Optimization of Power Flow Computation Methods

Christoph Kattmann

Schriftenreihe des Instituts für
Energieübertragung und Hochspannungstechnik

Band 39 - 2022

Universität Stuttgart
Institut für Energieübertragung und Hochspannungstechnik, Band 39

Optimization of Power Flow Computation Methods

Optimization of Power Flow Computation Methods

Von der Fakultät
Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von
Christoph Kattmann
aus Hannover

Hauptberichter:	Prof. Dr.-Ing. Stefan Tenbohlen
Mitberichter:	Univ.-Prof. DDipl.-Ing. Dr. Robert Schürhuber
Tag der mündlichen Prüfung:	19.07.2022

Institut für Energieübertragung und Hochspannungstechnik
der Universität Stuttgart

2022

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie, detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Universität Stuttgart
Institut für Energieübertragung und Hochspannungstechnik, Band 39
D 93 (Dissertation der Universität Stuttgart)

Optimization of Power Flow Computation Methods

© 2022 Kattmann, Christoph

Herstellung und Verlag: BoD – Books on Demand, Norderstedt
ISBN: 978-3-75-629545-6

Danksagung

Diese Arbeit entstand während meiner Tätigkeit am Institut für Energieübertragung und Hochspannungstechnik an der Universität Stuttgart. Ich bin allen IEHlern, die diese Zeit mit mir geteilt haben, dankbar für die Freundschaft, die Toleranz, die Geduld, den Humor, die Unterstützung und die Kompetenz.
Bei Prof. Tenbohlen muss ich mich vor allem für das Vertrauen, das er mir all die Jahre entgegengebracht hat, den Antrieb und das Engagement bedanken. Bei Prof. Braun, der mich am IEH eingestellt hat, und Prof. Rudion, der meine Arbeit immer unterstützt hat, möchte ich mich ebenfalls bedanken. Vielen Dank auch an Prof. Schürhuber für die Übernahme des Berichts und die wertvollen Anregungen.
Ebenfalls danken möchte ich allen Studenten, die bei mir ihre Abschlussarbeiten geschrieben haben und einen wertvollen Beitrag geleistet haben, insbesondere Arne Ellerbrock, Mathias Jaschke, und Julian Leppert.
Daniel Contreras und mein Bruder Tobias Kattmann haben viele kleine und große Verbesserungen in die Arbeit eingebracht, vielen Dank für die sicher langwierige Korrekturarbeit.
Meinen Kollegen Martin Siegel, Michael Beltle, Malte Gerber, und Michael Schühle danke ich für die freundschaftliche Rückendeckung insbesondere während der Endphase der Arbeit.
Meinen Eltern und meiner gesamten Familie danke ich für die Unterstützung, den Optimismus und die Zuversicht, die sie mir schon mein ganzes Leben lang mitgegeben haben und die bei dieser Arbeit manchmal besonders wertvoll war.
Meiner Freundin Melly danke ich aus tiefstem Herzen für die Geduld und den Beistand an vielen langen Tagen. Unsere Tochter Olivia hat von all dem wahrscheinlich noch nichts mitbekommen. Ich hoffe, die Arbeit besitzt noch Relevanz, sollte sie sie eines Tages lesen.

Abstract

Power flow computations are a cornerstone of many simulations regarding the electric grid. With the recent developments in distributed generation and the continuing electrification of mobility and residential heating, a targeted grid operation and expansion is more important than ever to ensure reliability and sustainability. At the same time, these developments change the preconditions of electric grid simulations and often necessitate large-scale simulations with millions of individual scenarios.
This thesis evaluates the landscape of power flow computation methods with a focus on practical computational performance in large-scale simulations, as they occur in modern distribution grid planning. For this purpose, a number of power flow methods are investigated. In order to enable a wider range of applications, a selection of complicating grid elements and factors like multiple slack nodes, transformers, and asymmetric conditions are investigated for their effect on complexity and performance.
The main finding of this thesis is that the Z_{BUS} Jacobi method, an often-neglected power flow algorithm, can be magnitudes faster than the other methods, especially the established Y_{BUS} Newton-Raphson method. An important factor in the computational performance of all power flow methods is an implementation guided by the requirements of modern CPUs, which requires a mathematical formulation that leverages CPU features like caching, instruction pipelining and branch prediction.
In addition to the solution methods, the thesis presents a series of acceleration methods like static acceleration factors, grid reduction and parallelization, which can be applied to power flow problems for a further performance boost.
Finally, the thesis presents three real-world example problems and investigates the most performant solution method as well as the effect of the various acceleration strategies.

Kurzfassung

Lastflussberechnungen sind der zentrale Baustein vieler Simulationen, die das elektrische Netz betreffen. Entwicklungen im Rahmen der Energiewende wie dezentrale Erzeugungsanlagen, elektrische Mobilität und elektrische Wärmepumpen erfordern einen gezielten Netzbetrieb und -ausbau, um die Zuverlässigkeit und Nachhaltigkeit der Energieversorgung sicherzustellen. Gleichzeitig ändern diese Entwicklungen die Anforderungen an Netzsimulationen und machen teilweise umfangreiche Simulationen mit Millionen von einzelnen Berechnungsfällen notwendig.
Die vorliegende Arbeit untersucht gängige Lastflussmethoden mit einem besonderen Fokus auf die Berechnungsgeschwindigkeit bei umfangreichen Simulationen, wie sie in der modernen Verteilnetzplanung vorkommen. Dazu werden eine Anzahl von Lastflussmethoden vorgestellt. Um eine breite Anwendbarkeit sicherzustellen, werden weiterhin eine Reihe von komplexeren Netzelementen und -umständen wie mehrere Slack-Knoten, Transformatoren und asymmetrische Bedingungen untersucht und ihr Einfluss auf die Komplexität der Modelle und die Berechnungsgeschwindigkeit analysiert.
Die Schlüsselerkenntnis der Arbeit ist die Effektivität der Z_{BUS}-Jacobi-Methode, einer oft vernachlässigten Lastflussmethode, die aber um Größenordnungen schneller als andere Methoden sein kann, insbesondere der Y_{BUS}-Newton-Raphson-Methode. Ein wichtiger Einfluss auf die praktische Performance ist die Implementierung gemäß den Anforderungen moderner Prozessoren unter Berücksichtigung von Caches, Instruction Pipelining und Branch Prediction.
Zusätzlich zu den Lastflussmethoden präsentiert diese Arbeit mehrere Beschleunigungsmethoden wie statische Beschleunigungsfaktoren, Netzreduktion und Parallelisierung, die die Lösung praktischer Probleme weiter beschleuningen können.
Abschließend untersucht die Arbeit drei reale Netzsimulations-Probleme im Hinblick auf die performanteste Lösungsmethoden und den Effekt der vorgestellten Beschleunigungsstrategien.

