} v_{i,l} c_{i,j-l}. \tag{2.123}$$

With these two substitutions, Equation 2.115 can be simplified to

$$\begin{aligned} F_{PV}(s) =& s\frac{2P_k}{|U_k|^2} U_k(s) - s\frac{1}{|U_k|^2} X_k(s) - sU_k(s) \\ &+ s\frac{I_k^\star}{|U_k|^2} V_k(s) + I_k + (1-s)Y_k \end{aligned} . \tag{2.124}$$

For the coefficient comparison, I need the coefficients of $F_{PV}(s)$ for certain powers of s. The first power s^0 is obtained by

$$\sum_i Y_{ki} c_{i,0} = I_k + Y_k, \tag{2.125}$$

the second one by

$$\sum_i Y_{ki} c_{i,1} = \frac{2P_k}{|U_k|^2} c_{k,0} - \frac{1}{|U_k|^2} x_{k,0} - c_{k,0} + \frac{I_k^\star}{|U_k|^2} v_{k,0} - Y_k \tag{2.126}$$

and all other coefficients by

$$\sum_i Y_{ki} c_{i,n} = \frac{2P_k}{|U_k|^2} c_{k,n-1} - \frac{1}{|U_k|^2} x_{k,n-1} - c_{k,n-1} + \frac{I_k^\star}{|U_k|^2} v_{k,n-1}. \tag{2.127}$$

At this point, I have introduced all formulas necessary for the calculation of the coefficients up to a certain index. The next step is the analytic continuation which evaluates the voltage functions at a certain point.

Analytic Continuation with Wynn's Epsilon Algorithm

Unfortunately, the convergence radius of the previously calculated Laurent series is too small to be able to evalute it at $s = 1$. Accordingly, a method for an analytic continuation has to be applied. [4] suggests the method of *Viskovatov*, but I achieved better results with *Wynn's epsilon algorithm* [5]. This method is basically a non-linear sequence transformation, which is applied to the m partial

sums of the series. Therefore, I have to calculate the partial sums of the voltage functions

$$U_i[n] = \sum_{k=0}^{n} c_{i,k}. \tag{2.128}$$

The algorithm is then initialized with these partial sums

$$\epsilon_0^{(n)} = U_i[n] \tag{2.129}$$

and

$$\epsilon_{-1}^{(n)} = 0. \tag{2.130}$$

The other values of the tableau

$$\begin{array}{cccccccc}
\epsilon_0^{(0)} & \epsilon_1^{(0)} & \epsilon_2^{(0)} & \dots & \epsilon_{m-3}^{(0)} & \epsilon_{m-2}^{(0)} & \epsilon_{m-1}^{(0)} \\
\epsilon_0^{(1)} & \epsilon_1^{(1)} & \epsilon_2^{(1)} & \dots & \epsilon_{m-3}^{(1)} & \epsilon_{m-2}^{(1)} & \\
\epsilon_0^{(2)} & \epsilon_1^{(2)} & \epsilon_2^{(2)} & \dots & \epsilon_{m-3}^{(2)} & & \\
\vdots & \vdots & \vdots & \iddots & & & \\
\epsilon_0^{(m-3)} & \epsilon_1^{(m-3)} & \epsilon_2^{(m-3)} & & & & \\
\epsilon_0^{(m-2)} & \epsilon_1^{(m-2)} & & & & & \\
\epsilon_0^{(m-1)} & & & & & &
\end{array} \tag{2.131}$$

are calculated according to

$$\epsilon_{k+1}^{(n)} = \epsilon_{k-1}^{(n+1)} + \frac{1}{\epsilon_k^{(n+1)} - \epsilon_k^{(n)}}. \tag{2.132}$$

The most accurate value for the total sum can then be found in the end of the last even column of Equation 2.131, as the odd columns diverge and the even columns converge. This divergence is actually a big problem in *HELM*, because the diverging columns explode so fast that even the mantissa of a 64-bit floating point datatype (*double*) is not big enough for more than 50 coefficients. If more coefficients are necessary for an accurate result, the floating point datatype has to be exchanged, which is an expensive modification in terms of runtime.

A nice feature of this algorithm for its practical application is that additional coefficients only cause the tableau to grow downwards. Consequently, there is no real additional effort to calculate partial results, which can be used for instance in a termination criteria. Keeping the direction of growth in mind, it is possible to optimize the calculation even further. In fact, only the last two values of each column have to be stored. All previous results can be discarded, as they are not being reused. This optimization reduces the memory footprint of the total algorithm and can even avoid reallocations if the total necessary coefficient count is not known at the beginning.

Summary

In summary, the steps of *HELM* are:

1. Calculate the next coefficients through solving the linear equation system

$$\sum_i Y_{ki} c_{i,n} = \begin{cases} F_{PQ}[k,n] & \text{if k is a PQ-bus} \\ F_{PV}[k,n] & \text{if k is a PV-bus} \end{cases} \tag{2.133}$$

2. Evaluate the voltage functions $U_i(s=1)$ with *Wynn's Epsilon algorithm*

3. Repeat these steps until the voltage change is small enough *or* until the power mismatch is small enough

The functions on the right hand side of Equation 2.133 are defined by

$$F_{PQ}[k,n] = \begin{cases} Y_k & n = 0 \\ I_k + S_k^\star w_{k,0} - Y_k & n = 1 \\ S_k^\star w_{k,n-1} & n \geq 2 \end{cases} \tag{2.134}$$

and

$$F_{PV}[k,n] = \begin{cases} I_k + Y_k & n = 0 \\ \frac{2P_k}{|U_k|^2} c_{k,0} - \frac{1}{|U_k|^2} x_{k,0} - c_{k,0} + \frac{I_k^\star}{|U_k|^2} v_{k,0} - Y_k & n = 1 \\ \frac{2P_k}{|U_k|^2} c_{k,n-1} - \frac{1}{|U_k|^2} x_{k,n-1} - c_{k,n-1} + \frac{I_k^\star}{|U_k|^2} v_{k,n-1} & n \geq 2 \end{cases} \tag{2.135}$$

The derived coefficients $w_{k,j}$ during the first step can be calculated with

$$w_{k,0} = \frac{1}{c_{k,0}^\star}; \tag{2.136}$$

in all further iterations

$$w_{k,j} = -\frac{\sum_{l=0}^{j-1} w_{k,l} c_{k,j-l}^\star}{c_{k,0}^\star} \tag{2.137}$$

can be used. The coefficients $v_{k,j}$ are calculated with

$$v_{k,j} = \sum_{l=0}^{j} c_{k,l} c_{k,j-l} \tag{2.138}$$

and the values for $x_{k,j}$ are determined by

$$x_{k,j} = \sum_{i \neq k} Y_{ki}^\star \sum_{l=0}^{j} v_{i,l} c_{i,j-l}. \tag{2.139}$$

As *HELM* is a fairly new algorithm and only little information is publicly available, I shall add a short numerical example in Chapter A. I will also include the coefficients and steps of *Wynn's Epsilon algorithm* to make it easier to reimplement *HELM*.