Contents

Table of Abbreviations

AC	Alternating Current
ALU	Arithmetic Logic Unit
BFS	Backwards-Forwards Sweep
BLAS	Basic Linear Algebra Subsystem
CPU	Central Processing Unit
CSC	Compressed Sparse Column
CSR	Compressed Sparse Row
DTLB	Data Translation Lookaside Buffer
EV	Electric Vehicle
FLOP	Floating Point Operations per Second
FPU	Floating Point Unit
GPGPU	General Purpose Graphics Processing Unit
HELM	Holomorphic Embedded Load Flow
HV	High Voltage
ITLB	Instruction Translation Lookaside Buffer
KCL	Kirchhoff Current Law
KVL	Kirchhoff Voltage Law
LAPACK	Linear Algebra Package
LV	Low Voltage
MV	Medium Voltage
PES	Power and Energy Society of the IEEE
PV	Photovoltaics or voltage-controlled node
RAM	Random Access Memory
RMS	Root Mean Square
TLB	Translation Lookaside Buffer

Nomenclature

As this thesis touches on topics from electrical engineering, mathematics and computer science, the notation cannot be adopted from one field alone. This nomenclature introduces all notations necessary to be unambiguous for researchers from all three fields.

Symbol Conventions

In all equations and algorithms, the mathematical expressions use

- italic letters like i for scalar values,
- underlined italic letters like $\underline{U}$ for complex values,
- bold letters like $\mathbf{P}$ for vectors and matrices and
- bold, underlined letters like $\underline{\mathbf{Y}}$ for complex-valued vectors and matrices.

A lower index signifies the corresponding entry in the vector or matrix, so $\underline{U}_i$ is the ith entry of the vector $\underline{\mathbf{U}}$.
The iteration count is put in parentheses in the exponent like $x^{(m)}$.

Vector & Matrix Operations

The *Hadamard operators* are frequently used to denote the element-wise multiplication and division of two vectors of the same size. The Hadamard Product ($\odot$) denotes the element-wise multiplication as in

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \odot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1\, b_1 \\ a_2\, b_2 \end{bmatrix}, \tag{1}$$

and the Hadamard division ($\oslash$) denotes the element-wise division as in

$$\mathbf{A} \oslash \mathbf{B} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \oslash \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1/b_1 \\ a_2/b_2 \end{bmatrix}. \tag{2}$$

Multiplications of scalars and matrix multiplications do not use a symbol, the dot $\cdot$ is reserved for the dot product between two vectors.
The unity matrix $\mathbf{E}$ (0 everywhere except 1 on the entire diagonal) is used without explicit definition of the dimensions. It is assumed that $\mathbf{E}$ is always chosen to fit the surrounding vectors and matrices.

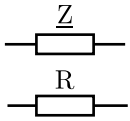
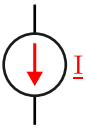
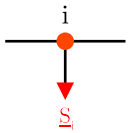
Electrical Conventions

The direction of an electrical current in an AC grid is of course ever-changing with the grid frequency. Nevertheless, the current is sometimes represented with an arrow in equivalent circuits. It is then to be understood as relative to the direction of the voltage, which changes in the same way.
In this thesis, the load convention is such that a positive real power denotes consumption or load, a negative real power denotes generation or injection. A positive reactive power is assigned to a load with inductive behavior, and a negative reactive power to capacitive loads.
Electrical quantities denoted in capital letters, like $\underline{U}$ and $\underline{I}$, refer to the RMS values.

Equivalent Circuits

The symbols in the equivalent circuits are chosen according to IEC 60617 [49].

Table 1: Symbols used in equivalent circuits

Symbol	Explanation
	Resistance with value R *or* complex impedance with value $\underline{Z}$
	Inductance with value L
	AC voltage source with voltage $\underline{U}$
	AC current source with current $\underline{I}$
	Load with constant power $\underline{S}$
	Node i, to which a load with constant power $\underline{S}_i$ is attached

Algorithm Notation

As a main subject of the thesis is the formulation of algorithms which are meant to be implemented using modern, high-level programming languages and numerical libraries, the capabilities of those environments are used in the algorithms even if they have no simple mathematical equivalent. In the algorithms shown in this thesis, the operators and functions are used analogous to the capabilities of the Numpy library for

Python. Indices of vectors and matrices therefore start at 0. Accessing parts of vectors or matrices is achieved using bracket notation, which is outlined in table 2.

Table 2: Bracket notation for partial access to vectors and matrices

Notation	**Explanation**
$\mathbf{P}[i]$	Value of $\mathbf{P}$ at index i
$\mathbf{P}[:i]$	Vector of values of $\mathbf{P}$ up to, but not including index i
$\mathbf{P}[i:]$	Vector of values of $\mathbf{P}$ starting at index i up to the end
$\underline{\mathbf{Y}}[i,:]$	Vector containing the entire ith line of the matrix $\underline{\mathbf{Y}}$
$\underline{\mathbf{Y}}[1:,1:]$	Matrix containing everything from $\underline{\mathbf{Y}}$ except the first line and column

The special functions outlined in table 3 are used throughout the thesis, all of which have direct analogs in Pythons Numpy and other numerical libraries.

Table 3: Special functions used in algorithm listings

Function	**Description**	**Example**
$\lvert()\rvert$	Absolute value	$\lvert[1,-1,3+4i]\rvert = [1,1,5]$
`arg()`	Argument	`arg`$([1,-1,3+3i]) = [0,\pi,\pi/4]$
`exp()`	Exponent	`exp`$(j\pi) = -1$
`cumsum()`	Cumulative sum	`cumsum`$([1,2,3]) = [1,3,6]$
`diag()`	Diagonal of matrix	`diag`$(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}) = [1,4]$
`reversed()`	Reverse vector	`reversed`$([1,2,3]) = [3,2,1]$
`argmax()`	Index of max. element	`argmax`$([1,6,3]) = 1$

In some places, the breakdown of operations uses the star symbol $\star$ for a previously computed value with no explicit name. Its meaning should always be clear from the context.

Variables and Symbols

Table 4 lists the used variable and symbol names.

Table 4: Variable and symbol names used throughout the thesis

Symbol	Unit	Explanation
n	$-$	Number of nodes in the grid
i, j, k	$-$	Node indices
m	$-$	Iteration count
$\underline{U}$	V	Voltage
$\underline{U}_i$	V	Voltage from node i to ground
$\underline{U}_{ij}$	V	Voltage difference from node i to j
$\underline{\mathbf{U}}$	A	Vector of the voltages of all nodes
$\underline{I}$	A	Current
$\underline{I}_i$	A	Load Current at node i
$\underline{I}_{ij}$	A	Current through the line connecting nodes i and j
$\underline{\mathbf{I}}$	A	Vector of the load currents at all nodes
$\underline{\mathbf{I}}_R$	A	Vector of residual currents
$\underline{S}$	VA	Complex Power
$\underline{S}_i$	VA	Complex Power at node i
$\underline{\mathbf{S}}$	VA	Vector of the complex powers of all nodes
$\underline{\mathbf{S}}_R$	VA	Vector of residual complex powers
$\underline{Z}$	Ω	Impedance
$\underline{Z}_{ij}$	Ω	Impedance of the line connecting nodes i and j
$\underline{Z}_{ij,p}$	Ω	Impedance of the line connecting nodes i and j in phase p
$\underline{\mathbf{Z}}$	Ω	Inverted admittance matrix
$\underline{Y}$	$1/\Omega$	Admittance
$\underline{Y}_{ij}$	$1/\Omega$	Admittance of the line connecting nodes i and j
$\underline{\mathbf{Y}}$	$1/\Omega$	Admittance matrix
$\underline{\mathbf{Y}}_{diag}$	$1/\Omega$	Vector containing the main diagonal of the admittance matrix
ϵ	$-$	Convergence Threshold
ϵ_S	VA	Convergence Threshold for Power Residual
ϵ_I	A	Convergence Threshold for Current Residual
ϵ_U	V	Convergence Threshold for Voltage Step

Chapter 1

Introduction

1.1 Motivation

The electric power supply system is one of the most large-scale and complex man-made systems in the world. Its continuous operation is essential for many parts of modern civilization, and ensuring its reliable service is the primary task of all power plant and grid operators as well as power system regulators.
The reliability of the electric power system is a constant challenge because the operating state of the electric grid is inherently unstable on several levels. Without constant intervention by several, deeply interwoven control loops, electric power generation would cease to function within seconds. Without finely tuned market systems, the power demand might not be met by an equal generation at all times, and system-critical institutions and companies might be at risk. Without sensible regulations and focused technical progress, the long-term growth and change of power generation and consumption would slowly lead to an unsuitable electric grid. The fact that this techno-economic system has largely worked without fault for decades is a testament to the work of thousands of engineers and scientists in many fields.
Their work is complicated by the fact that, as opposed to many other engineering and scientific problems, it is very hard to develop any aspect of the electric grid *in situ*, i.e. using the real grid for experiments. This has led to the rise of simulations as a major component of all research and development concerning the electric grid. Today, simulations are a part of almost all short- and long-term decisions about the electric sys-

tem, like the dispatch of power plants, grid expansion planning, approval of local photovoltaic (PV) plants, the scheduling of maintenance work or even strategic directives that affect the economies of entire nations for decades to come.

The trend towards a more sustainable energy generation has made the conditions for those simulations more complex. Historically, electric power generation was centralized in big power plants, which were connected to the highest-level transmission grid. From there, the power flowed through the distribution grid levels (high, medium, and low voltage) to the consumers, which largely consisted of predictable household, commercial, and industrial loads. Under these conditions, the design and dimensioning of the distribution grids was simple: The grid had to withstand the highest consumer load. If the grid was suitable for this single condition, it was suitable for all other operating conditions as well. The prevalence of distributed generation like PV plants and wind power generators, which are predominantly connected to distribution grids, has invalidated this simple approach. The state of highest load might now also occur when consumption is low and generation is high, turning distribution grids into net generators of power. Overloads of individual lines or voltage band violations can also be caused by power flowing from generators to consumers inside of one grid, even if the overall loading condition seems uncritical.

At the same time, the trend towards electric vehicles and residential heat-pumps changes the nature of the consumer side. Whereas before, the largest household loads like electric stoves were active only during short and well-distributed times during the day, electric vehicles can draw a multiple of that power and also show a considerable correlation among households.

As a consequence of these developments, grid planning today involves a lot more individual simulations than before, usually as time-series simulations, e.g. the individual simulation of every hour or even minute in a year. A comprehensive grid expansion study including multiple expansions variants and contingencies can quickly comprise millions of individual simulations [85].

At the core of these simulations is usually some variant of a *power flow computation* - a function that takes a set of powers and a grid model expressed through its topology and individual line impedances as input, and computes the voltage at each individual node, from which all other electrical parameters concerning the flow of power in the grid can be derived.

The computational runtime of these power flow computations, which was not an issue in the days of single-case grid expansion planning, is becoming a serious obstacle for large-scale studies. Alternative approaches like *probabilistic power flow* [121] can approximate the results of these studies, but for more complex scenarios e.g. involving intermittent producers and consumers, conventional power flow computations are still indispensable and increasingly represent a serious bottleneck for data-driven distribution grid planning. This emerging problem motivated the research that led to this thesis.

1.2 History and State of the Art

The need for studies on the state and reliability of the electric grid has driven the development of power flow computation methods since the beginning of the 20th century. The first computational aids for the simulation of electric grids and its connected elements were AC network analyzers, scaled-down grids in a lab environment which contained physical models of transmission lines, generators, loads, transformers, and shunt elements with matching values (not to be confused with EMC network analyzers used to evaluate high-frequency circuits). Often, the operating frequency was elevated to scale down the model capacitances and transformers. The first general-purpose AC network analyzer was built at MIT in 1929 [44] , and the design was improved at least until 1952 [60].
The advent of digital computers in the 1940s was closely followed by the engineers of grid operators who were looking for cheaper and more flexible ways to conduct power flow studies. The first paper in the IEEE database that mentions this possibility is from 1946 [56]. Up to around 1955, research in this area mainly focused on establishing the possibility of power flow computations with all their complications and the optimal usage of the punch-card computers of the day. The American grid operators and universities were the most active in this domain, probably because they had the best access to the first computers. Lyle Dunstan from the Federal Power Commission was one of the first to publish results in 1947 [28], needing "only ten hours" to compute the power flow for a network with 24 nodes. (Today, that task would take only milliseconds on even the lowest-end hardware.) He then used the method to calculate coefficients for the influence of load changes on currents and voltages [29], the first automatic linearization of power

flow equations.
In the late 1950s, research around digital power flow solutions heated up with the increasing capabilities and availability of computers. Dunstan [27] and J. Henderson [45] presented their workflows on punch-card machines in detail.
Up to this point, one severe obstacle for the proliferation of power flow computations was the arduous and error-prone manual data parsing from grid plans and parameter tables to the matrices and vectors required for the computation. In 1956, Ward and Hale [120] presented an approach to use a *nodal formulation* of networks to make the automatic formulation of the required data structures easier. Their equations evolved into the Y_{BUS} *formulation* of the power flow problem (introduced in section 3.2) and enabled the application of standard mathematical methods with all their optimization approaches.
Glimn and Stagg [39] [57] [14] for instance built on the nodal formulation to investigate various iterative solution methods, like the Gauss-Seidel method with acceleration factors, pushing the time required for one power flow computation down to minutes.
Almost every researcher from that time still spent considerable effort comparing digital computers to analog AC network analyzers, usually emphasizing the need for both. The rapid development of computers however rendered network analyzers obsolete by the 1960s, relegating them to a curiosity of technology history.
In 1959, J. van Ness applied the Newton-Raphson root-finding method to the Y_{BUS} formulation [115]. His outstanding contribution is the formulation of the required Jacobi matrix in terms of known values, eliminating the need for a costly manual or automatic derivation of the power flow equations. Due to the, at the time, "excessive" storage demands, the method found little adoption until significant progress was made by W. Tinney in 1967, who devised a sparse storage scheme for the Jacobi matrix [109]. The robustness of the Newton-Raphson solution method against complicated network topologies contributed to its general applicability and established it as the most-used solution method until today.
In 1962, Brameller and Denmead introduced the Z_{BUS} method [10]. Brown et al. later explored further solution methods based on the method and added many relevant component models [12][13]. The method was however never widely adapted, again due to the storage demands, which could not be overcome with a sparse formulation this time.