Implementation

During the research for this thesis I developed an application which is able to apply all the previously mentioned calculation methods for the load-flow calculation. Most of the application is written in *C#*, only *HELM* is implemented as a *C++*-library. This became necessary because of the superior abilities of templates over generics. I will give a more detailed explanation of the implementational aspects of the calculation methods in Section 3.2. As an input format two options are available: a custom format, stored in a *Microsoft SQL Server*, and the format used by *PSS SINCAL*. I will explain the implementation of the latter in Section 3.3. Ahead of the discussion of these implementation details, I want to give a short overview of the software architecture in the following section.

3.1 Software Architecture

The application is split into several subprojects which build upon each other. This can be seen in Figure 3.1. The most important part is the Calculation, where the calculation methods are implemented. In Figure 3.2 this subproject is shown in detail; this also consists of hierarchical blocks. This design has two big advantages: Firstly, every single subsystem can be tested seperately, without the

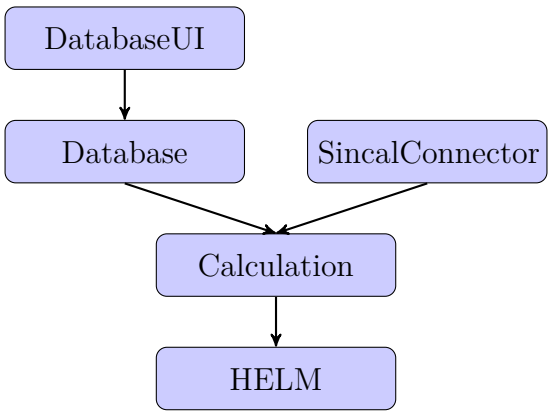

Figure 3.1: Overall software architecture

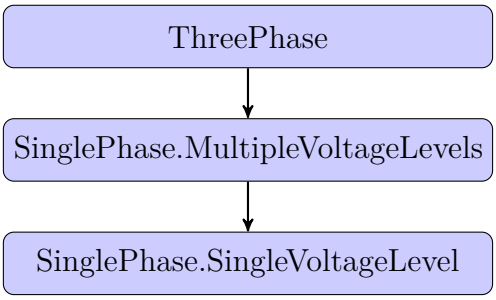

Figure 3.2: Software architecture of the subsystem Calculation

necessity to touch the other systems. Secondly, this design is more flexible. For instance, the consideration of unsymmetric situations could be achieved through three instances of a single phase net, one for each symmetric component.

Chapter C shows how the Calculation subsystem can be used. For more complex examples I would like to refer to the code of the unit tests.

3.2 Calculation Methods

3.2.1 Iterative Methods

I implemented all the iterative methods, like the *Current Iteration*, *Newton-Raphson* and *FDLF*, in *C#*. For the linear algebra I used the library *math.net numerics*[1], which has all the necessary tools implemented already. This library is obviously not optimized for this use case. Therefore, it would certainly be possible to speed up the iterative methods, although *math.net numerics*, for instance, already utilizes multiple cores. As the iterative methods are not the main focus of this work, I focused on *HELM* and just selected the best fitting methods from the library, instead of reimplementing and optimizing them.

3.2.2 Holomorphic Embedding Load Flow

First of all, I want to explain the design decision to implement *HELM* as a separate library, written in *C++*. Unfortunately, in some cases the mantissa of a 64-bit floating point is not sufficient to benefit from the theoretically perfect convergence behaviour of *HELM*. The problem here lies within the very small convergence radius of Equation 2.91. As already mentioned, the theoretical solution to this is an analytic continuation, in this case *Wynn's Epsilon Algorithm*. Approximately 50 coefficients already reach the limit of the precision of a 64-bit floating point. The calculation of more coefficients does not improve the results at all because the numerical

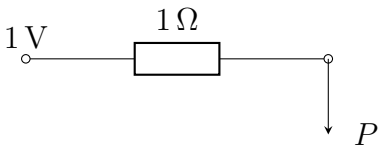

Figure 3.3: Test net for convergence border

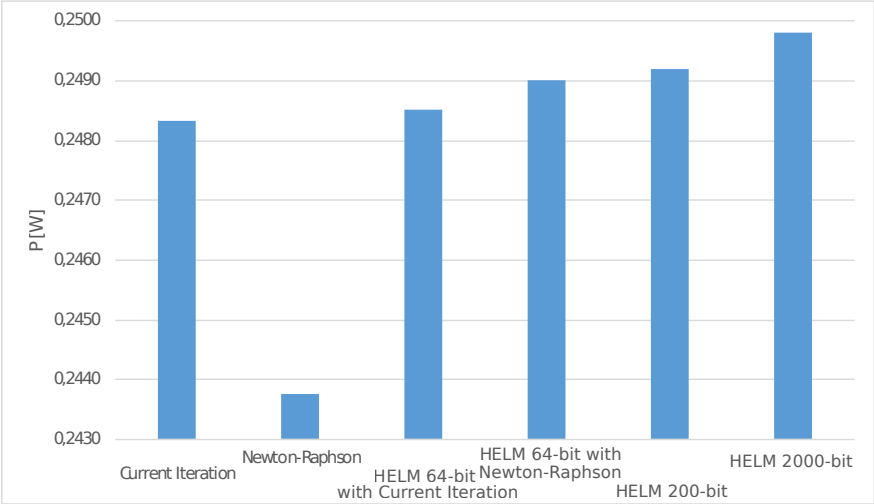

Figure 3.4: Convergence border for Figure 3.3

error, caused by the machine epsilon, is bigger than the possible gain in accuracy.

To give a more demonstrative explanation, I want to show a comparison between the convergence behaviour of the different algorithms. For this purpose I used the net in Figure 3.3, which is stable for $P \leq 0.25\,\mathrm{W}$. In this net I increased the power for as long as the algorithm converged and noted down the value closest to the border of stability. The result of this procedure is Figure 3.4, in which it can be observed that a more accurate floating point datatype enables *HELM* to get closer to the border of stability. The use of *HELM* with only 64-bit for the initial voltages already is an improvement over the direct application of the *Current Iteration* and *Newton-Raphson* in terms of convergence behaviour. But in the end, only *HELM* with an arbitrary precise datatype is able to get to the border of stability as close as desired, although this advantage is traded in for a lot worse performance.