With that, a wide selection of solution methods was known and the foundations for decades of optimization, extension and adaption were set. In fact, in 1964, M. Laughton and M. Humphrey Davies published the first overview paper [63] and attempted a classification of many different power flow solution algorithms.
In the following years, most research focused on the application of the newfound capabilities to more advanced concepts like state estimation [62], probabilistic calculations [3] [4] [121], and technical and economic optimizations [22] [83]. In fact, by the 1970s, the progress of computing performance had satisfied the technical needs, and the speed of power flow computations was no longer a limiting factor for many applications. Today, power flow computation has become a feature of programs or libraries that often have a greater scope like dispatch and flow optimizers or asset management. For those programs, the speed of the power flow computations is not the main focus.
In the scientific community, after the heydays of power flow developments in the 1960s and 70s, the number and focus of publications on the performance of power flow methods has declined. A frequent problematic issue of newer publications is the reliance on findings of older papers regarding computational properties, which were observed with the hardware available at the time and whose performance - quantitatively as well as qualitatively - bares little resemblance to modern computers. Also, over the years, some precedence for imprecise usage of formal numerical terms was set, and so some papers today continue to cite these inaccurate notations, most egregiously *ill-conditioned* (see section 4.7), which is frequently broadly attributed to matrices, grids, networks, or 'systems' [55] [113] [98] [66] [11] [73] [38] [31] [71] [7] [20] [78] [80] as a justification for new algorithm developments, evidently without any investigation of the numerical phenomenon, its effect or the readily available remedies.
A list of selected publications from circa 1990 to 2020 is listed below.

Reviews and Comparisons

There is no shortage of review and comparison papers about power flow solution methods. However, there is very little value these papers could contribute since the 1970s. Regrettably, a common understanding of the existing landscape of power flow methods is missing, and long-known methods are sometimes presented as 'novel'.

The challenging task of properly conducting benchmarks on modern CPUs and explaining the results is mostly avoided. The author has no knowledge of an overview paper with comprehensive benchmarks that were conducted with modern, cache-based CPUs, which were introduced in the 1990s.
In [126] from 1995, the author lists several solution methods specifically for radial distribution grids, including a network reduction method and three topological methods and compares them to the Y_{BUS} Newton-Raphson method and a Z_{BUS} Jacobi method. The computational performance is measured in FLOPs, which is a flawed metric. (see section 2.4)
In [73] from 2008, the authors list works on power flow methods with a focus on distribution grids and sort them into three categories: Backward-Forward Sweep, "Implicit Z_{BUS} Gauss Methods", and Newton-like methods. They state that "classical techniques [...] may become inefficient as in the case of ill-conditioned [...] networks". Several of the methods listed under Backward-Forward Sweep ([37] [69] [90] [98]) are actually Z_{BUS} Jacobi methods which use a topological approach to construct the matrix $\underline{\mathbf{Z}}$.
In [7] from 2011, the authors also state that 'distribution networks [...] fall in the category of ill-conditioned power systems'. They use roughly the same categories as [73] and additionally list "Compensation methods", an "Implicit Z_{BUS} Gauss Method", and a "Direct method", which are all just various formulations of the Z_{BUS} method.
In [74] from 2018, the authors claim that "the Gauss-Seidel and Newton-Raphson techniques [...] fail to the meet the requirements in both performance and robustness aspects" without any further explanation. It goes on to list a "Network Topology method" [107], a Backward-Forward Sweep method [78], and a "Novel Based Method" [40] which all end up being mathematically identical to the Z_{BUS} Jacobi method.
In [97] from 2018, a paper on design considerations for distribution test feeders, the authors list the Backward-Forward Sweep method, two 'Impedance Methods', which are both identical to the Z_{BUS} Jacobi method, and the Y_{BUS} Newton-Raphson method. The paper also lists a "fixed-point iteration method" under "Other Methods", although the Backward-Forward Sweep method and the Z_{BUS} Jacobi methods are also fixed-point methods. The cited fixed-point method is again mathematically identical to the Z_{BUS} Jacobi method.
The book 'Elektrische Kraftwerke und Netze' by Oeding and Oswald [79] (7th edition from 2011) is a German standard textbook on power

grids and contains a classification of power flow methods. They separate the methods based on whether they are derived from a current or a power equilibrium, which leads to a completely different classification. The authors place a great emphasis on the better algorithmic efficiency of Gauss-Seidel methods compared to Jacobi methods, but the computational performance is not evaluated at all.
In the book 'Power System Modelling and Scripting' [71] from 2010, the author Frederico Milano presents practical, albeit computationally inefficient implementations of the Y_{BUS} Jacobi, Y_{BUS} Gauss-Seidel and Y_{BUS} Newton-Raphson methods as well as a comprehensive set of device models in Python. He correctly recognizes that the Y_{BUS} Jacobi method has a computational advantage over the Y_{BUS} Gauss-Seidel method because it can be implemented in a vectorized fashion, he however incorrectly assumes that that advantage would not occur in a "system language" like C or Fortran.

Y_{BUS} Newton-Raphson Methods

The Y_{BUS} Newton-Raphson method quickly came to dominate the scientific and commercial landscape of power flow algorithms. However, just like in other domains, the Newton-Raphson method is very hard to improve on algorithmically. One promising approach is to approximate the Jacobi matrix, which creates an *inexact Newton method*. However, no specific acceleration approach has found widespread usage beyond the decoupling approach only usable in wide-area transmission grids [103].
In [88] from 1982, the authors present an implementation of the Fast-Decoupled Load Flow method on a specialized array coprocessor which can execute vectorized operations. The paper is fully dedicated to the implementation and contains many statements that are true with today's CPUs, which integrated these vector instructions shortly after their usefulness was proven by these dedicated processors.
In [66] from 1997, the author claims to develop a formulation of the Jacobi matrix in complex terms only, but actually attempts to make the Newton-Raphson method more efficient by using partial difference equations, not derivations, and then neglect one of the partials, which leads to a formulation of the simplified Jacobi matrix in complex terms. This is in fact just an inexact Newton-Raphson method.
In [124], also from 1997, the author presents a method derived from the

Newton-Raphson method, which ends up being mathematically similar to a Backward/Forward sweep method and has no mathematical features of a Newton-Raphson method. The specific construction of the matrices required for the algorithm is numerically convoluted and offers no advantage with a modern cache-based CPU.
In [34] from 1998, the authors apply an inexact Newton-Krylov method to the power flow problem, which has proven its power in large-scale equations of linear systems in other domains. They analyze and compare different pre-conditioning approaches. Naturally, the method works best for very large power systems with thousands of nodes.
In [19] from 2014, a comprehensive computational optimization of the Fast Decoupled Power Flow method is laid out, including optimal usage of SIMD using SSE and AVX. Additionally, a parallelization scheme for a contingency scheme is presented.
In [70] from 2015, a GPGPU implementation for the Fast Decoupled Power Flow method is presented. The thesis is focused on large transmission grids, and shows that there are no significant performance gains.

$\mathbf{Z_{BUS}}$ Methods

After the initial papers in the 60s [10] [12] [13], the attention on the Z_{BUS} method was dwarfed by the Y_{BUS} Newton-Raphson method. After the proposal of the Backward-Forward Sweep method for an individual feeder, multiple attempts were published to extend the principle to arbitrary grid shapes, without realizing that this inevitably leads to methods which are mathematically identical and computationally inferior to the Z_{BUS} Jacobi method. The original sources are very seldom mentioned, and there has been barely any progress or deeper analysis concerning the original method.
In [98] from 1988, the authors develop a method to topologically construct the relationship between the currents injected in the nodes and the resulting voltages by ordering the network into layers and iterating through it from the furthest node towards the center. They also extend the method to work with meshed networks by introducing breakpoints which act as injection nodes in two 'arms' of the network, and call this approach a "compensation method". It is not clear how the topological operation could be executed in an automated fashion. The method is mathematically identical to the Z_{BUS} Jacobi method, which is not men-

tioned in the paper, but at least alluded to in the answer to a reviewer.
In [17] from 1991, the author introduces a set of component models and briefly introduces the Z_{BUS} Jacobi method under the name 'Z_{BUS} Gauss method'. This paper is frequently cited as a source for the Z_{BUS} method, although it contains no well defined algorithm and there are earlier sources (see above).
In [37] from 1999, the authors devise a comprehensive algorithm to extend the Backward/Forward Sweep method to a radial topology by creating an elaborate numbering scheme for nodes that appear in a branch. This is again a convoluted way to construct $\underline{\mathbf{Z}}$.
In [107] from 2003, the author presents another topology-based approach to construct the matrix $\underline{\mathbf{Z}}$, based on a modified incidence matrix, and calls it a 'direct approach', although the method is again identical to the Z_{BUS} method and therefore requires iterations. The author also includes a normalized execution time comparison against a method based on [17], which is algorithmically identical, and concludes that his method is faster, but there are no implementation details for either method.

Available Power Flow Codebases

There are a number of for power flow software packages available, but only a few are based on completely open-source stack.
The power system analysis software package **OpenDSS** [25] by EPRI is written in Delphi, a dialect of Pascal. It contains a multitude of features to simulate more high level concepts like active operation using voltage regulation devices, reliability, and efficiency. The integrated power flow solver contains implementations of a Y_{BUS} Jacobi method and Y_{BUS} Newton-Raphson method [26]. However, the main power flow feature of OpenDSS is the on-the-fly transformation of electrical devices between nodal current injections and admittances in the Y_{BUS} matrix, to employ a direct solution of the remaining linear equations when the precision requirements are lower, e.g. during fault studies. From a technical standpoint, OpenDSS is built on the COM interface of Microsoft Windows and is not fully compatible with other operating systems.
GridLAB-D [16] by the US Department of Energy is described as a 'modeling and simulation environment'. It is based on an agent-based approach to power systems and includes provisions for market simulations, distribution automation control, and time-series. The power flow itself is conducted using a Y_{BUS} Newton-Raphson method, and there is

no particular focus on performance.
PYPOWER is a power flow and optimal power flow library written for use with Python. It is derived from the Matpower library for the commercial numerics software MATLAB. PYPOWER contains an implementation of the Newton-Raphson method which is algorithmically solid, but computationally poor. Pandapower [108] is a more comprehensive power system analysis suite built on top of PYPOWER which includes a slightly more computationally performant construction of the Jacobi matrix using the JIT-compiling functionality of *Numba*. A comparison to the runtime of PYPOWER is included with every practical example in chapter 6.

1.3 Contributions of the Thesis

Any contribution to a numerical simulation of a physical system happens in one or more of the steps outlined in Figure 1.1.

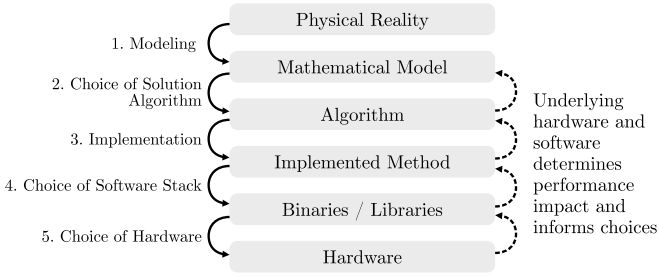


Figure 1.1: The stack of numerical computations of physical systems

Each of the five distinct steps can be optimized to achieve a better runtime for the entire simulation. In the case of power flow computations, many existing approaches are top-down: The fast-decoupled power flow method, for example, defines a modified mathematical model in step 1, in order to accelerate the subsequent solution with a conventional Y_{BUS} Newton-Raphson method.
This thesis presents a bottom-up approach: At first, a selection of existing algorithms is optimally implemented based on the properties of

the numerical interface defined by the underlying hardware and low-level numerical software. The optimized implementations then allow for an informed choice when choosing an algorithm for a particular problem. It turns out that, for the specific large-scale problems that occur in distribution grid planning, the Z_{BUS} Jacobi method can be remarkably effective. The thesis outlines a novel extension to the method to work with grids with multiple slack nodes in order to contribute to its wider applicability. Finally, the optimization effort is put into context by evaluating more general acceleration approaches and presenting results from three large-scale example problems.
The individual research goals of this thesis are as follows:

Research goal	**Chapter**
Establish the foundations of computationally performant numerical methods for modern CPUs and assess their impact on power flow methods	2
Derive, define and classify the most relevant existing power flow algorithms including the modeling assumptions	3 & 4
Find and note the most performant implementations of the chosen power flow algorithms	4
Fill the gaps in the applicability of the highly performant Z_{BUS} method, especially in grid models with multiple slack nodes, which occurs frequently in high-voltage grids	4.7
Explore the performance impact of popular acceleration approaches in the context of the computationally optimized methods and find new approaches suitable for large-scale time series problems in distribution grids	5
Demonstrate the achievable performance of the optimized methods and acceleration approaches by solving example problems characterized by large-scale time series	6

Because the thesis is motivated by distribution grid planning, model simplifications related to a specific R/X ratio, which are the basis of many acceleration approaches tailored to transmission grids, are not considered. The conventional nodal grid model is regarded as given throughout the thesis except for the grid reduction approaches in Section 5.3.1. There is also no attempt to find a new solution algorithm, as there is

a wealth of ultimately fruitless attempts in this direction already. The first two steps in the numerical stack in Figure 1.1 are therefore not considered as degrees of freedom in the search for optimal performance.
The focus is also not on a choice of specific hardware platforms in step 5 in Figure 1.1, like GPGPUs, FPGAs, or specific CPUs built for scientific calculations. The computations are all performed on a standard x86/x64 platform. Acceleration efforts are concentrated on the implementation (step 3), guided by an optimal choice of underlying numerical routines in step 4.
Finally, there is no attempt to reach rigorous mathematical assertions related to power flow computations. All discussed factors are in the interest of computational performance.