In order to evaluate *HELM* in detail, even for nets close to their border of stability, which can not be calculated with the iterative methods, I decided to implement *HELM* with the possibility of a configurable precise datatype. As the generics in *C#* did not give me the ability to use a library of precise datatypes together with a package for linear algebra, I decided to implement this part in *C++*. In this language I had templates available, which allowed the

combination of *MPIR*[2], a library for multi precision integers and rationals, with a library for sparse linear algebra, like *Eigen*[3].

3.2.3 Linear Algebra

All the implemented methods have one thing in common: linear algebra. For all these methods it is necessary to solve linear equation systems, although the equation systems differ in their properties. In *HELM* and the *Current Iteration* method the system matrix is the admittance matrix, in *Newton-Raphson* and *FDLF* it is a Jacobian matrix. Both types of matrices are typically sparse, therefore the use of sparse linear algebra improves the performance of the algorithms significantly.

To solve these linear equation systems with a LU-factorization is quite slow for big power nets. One possible solution is the use of iterative solvers like *BiCGSTAB* [7].

Unfortunately, admittance matrices of power nets are sometimes ill-conditioned. Therefore, these iterative solvers may not converge to an accurate result. In order to be able to handle these power nets, it is necessary to reduce the bandwidth of the admittance matrix, for instance through Cuthill-McKee [9]. This method reduces the overhead caused by fill-ins during the LU-factorization. Consequently, the memory usage as well as the total runtime is reduced significantly and it is, again, possible to use the LU-factorization for nets with a six digit number of nodes.

The last important detail of the implementation of the calculation methods is the preconditioning. With this step special properties of the system matrix can be leveraged to improve the condition of the equation system and therefore the iterative solvers are accelerated. Fortunately, the admittance matrix, which is the system matrix in the *Current Iteration* and *HELM*, is approximately diagonal dominant. This makes the application of a diagonal preconditioner very handy because this type of preconditioning is very efficient to calculate. Furthermore, this preconditioning improves the condition of the equation system significantly for admittance matrices.

Optimizations

Due to the special circumstances given by *HELM* a few optimizations are possible, however can not be made in general. At the beginning I used *Eigen* for the linear algebra, but with bigger power nets I ran into performance problems with this general purpose library. Consequently, I reimplemented the necessary data structures and algorithms:

- Dense vector

[2] http://mpir.org/
[3] http://eigen.tuxfamily.org/

- Sparse matrix
- Multiplication of a sparse matrix with a dense vector
- Various operations on dense vectors
- *BiCGSTAB*
- LU-factorization
- Forward and back substitution

The specialization on dense vectors is useful in the case of *HELM* as the solutions for the equation system are typically dense. Therefore, there is no need to apply more complex operations on sparse vectors.

However, the system matrix of the linear equation system is the admittance matrix that is sparse for big problems. In fact, the amount of nonzero values in every row is nearly independent of the size of the total matrix. The reason for this lies within the physical structure of a power net, where every node only has a limited number of neighbours, no matter how big the total graph is. Therefore, the density of the admittance matrix decreases with a growing problem size. For small problems this specialization is, in fact, a performance drawback, but for these problems the performance does not matter in any case.

As there must not be a zero-row, a version of a sparse matrix with one array for each row is superior to CRS [8] or CCS [8]. Through this decision regarding the internal representation of the sparse matrix, during the calculation of the LU-decomposition it is possible to avoid a lot of memory reallocations, which become the most expensive operations if the sparse matrix is represented internally as CRS or CCS. The decision to store the matrices row- and not column-oriented is related to the matrix multiplication and the forward and backward substitution, which benefit from this format in terms of runtime.

The multiplication of a sparse matrix with a dense vector can leverage the special sparse matrix format and the dense vectors. The key point is to only iterate over the nonzero elements at each row in the matrix and select the respective elements of the vector in $\mathcal{O}(1)$. Additionally, the operation can be parallelized effectively, as all elements in the result vector are independent from each other.

Operations on the dense vectors, like the dot product or adding two vectors, can be parallelized as well. With regards to the performance of the operations, the dense storage of the values is very convenient.

The iterative solver *BiCGSTAB* benefits from the already optimized matrix and vector operations. Additionally, it is possible to avoid a few memory allocations and move operations through implementing all these operations in-place.

The calculation of a LU-factorization and the forward and backward subsitution can not benefit from multiple cores, at least for small datatypes like a double, where the floating point operations are not that expensive. For bigger datatypes with more expensive operations there is a small speedup possible.

Additionally, I had to concentrate on numerical stability. For this purpose, I sorted all values in ascending order before I summed them up. This special modification has a negative impact on the performance, but it improves the convergence behaviour of *HELM*.

3.3 Link to PSS SINCAL

The tool developed during the research for this thesis has a parser for the file format of *PSS SINCAL*. This parser allows to read a power net from the file and write back the calculated node voltages.

The basic structure of a power net in *PSS SINCAL* is as follows:

- ⟨name⟩.sin
- ⟨name⟩_files
  - database.001.dia
  - database.ini
  - database.mdb

The main information about the electrical characteristics of the power net are stored in a database, for instance a *MS Access* database. It is also possible to use *Oracle Database* or *Microsoft SQL Server*. However, for this thesis only nets with a *MS Access* database were used. Therefore, I only implemented a parser for this configuration.

Fortunately, this database is documented very well online[4] and by the documents delivered together with the application.

The most important relations in this database are:

- Terminal: contains information about the connection of the net elements with the nodes
- VoltageLevel: mostly used for the frequency of the power net
- Element: contains all net elements
- Node: contains all nodes with their ID, name, voltage level, ... etc.
- TwoWindingTransformer, ThreeWindingTransformer, Line, SynchronousMachine, Load, Infeeder: contain the corresponding net elements

[4] `http://sincal.s3.amazonaws.com/doc/Misc/SINCAL_Datenbankinterface.pdf`; accessed on 14.03.2015

- LFNodeResult: contains the node results of the load-flow calculation

PSS SINCAL obviously supports a lot more net elements and ways to describe them than the tool developed for this thesis supports. Therefore, for more sophisticated or exotic selections in the database the tool will fail to parse the power net. Especially unsymmetric power nets and short circuit calculations are not supported by this tool. Consequently, the tool neglects the values related to these calculations.

For more detailed information on the database I would like to refer to the official documentation or the implementation of the parser in the subsystem *SincalConnector*. This part of the software also has a few unit tests, which may also be used as documentation.

Results

The most interesting questions regarding *HELM* are

- How well does *HELM* perform in comparison to the iterative methods?
- Is *HELM* able to calculate large scale power nets, for which iterative methods do not converge?

To answer these questions I have run several experiments. The ones related to the first question can be found in Section 4.1. To answer the second question I will describe the results of running *HELM* on a large scale power net with a few thousand nodes in Section 4.2.

4.1 Comparison of the Load-flow Algorithms

To compare the different load-flow algorithms I used the sample nets of the Institute and compared the algorithms regarding their runtime and accuracy. I have already considered the improved convergence behaviour in Section 3.2.2, but I will show additional results regarding this aspect in this chapter.

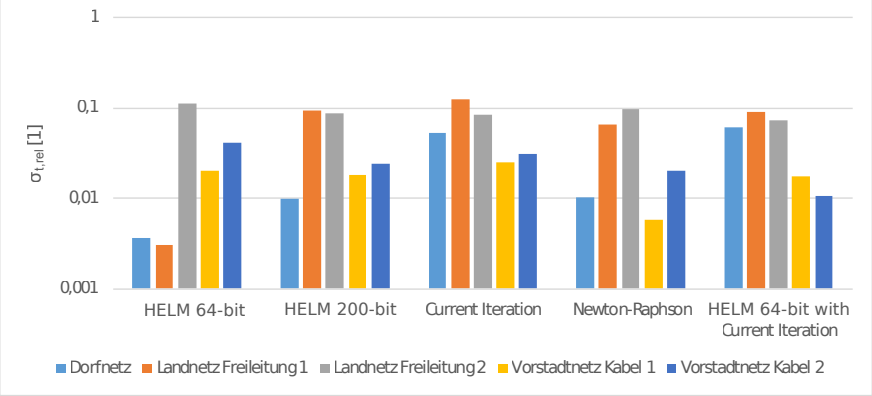

Figure 4.1: Relative standard deviation of the runtime of the algorithms

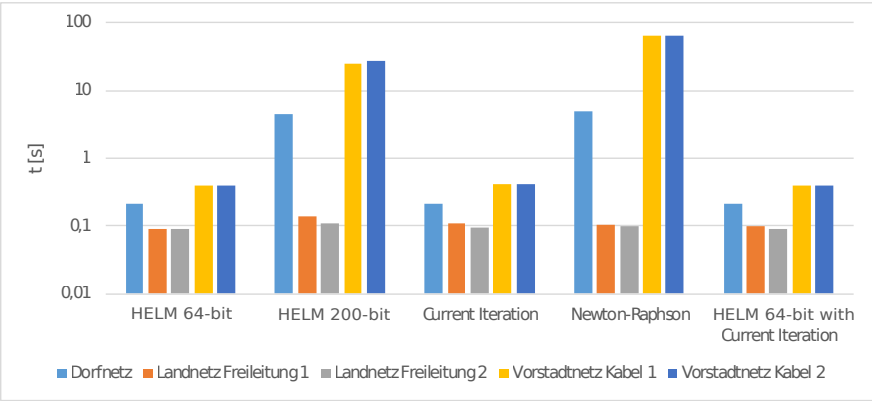

Figure 4.2: Average runtime of the algorithms for several power nets

The selected algorithms for this comparison nearly cover the whole possible spectrum that is implemented in the tool, including the usage of *HELM* to calculate seed values for an iterative method. This special application of *HELM* is represented by the algorithm *HELM* with *Current Iteration*.

The parameters for the algorithms can be found in Chapter B. The settings for the accuracy and runtime comparison are in Table B.2, the settings for the convergence border tests are in Table B.1.

To eliminate influences of other processes or the garbage collection at runtime, I ran the calculations five times. The resulting runtimes did not vary much considering the relative standard deviations (Figure 4.1).

4.1.1 Runtime

One key point is the runtime and this not only depends on the algorithms themselves but also on the used tools like the library for the sparse linear algebra. Therefore, I am not able to make absolute statements, especially because I implemented *HELM* in a different language than the other algorithms and optimized the linear algebra in *HELM*.

The first important message here is that *HELM* with 64-bit is considerably fast compared to the *Current Iteration* and *Newton-Raphson*, as seen in Figure 4.2. If *Newton-Raphson* runs into convergence problems like in the case of the *Vorstadtnetz*, *HELM* is even faster than this iterative approach.

Another conclusion from these tests is that the combination of *HELM* with an iterative method, for instance with the *Current Iteration*, is very useful. In Section 3.2.2 I have already shown that this improves the convergence behaviour, but if we take the runtime into account, the combination is not really a drawback. Considering this, I recommend to *always* use *HELM* with the *Current Iteration* instead of only using the latter.

The last important message is the insufficient performance of

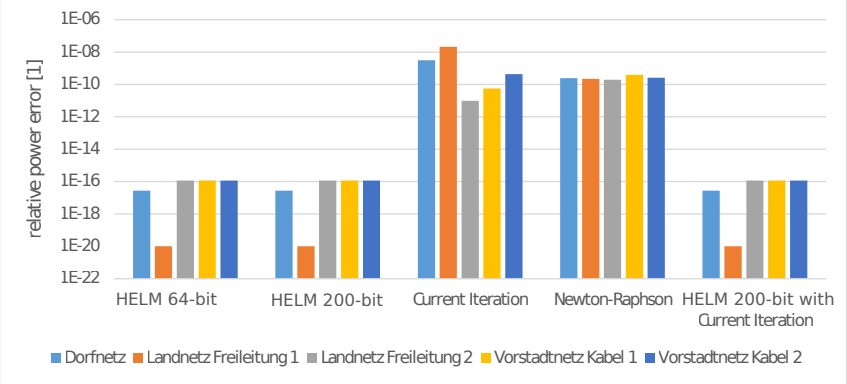

Figure 4.3: Relative power error of the algorithms

HELM with a datatype bigger than 64-bit. This setting avoids that floating point operations can be executed within a few clock cycles with the integrated assembler commands and therefore degrades the performance of the algorithm significantly. Consequently, I recommend to use *HELM* with a bigger datatype only in situations where the calculation with *HELM* with 64-bit failed.

4.1.2 Accuracy

As an accuracy metric I used the relative power error, which is the ratio between power error and total power, both summed up absolutely:

$$\epsilon_r = \frac{\sum |P_i - P_{spec,i}| + \sum |Q_i - Q_{spec,i}|}{\sum |P_{spec,i}| + \sum |Q_{spec,i}|} \tag{4.1}$$

The accuracies of the algorithms (Figure 4.3) were all sufficient for most applications, although a difference can be seen between pure iterative approaches and the ones with *HELM*. The latter ones are able to deliver more accurate results for these nets.

Keeping in mind that *HELM* is even as fast as the *Current Iteration*, there is no reason to use the *Current Iteration* instead of *HELM*. Also, *HELM* together with the *Current Iteration* produces more accurate and reliable results than the *Current Iteration* only.