1.4 Structure of the Thesis

The thesis is organized in 7 chapters which build on each other linearly. In Chapter 2, the foundations of computational performance and the conditions for an efficient program on modern CPUs are introduced. Building on those foundations, a set of performance indicators and the benchmarking methodology are established.
In Chapter 3, the process of building a mathematical model from a real-world electric grid is discussed in detail. After a general discussion of complexity vs. computability in the context of electric grids, the Y_{BUS} formulation is introduced, together with a guide on how to handle slack nodes and the available convergence criteria.
In Chapter 4, six different solution methods are introduced and discussed. The mathematical foundations and efficient algorithms are presented. A discussion about their computational performance using the indicators established in Chapter 2 follows. Additionally, some complicating real-world grid elements and circumstances, like multiple slack nodes, complex load characteristics, or transformers are discussed and their impact on the power flow model, the solution method, and the performance is presented.
In Chapter 5, some optimization approaches for the runtime of power flow computations are presented. These include implementation details like sparse matrix representations, acceleration factors, and parallelization as well as grid reduction methods and heuristic methods to skip certain power flow computations altogether.
In Chapter 6, the presented power flow methods and optimization ap-

proaches are applied to three real-world problems, and several solution strategies are discussed for each. The problems are time-series power flow computations in a low voltage grid, a Monte-Carlo simulation to evaluate the impact of electric vehicle charging on a low-voltage grid and n-2-contingency analysis in a high voltage grid.
Chapter 7 concludes the thesis with a summary.
The appendix is comprised by an overview of raw floationg point operation counts of the BLAS/LAPACK routines used throughout the thesis, a benchmark of the dominating BLAS/LAPACK operations, and the detailed node and line data for the grids used in the examples in sections 6.2 and 6.3.

Chapter 2

Computational Performance

The computational performance of power flow algorithms is the central research subject of this thesis. It is therefore warranted to discuss the broader context of computational performance today.
The features of modern CPUs, like caches, branch prediction, and out-of-order execution are introduced and their impact on the optimal formulation and implementation of power flow methods is discussed. These impacts lead to a set of guidelines that the later chapters follow. Finally, the benchmarking methodology for the remainder of the thesis is presented.

2.1 A Performance Comparison Comparison

In their influential 1968 book 'Computer Methods in Power System Analysis'[102], the authors Glenn Stagg and Ahmed El-Abiad show a comparison between different power flow methods in terms of iteration count and computing time per iteration. Their findings on iteration count are of course timeless - the mathematics of iteration and convergence has not changed. The computation time per iteration however has changed dramatically between 1968 and 2021. Figure 2.1 (a) shows their Figure 8.16 from the book which indicates the performance of their methods on

the computers of the day.

Regrettably, they chose to measure the time in term of unspecified

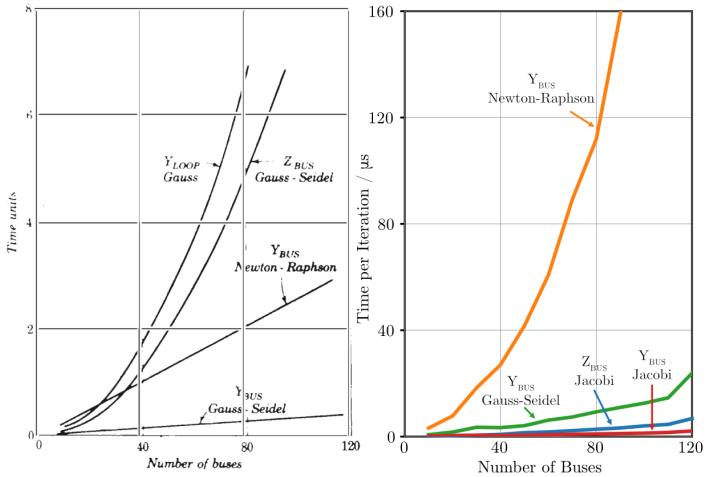


Figure 2.1: Comparison of time per iteration of power flow methods from (a) 1968 [102], most likely on an IBM 650 or IBM 70-series and (b) 2021, created by the author on an Intel Core i7-8550U.

'time units', although its plausible for the era that the unit is actually minutes. The exact grid is also unspecified, but the focus of the book on transmission grids indicates that the grids are probably meshed and have a high voltage level.

Notably, their time per iteration for the Z_{BUS} method (Gauss-Seidel in this case) increases exponentially with the number of nodes, and timings for the Y_{BUS} Newton-Raphson and Y_{BUS} Gauss-Seidel methods increase linearly. The authors chose to omit the Jacobi method completely (called "Gauss" method in the book), because, as they claim, "the time per iteration for these two methods [the Jacobi and Gauss-Seidel methods] is about the same". The qualitative hierarchy between the methods is clear: For all but the smallest grids, the Y_{BUS} Gauss-Seidel method has the fastest computation time per iteration, the Y_{BUS} Newton-Raphson method comes next, and the Z_{BUS} method is slower still. The Y_{LOOP} method is a power flow method based on manually drawn loops in the grid which is irrelevant today.

Figure 2.1 (b) shows a recreation of the same plot on a CPU from 2018 with the timings of the methods developed in this thesis, as faithfully to the original as possible regarding the lack of information about the exact grid and loads used.
The comparison of the two plots shows that the runtimes have changed not only quantitatively - if the 'time unit' is indeed minutes, the times per iteration are around 3 million times faster - but also qualitatively. First, the comparison between the Y_{BUS} Gauss-Seidel (green line) and Y_{BUS} Jacobi (red line) methods shows that their timings are not the same at all, the Jacobi method is around 10 times faster. Most dramatically, the Z_{BUS} method (blue line) that was among the slowest in 1968 is now faster than the Y_{BUS} Gauss-Seidel and Newton-Raphson methods in terms of time per iteration. The Y_{BUS} Newton-Raphson (orange line) method stands out as comparatively slow in 2021. The scaling of all methods with respect to the number of nodes has also changed - the time per iteration of all methods now grows exponentially with node count. So, what happened in the last 50 years that not only sped up these computations by a factor of millions, but also put the rules of slow and fast numerical computing on its head?
The answer lies in the architecture of modern CPUs, which over the years has exploded in complexity to maintain the rate of progress in performance. As a result, CPUs today have different performance characteristics and favor a different program design than the machines that Stagg and El-Abiad worked with.