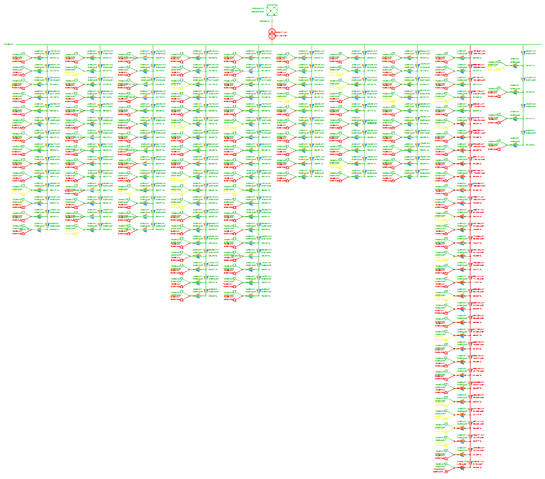

Figure 4.4: *Vorstadtnetz* used for the convergence comparison

4.1.3 Convergence

To test and compare the convergence behaviour of the different algorithms I used one of the example nets from the Institute, the so-called *Vorstadtnetz* (Figure 4.4) with nearly 300 nodes. In this net I increased the load at the most outer ends of the radial net up to a point, where the algorithm did not converge anymore. To find this convergence border more efficiently, I applied the bisection method.

The first and most obvious conclusion, which can be drawn from Figure 4.5, is that the iterative solver for the internal linear equation systems deteriorates the convergence behaviour of *HELM* significantly.

Secondly, at least for this special case, the implementation of *Newton-Raphson* in *SINCAL* has a worse convergence behaviour with these settings, compared to my implementation. But I want to make clear that this heavily depends on the settings of the algorithm, and I could only guess how certain parameters are actually implemented in *SINCAL*. Therefore, it is not really possible to draw a meaningful conclusion from this experiment regarding the implemention in *SINCAL*.

If one zooms into this chart one gets Figure 4.6, which reveals that *HELM* in its pure form outperforms the other methods, considering the convergence behaviour. After another zoom step one can see in Figure 4.7 that a more accurate datatype improves the convergence behaviour of *HELM*, although only by a few thousand watts. Additionally, such extrordinary settings affect the runtime of the algorithm significantly, as it takes a few hours to calculate this kind of net, compared to only a few seconds with 64-bit.

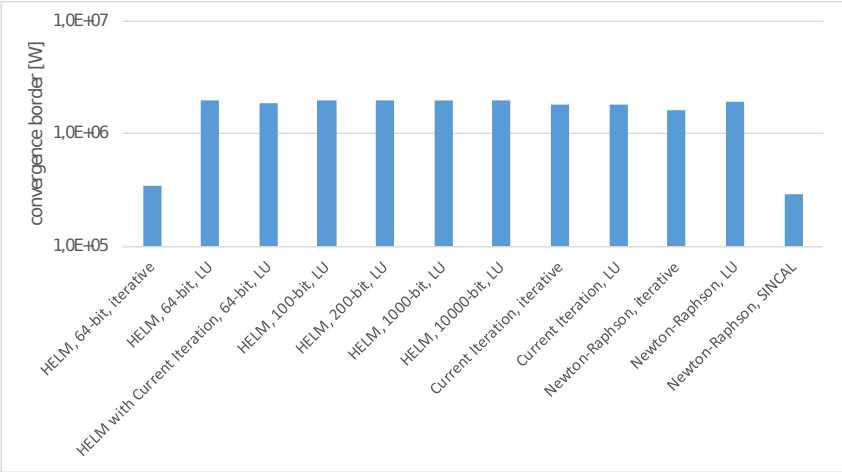

Figure 4.5: Convergence border of the algorithms

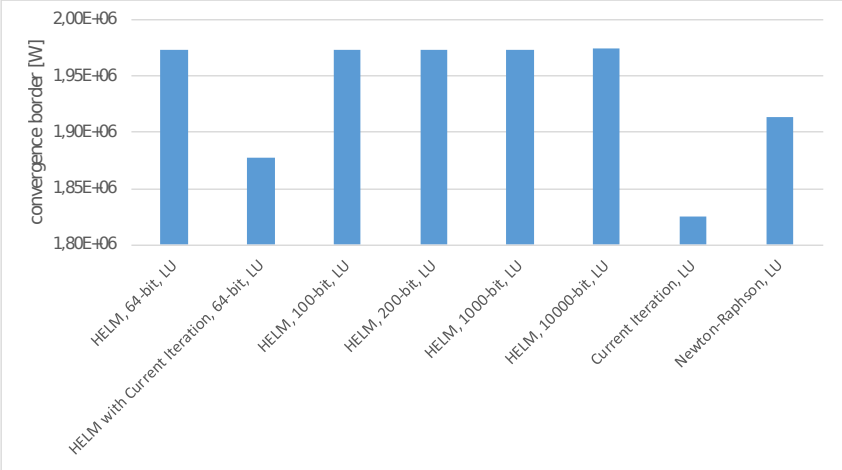

Figure 4.6: Convergence border of the algorithms

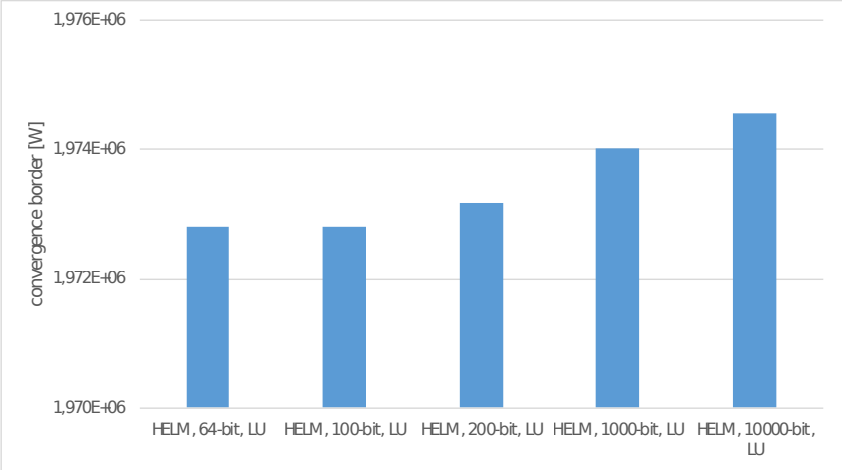

Figure 4.7: Convergence border of the algorithms

4.2 Calculation of Large-Scale Power Nets

To evaluate *HELM* in the scenario of large-scale power nets I used the power net of *infra fürth* (Figure 4.12), which was kindly provided for this purpose. Due to the current limitations of *HELM*, I had to adapt the power net. For instance, *HELM* currently does not support non-linear current controlled sources, which were used in the power net of *infra fürth* for the photovoltaic installations, as well as for the generators. To circumvent this problem I removed all the unsupported elements.

Another important aspect to point out is the switching state. The version I received was configured in a way that the total net was split up into three parts. For the comparison later on I used this initial version, as well as one where the three parts were connected together. In the connected version there was one big net with more than 50000 nodes.

As algorithms for the comparison I selected:

- *HELM* with a 64-bit datatype and LU factorization
- *Current Iteration* with an iterative solver
- *HELM* with 64-bit datatype and LU factorization and as second step *Current Iteration* with an iterative solver
- *PSS SINCAL* with the default configuration

Unfortunately, the library I used for the linear algebra in the iterative load-flow algorithms was not able to calculate the LU factorization in a reasonable amount of time. Additionally, the calculation of the Jacobian matrix was not very efficient either, due to the sparse matrix implementation. Therefore, I had to skip my implementation of *FDLF* and *Newton-Raphson* but this class of algorithms is still represented in the comparison with *PSS SINCAL*.

As I implemented *HELM* in *C++* and optimized the LU factorization for these circumstances, I was able to select this combination for the tests. This shows that the performance of the algorithms depends heavily on the implementation of the linear algebra.

First, I would like to point out the relative power errors of the algorithms for these two versions of the power net in Figure 4.8 and Figure 4.9. For most applications of a load-flow algorithm this accuracy should be sufficient.

Second, the comparison of the runtime in Figure 4.10 and Figure 4.11 shows that *HELM* is able to get close to the performance of *PSS SINCAL*. Contrary, the *Current Iteration* with the iterative solver for the linear equation systems is outperformed by *HELM* and *PSS SINCAL* by orders of magnitude. Obviously, the main difference is the implementation of the linear algebra, as the admittance matrix is ill-conditioned in these scenarios.

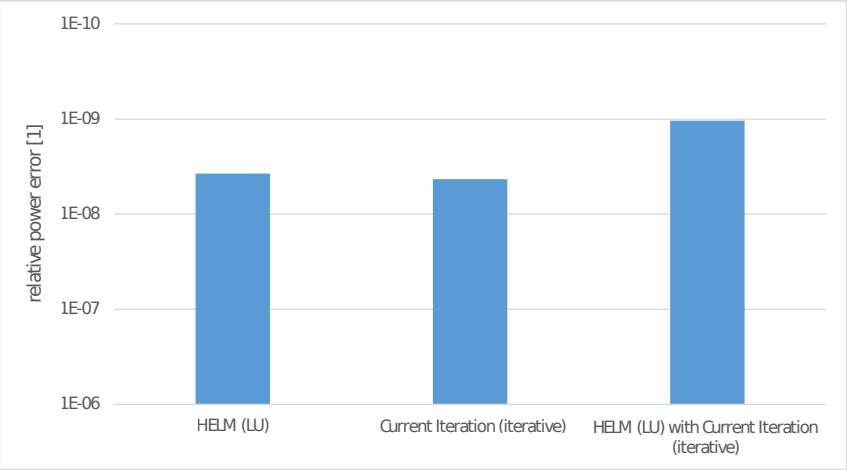

Figure 4.8: relative power error of the algorithms for the separated version of the power net of *infra fürth*

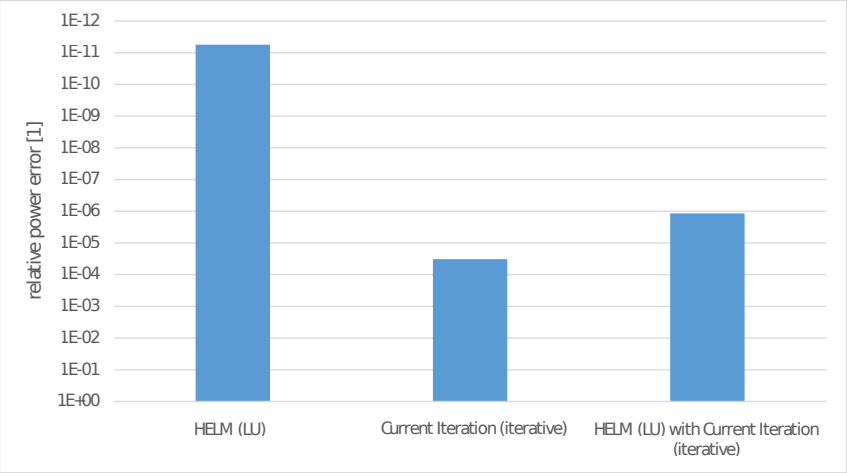

Figure 4.9: relative power error of the algorithms for the connected version of the power net of *infra fürth*

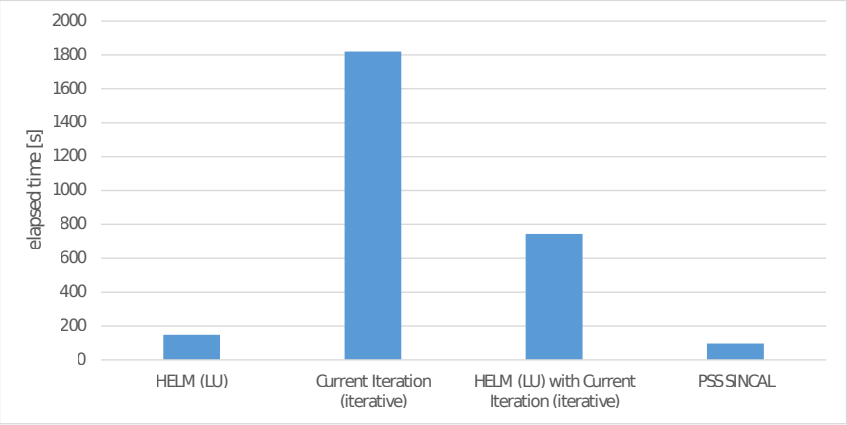

Figure 4.10: runtime of the algorithms for the separated version of the power net of *infra fürth*

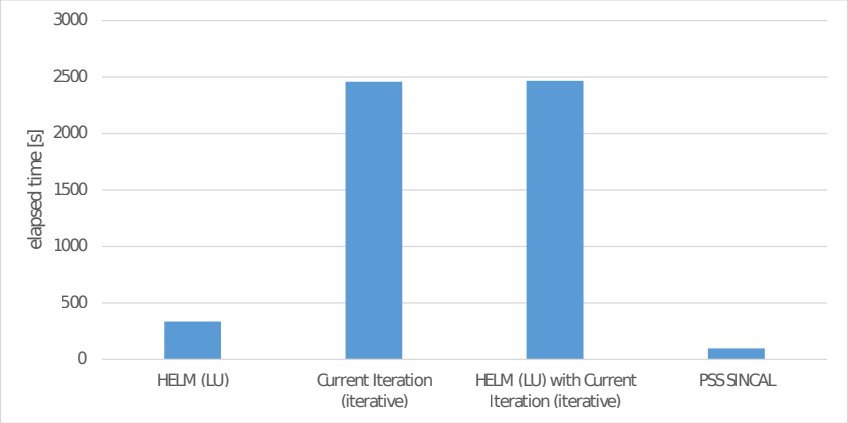

Figure 4.11: runtime of the algorithms for the connected version of the power net of *infra fürth*