2.2 Modern CPU Architecture

As shown, computers have undergone an explosive development since the days of the first power flow computations and are now magnitudes faster. The raw performance metrics of CPUs are particularly impressive: The IBM 650 used for the first power flow computations in the fifties achieved 5000 scalar multiplications per *minute* [48], the performance of a modern CPU is measured in GFLOPS (billion floating point operations per second).
However, a computer consists of more than just the CPU core which executes these operations. The devices supplying the data such as memory, disk storage and the connecting buses also contribute to the overall performance of program execution. Although there has been major progress in these areas as well, modern CPUs can now process data much faster

than it can be transmitted to them from conventional memory [33]. In response, *CPU caches* and techniques like instruction pipelines and branch prediction (all further explained below) were developed. This has a lead to a complex *data pipeline*, which can speed up the data transmission to the CPU core, but whose effectiveness can vary greatly between different programs. As a result, the adaption to this data pipeline is the main challenge of computational performance and the main optimization approach for silicon, microcode, driver and library development today [46]. These developments have in turn shaped the requirements for performant program design and implementation. Algorithms are now computationally performant if they can leverage CPU caches, instruction pipelining, branch prediction and other techniques to their advantage, and not only if they perform their task in the fewest steps or show good theoretical scaling behavior (usually expressed in *Landau* or *Big-O notation*, e.g. $\mathcal{O}(n^2)$, which represents an upper bound on the function that describes the computational complexity of an algorithm.)
Figure 2.2 shows the basic building blocks of a modern CPU and the full data pipeline. The central computing elements for the numerical tasks required for power flow computations are FPUs (Floating-Point Units). The Intel 7260U CPU which is used for the main benchmarks in this thesis has two cores which each contain one FPU.
The Figure also shows the rough timings [68] of the data transfer between the CPU core, the caches, RAM, and the disk, including a disk cache.

Caching

Caches are a smaller, but much faster segments of memory built directly into the processor [82]. They are shown in green in Figure 2.2. CPUs today usually have 3 levels of builtin cache, L1, L2, and L3, with increasing size but decreasing speed. The memory management engine of the CPU loads portions of memory into the caches, either on specific request of an instruction or speculatively (prefetching [8]).
If an instruction requests a piece of data that is not in the cache, that is called a *cache miss* and carries a large performance penalty [23]. The data then has to be loaded from the main memory, which on normal systems takes in the neighborhood of 50 ns (150 CPU cycles on a core with 3 GHz clock frequency) instead of around 1 ns (3-4 CPU cycles) [46]. If the CPU needs that data to continue, which it usually does, the CPU core is *stalling* in that time and sits idle.

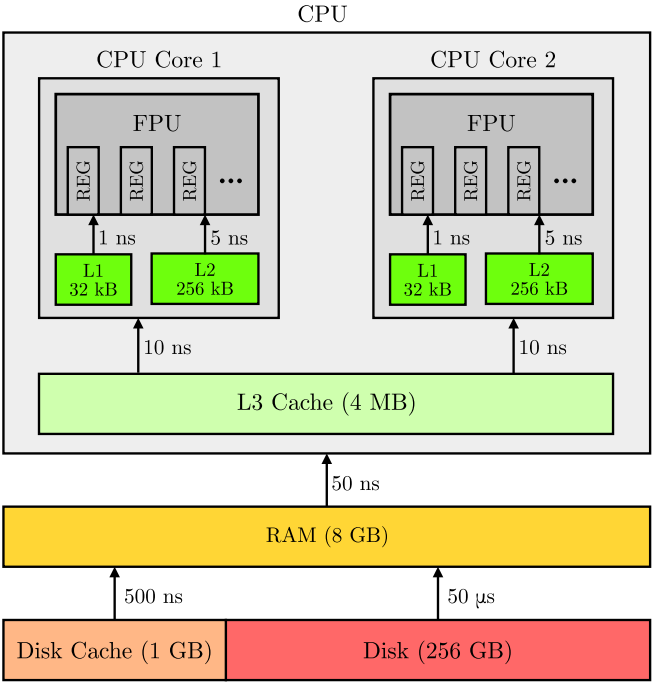


Figure 2.2: Basic memory hierarchy of the Intel i5-7260U CPU and system used for all benchmarks throughout this thesis. The timings show the large differences between data load times from different sources and highlight the importance of *caching*.

The caches always request and store data in fixed portions, the *cache lines*, which, are usually 64 Bytes long today [52]. For the double-precision floating point numbers that are common for scientific computing problems, this means that the cache is always filled with 8 numbers that were contiguous (aligned in series) in memory. This is advantageous if the memory access is uniform, and data is loaded and operated upon in series, because in that case the next value already sits in the cache and does not need to be loaded at all. On the flip side, non-uniform, random memory accesses trigger frequent cache misses and can severely hurt performance.
The number of cache loads and misses can be measured and serves as an important indicator for the efficiency of an implementation.

Translation Lookaside Buffers

All of the memory of a computer, including the caches, is organized as *virtual memory* by the operating system. This provides every process with a virtual, uniform memory segment, even when its fragmented in the physical memory or overflows onto the hard drive (*paging*). The translation of virtual to physical memory addresses is managed by the kernel and is supported by another cache: the *translation lookaside buffer* (TLB) [46]. It exists for the data of a process (DTLB) as well as for the instructions themselves (ITLB). A TLB miss also causes a severe delay while the data has to be loaded from memory. This also highlights another property of all modern CPUs which follow the *von Neumann architecture*: Instructions are also data which sit in memory, and a uniform access pattern is just as important for instructions as it is for data. If a program contains a *hotloop* which makes up a large part of the overall programs' runtime (which is definitely the case for iterative power flow methods), this loop should contain as few instructions as possible in order not to overwhelm the instruction TLB.
The number of ITLB and DTLB misses is another metric which can be measured.

Pipelining & Branch Prediction

Before an instruction can be executed, it itself and the associated data needs to be loaded from memory (via the respective TLBs) into the

registers, which are the memory units closest to the actual computation units of the CPU. The instruction then has to be decoded to set up the FPU, is then executed and the result is written back into memory. These are four distinctive steps (Instruction Fetch, Instruction Decoding, Execution, Write Back) that have to occur in series, and each take at least one CPU cycle [106]. However, multiple successive instructions can be in different stages of the pipeline, so that one instruction can finish per clock cycle. This works well as long as there are no *branches*, i.e. choices between two instructions dependent on the result of a previous instruction, usually as a consequence of an if-statement or a loop instruction in the code. In order to prevent stalling, modern CPUs take an informed guess about the result of previous instructions and enter the instruction they deem most likely to follow into the pipeline. This mechanism is called *branch prediction* or *speculative execution*. Its implementation inside the CPU has evolved from a simple state machine [101] to a complex statistical algorithm which increasingly also employs machine learning techniques and neural networks [117]. When the branch prediction fails (a *branch misprediction*), the pipeline has to be cleared and the new, correct, state has to be entered into the pipeline. In order to support the branch prediction, branching code should be laid out carefully. If-statements and loops that show regular patterns during the execution don't interrupt the pipeline more than necessary. Unpredictable patterns can cause frequent CPU stalling.

Out-of-Order Execution

Another feature that differentiates modern CPUs from the processors from a few decades ago is their *superscalar* nature [106]. The *execution units* of the CPU that do the actual work are all highly specialized (arithmetic logic unit (ALU), floating-point unit (FPU), load-store unit (LSU), etc.) and would sit idle most of the time if all instructions were processed in series. In order to use all execution units optimally, modern CPUs internally rearrange and parallelize instructions. Instructions that occur frequently together can even be combined into a single instruction (*μop-fusing* [36]). A program that can take advantage of these capabilities can be executed with more than one instruction per cycle, even without explicit parallelization.