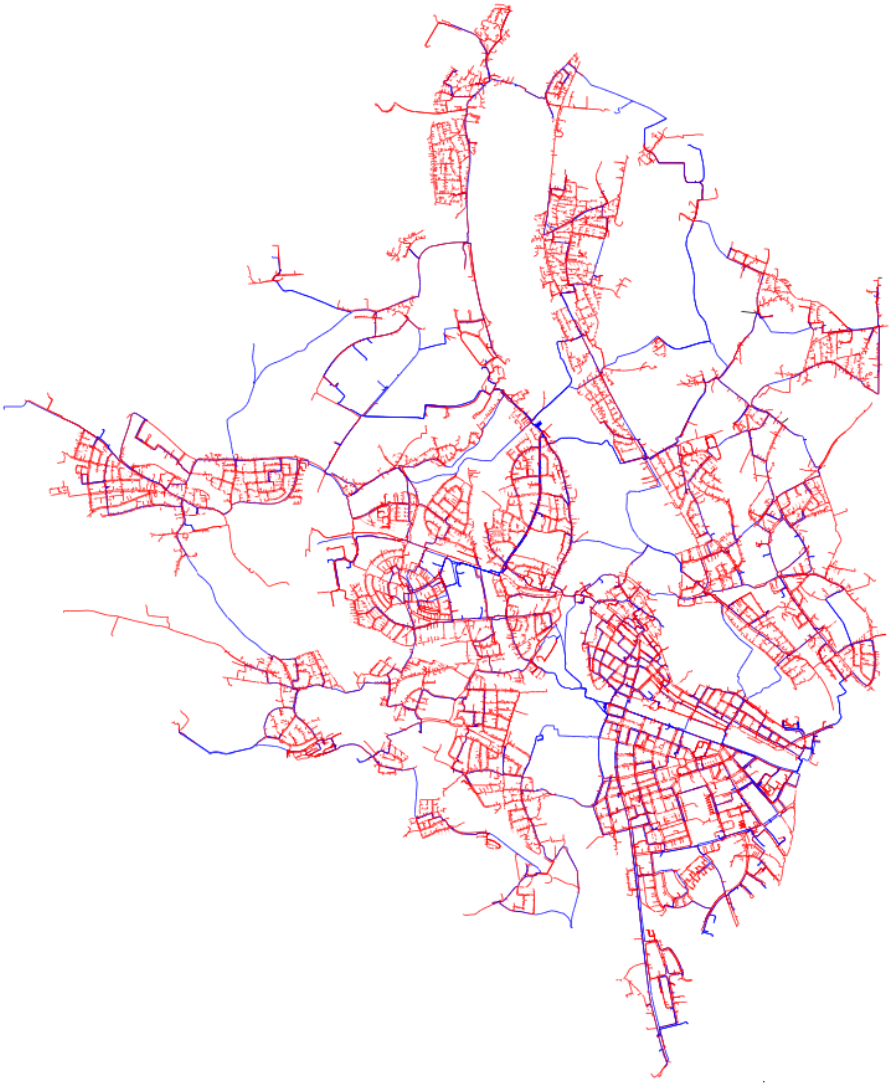

Figure 4.12: net of *infra fürth*

4.3 Conclusion

HELM is superior to the iterative methods with regards to the convergence behaviour. This advantage comes directly from the theoretical background, where so far only for *HELM* it can be proved to have a perfect convergence behaviour. The only limitation which is left here is caused by the machine epsilon of the computer. Considering the runtime, *HELM* can not reach the performance of, for instance, *FDLF* if the latter one converges within only a few iterations.

In summary, mainly two reasons not to use *HELM* exist:

1. The calculation has to be done fast
2. A certain control is used in the power net, which is not yet supported by *HELM*

The first drawback here is immanent in *HELM*, but the second one will be a topic for future research.

Finally, in practical applications it is always handy to have a fallback in case the iterative methods do not converge.

Holomorphic Embedding Load Flow Example

For the sake of simplicity, only the small net from Figure A.1 is calculated. This example has no imaginary parts in the solution and the intermediate results, because only real-valued input parameters ($U_1 = 1V$, $Z = 1\Omega$ and $P = 0{,}23W$) are used. The exact solution is determined with the current sum on the second node

$$\frac{U_1 - U_2}{Z} = \frac{P}{U_2}. \tag{A.1}$$

This is only one quadratic equation

$$U_2^2 - U_1 U_2 + PZ = 0, \tag{A.2}$$

whereas the physical solution is

$$U_2 = \frac{U_1 + \sqrt{U_1^2 - 4PZ}}{2} \tag{A.3}$$

$$= \frac{1V + \sqrt{(1V)^2 - 4 \cdot 0{,}23W \cdot 1\Omega}}{2} = 0{,}641421356V. \tag{A.4}$$

The first 15 coefficients and the tableau of the analytic continuation for this example can be found in Table A.1.

As one can see, the result of 0,6414674063V with HELM is already close to the exact solution of 0,641421356V, although only 15 coefficients were calculated. This is caused by the very special and minimalistic example. For realistic problems the analytic continuation is absolutely necessary to gain reasonable results.

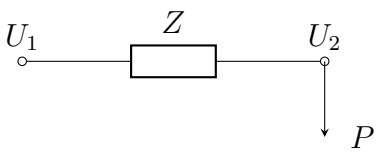

Figure A.1: Power net with two nodes

n	c_n	$\epsilon_0^{(n)}$	$\epsilon_1^{(n)}$	$\epsilon_2^{(n)}$	$\epsilon_3^{(n)}$
0	1	1	-4,347826087	0,7012987013	-53,0779978631
1	-0,23	0,77	-18,9035916824	0,672037037	-123,1056206917
2	-0,0529	0,7171	-41,094764527	0,6598435294	-228,1834058046
3	-0,024334	0,692766	-71,4691556991	0,6534624888	-377,2251277719
4	-0,01399205	0,67877395	-110,9769498434	0,6497065945	-581,3877511012
5	-0,0090108802	0,6697630698	-160,8361591933	0,647328765	-854,33787053
6	-0,0062175073	0,6635455625	-222,5006154848	0,6457460789	-1212,6396784013
7	-0,0044943696	0,6590511929	-297,6596862673	0,6446531589	-1676,2347927996
8	-0,0033595413	0,6556916516	-388,2517646964	0,6438767511	-2269,0112954398
9	-0,0025756483	0,6531160033	-496,4856326042	0,6433125844	-3019,4684373807
10	-0,002014157	0,6511018463	-624,8675009892	0,6428949783	-3961,4878350678
11	-0,0016003393	0,6495015071	-776,2329204835	0,6425810317	-5135,2249109319
12	-0,0012882731	0,6482132339	-953,7833616527	0,6423418797	
13	-0,0010484561	0,6471647778	-1161,1275707076		
14	-0,0008612318	0,646303546			

n	$\epsilon_4^{(n)}$	$\epsilon_5^{(n)}$	$\epsilon_6^{(n)}$	$\epsilon_7^{(n)}$	$\epsilon_8^{(n)}$
0	0,6577569578	-257,6917202828	0,6463316434	-965,4309176106	0,6429378349
1	0,65032677	-507,9966791024	0,6441455368	-1793,4498231255	0,6422671252
2	0,6467529581	-891,5173618297	0,6430368063	-3092,6891389693	0,641918518
3	0,6448085383	-1455,9368172673	0,6424258404	-5063,8222472924	0,6417255516
4	0,6436650919	-2262,8754903897	0,6420688182	-7977,0096092318	0,6416135118
5	0,6429551357	-3391,1393489018	0,6418507571	-12192,0568935985	0,641545953
6	0,6424961042	-4940,6930609089	0,641712852	-18183,7051794566	0,6415039389
7	0,6421897747	-7037,4691049271	0,6416231356	-26573,1993310289	
8	0,6419800633	-9839,157482384	0,6415633772		
9	0,641833429	-13542,1506659578			
10	0,6417290521				

n	$\epsilon_9^{(n)}$	$\epsilon_{10}^{(n)}$	$\epsilon_{11}^{(n)}$	$\epsilon_{12}^{(n)}$	$\epsilon_{13}^{(n)}$
0	-3284,407797319	0,6418935502	-10759,3888854953	0,6415687586	-34732,5003384756
1	-5961,2467473499	0,6416851362	-19352,1012790472	0,6415037407	-62254,6325981286
2	-10246,0728069144	0,6415753189	-33322,845655593	0,6414691767	
3	-16902,4129690484	0,6415144192	-55425,9705199859		
4	-26993,9749727727	0,6414792475			
5	-41985,2383684168				

Table A.1: HELM example coefficients and tableau

Algorithm Parameters for the Comparison

method	target precision	maximum iterations	datatype size	maximum coefficients	solver
HELM, 64-bit, iterative	1e-10		64	50	iterative
HELM, 64-bit, LU	1e-10		64	50	LU
HELM with *Current Iteration*, 64-bit, LU	1e-10	100	64	50	LU
HELM, 100-bit, LU	1e-10		100	70	LU
HELM, 200-bit, LU	1e-10		200	100	LU
HELM, 1000-bit, LU	1e-10		1000	200	LU
HELM, 10000-bit, LU	1e-10		10000	300	LU
Current Iteration, iterative	1e-10	100			iterative
Current Iteration, LU	1e-10	100			LU
Newton-Raphson, iterative	1e-10	100			iterative
Newton-Raphson, LU	1e-10	100			LU
Newton-Raphson, SINCAL	1e-10	100			

Table B.1: Algorithm parameters for the convergence comparison

method	target precision	maximum iterations	datatype size	maximum coefficients	solver
Current Iteration	1e-5	100			iterative
Newton-Raphson	1e-5	100			iterative
HELM 64-bit	1e-5		64	50	iterative
HELM 200-bit	1e-5		200	100	iterative
HELM 64-bit with *Current Iteration*	1e-5	100	64	50	iterative

Table B.2: Algorithm parameters for the runtime and accuracy comparison

Calculation API

```
var nodeVoltageCalculator = new
    HolomorphicEmbeddedLoadFlowMethod(1e-5, 50, 64);
var symmetricPowerNet =
    SymmetricPowerNet.Create(nodeVoltageCalculator, 50);

symmetricPowerNet.AddNode(1, 1000, "source");
symmetricPowerNet.AddNode(2, 1000, "load");
symmetricPowerNet.AddTransmissionLine(1, 2, 0.0002,
    0.0009, 0, 0, 2000, false);
symmetricPowerNet.AddFeedIn(1, new Complex(1050, 100),
    new Complex());
symmetricPowerNet.AddLoad(2, new Complex(-200, -100));

double relativePowerError;
var nodeResults =
    symmetricPowerNet.CalculateNodeVoltages(out
    relativePowerError);

if (nodeResults == null)
   Console.WriteLine("was not able to calcuate the power
       net");
else
   foreach (var nodeResult in nodeResults)
      Console.WriteLine("node with ID " + nodeResult.Key
          + " has the voltage " +
          nodeResult.Value.Voltage + " V");

Console.ReadKey();
```

List of Figures

List of Tables

Bibliography

[1] A. Trias, *The Holomorphic Embedding Load Flow Method*, IEEE PES General Meeting, July 2012

[2] M. K. Subramanian, Y. Feng, D. Tylavsky, *PV Bus Modeling in a Holomorphically Embedded Power-Flow Formulation*, 978-1-4799-1255-1/13, IEEE, 2013

[3] A. Trias, *System and Method for Monitoring and Managing Electrical Power Transmission and Distribution Networks*, US Patent 7,519,506 B2, April 2009

[4] A. Trias, *System and Method for Monitoring and Managing Electrical Power Transmission and Distribution Networks*, US Patent US 2009/0228154 A1, September 2009

[5] P. Wynn, *The Epsilon Algorithm and Operational Formulas of Numerical Analysis*, June 1960

[6] H. Saadat, *Power System Analysis*, ISBN 0-07-012235-0, McGraw-Hill, 1999

[7] H. A. van der Vorst, *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*, ISSN 2168-3417, SIAM Journal on Scientific and Statistical Computing, Volume 13, 1992

[8] M. Luján, A. Usman, P. Hardie, T. L. Freeman, J. R. Gurd, *Storage Formats for Sparse Matrices in Java*, Computational Science – ICCS 2005: 5th International Conference, Proceedings, Springer Science & Business Media, 2005

[9] E. Cuthill, J. McKee, *Reducing the Bandwidth of Sparse Symmetric Matrices*, Naval Ship Research and Development Center, Washington DC

[10] B. Stott, O. Alsac, *Fast Decoupled Load Flow*, ISSN 0018-9510, Power Apparatus and Systems, IEEE Transactions on (Volume: PAS-93, Issue: 3), May 1974