One important conclusion from these features is that *computational performance* is different from *algorithmic efficiency* [99]. A good algorithmic efficiency results in a low number of atomic, numerical operations, but this does not directly mean the algorithm is computationally performant. Another algorithm, which is theoretically worse, can be faster to execute due to its better computational properties if it is better adapted to the technologies discussed above.
One prominent example for this is the basic operation of matrix multiplication: The naive implementation uses three nested for-loops, leading to a theoretical complexity of $\mathcal{O}(n^3)$. There are algorithms which realize a 2D matrix multiplication with algorithmic complexity $\mathcal{O}(n^{2.3727})$ (modified Coppersmith-Winograd, [122]). However, the naive implementation is so well suited to modern CPUs that it almost always performs better in practice, and the theoretically better algorithm is only used for special applications with huge matrices with billions of elements.
All the features listed above lead to guidelines about how performant programs should be designed and implemented, which are outlined in the following section.

2.3 Guidelines for the Design of Performant Numerical Algorithms

Optimizing code to leverage all these principles can be an arduous task and requires knowledge of low-level programming languages and vendor-specific functions, which severely limits the applicability to different computational architectures. Luckily, for numerical problems like power flow, there are libraries which provide low-level numerical vector and matrix operations in highly optimized implementations. These libraries are the basis of many numerical programs like MATLAB or Octave and higher-level libraries like Numpy for Python and Boost for C++. The oldest and most basic of these is the *BLAS* (Basic Linear Algebra Subprograms), which provides routines for basic vector and matrix algebra like multiplication. The name BLAS stems from a specific Fortran library from 1979 [65], which was based on an initial specification from 1973 [43]. Over the years the actual code has been re-implemented many times to optimize the performance on many different *ISA*s (Instruction Set Architectures), but the interface, i.e. the names, functionalities,

and parameters of the routines remains as a de-facto standard. Today, optimized BLAS binaries are crucial for the real-world performance of a CPU and therefore usually maintained and distributed by CPU manufacturers directly.
Based on the BLAS routines, *LAPACK* [61] provides more high-level matrix operations like LU-decomposition, inversion, or solution of linear systems $\mathbf{Ax} = \mathbf{b}$, based heavily on BLAS routines. In the same vein as the BLAS, it has been re-implemented by CPU manufacturers to provide optimal performance, but the interface also stands as a de-facto standard.
Today, CPU manufacturers usually provide these numerical routines together with more mathematical functions in a unified package (Intel: MKL [53], AMD: AOCL [5]), which makes use of special instruction set extensions and the properties of the specific CPU architecture. The collection of these functions defines the heavily optimized *numerical interface* of the system. Figure 2.3 outlines the relationship between the numerical packages for the example of an Intel System.

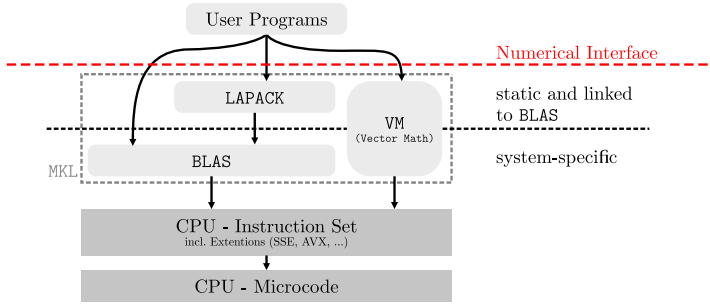


Figure 2.3: Numerical Interface and packages of modern Intel CPUs.

As a result, the functions of the numerical interface are often magnitudes faster than handwritten implementations, especially in high-level languages like Python, which was used for all implementations in this thesis. In fact, one goal during the optimization of the methods devised in this thesis is to move as many computations as possible to MKL functions, and implement as little functionality as possible explicitly. That way, the performance of the runtime (CPython in this case) itself is largely irrelevant for the computational performance.

For the design of the algorithms, this means that the equations should optimally be transformed in a way that the solution uses straight vector-matrix and vector-vector operations, and not explicit loops, maybe even containing if-conditions (and therefore branches). This is the main goal of the development of the power flow solution methods in Chapter 4.
All modern numerical libraries also feature routines that directly work with complex numbers, this capability should be used for optimal performance. An explicit deconstruction of complex numbers into two real components, Cartesian or polar, should be avoided whenever possible. In the algorithms that are outlined in Chapter 4, most operations are performed with complex numbers, and the pseudo-code assumes that a numerical library with appropriate capabilities is used.
From an architecture standpoint, the program should keep the number of instructions per function low (in order to not overwhelm the ITLB cache) and data access uniform. In practice, this can only ever be achieved for small sections of the entire program. However, a power flow simulation involves a lot of code that is usually not performance-critical, like loading and parsing input data and evaluating the output. It is worthwhile to logically push complex and 'messy' operations into those sections, so that the performance-critical core can perform at its best.

2.4 Benchmarking Methodology

The elaborate optimization schemes of modern CPUs also have interesting consequences for the measurement of computational performance. Although the results of a program are usually deterministic, the runtime of the program is not. The runtime is influenced by many factors outside the programmers control: The operating system runs a multitude of programs concurrently and divides up the processing time between them, using a *scheduling mechanism* which can be unpredictable. The CPU itself can change its clock frequency due to thermal restrictions. The initial placement of the data in memory can line up favorably in one run and less favorably in the next. When data has already been placed in a cache, execution times can be much faster in subsequent runs.
All this means that the simplest indicator of performance, the *wall time* (time elapsed in the real world), is not always a reliable and transferable indicator of performance. This problem is sometimes sought to alleviate by using the number of CPU instructions, or the number of *FLOPs*

(Floating Point Operations) per second. However, these measurements are not suitable for modern CPU architectures - CPU stalls and cache misses can cause otherwise identical instructions to have wildly different execution times, so these metrics can, in fact, 'hide' a bad implementation. Additionally, a precise measurement of the number of FLOPs on the CPU via instruction counters is not easily possible. Floating point operations are contained in many vectorized instructions from instruction set extensions such as SSE and AVX, which can execute a variable number of individual FLOPs in one instruction.
A software-implemented version of FLOP measurement was briefly included in MATLAB until it was removed in 2000 because it delivered wildly incorrect and practically useless results [116] [93]. FLOP counts were therefore used in some papers in the 90s as proxies for performance [126], but these results were questionable at the time and are not applicable today.
In this thesis, the methods developed in Chapter 4 are first evaluated in detail in section 4.5 using the performance metrics established in section 2.2. In the later chapters, the wall time is used as the only performance metric. In order to minimize all disturbing influences, all non-essential processes were terminated prior to running the benchmarks. The clock speed was monitored through all tests to ensure that the CPU does not throttle and put later tests at a disadvantage.
Technically, all the benchmarks use the UNIX system call *gettimeofday()* either directly or via the Python *time* module which provides a microsecond accuracy. Unless otherwise stated, all benchmarks are performed on a standard Intel NUC 7i5 BNK with the specs outlined in table 2.1.

Table 2.1: Specification of the computer that was used for all benchmarks, unless otherwise specified

CPU	Intel Core i5-7260U @ 3.4 GHz
RAM	8 GB DDR4 @ 2400 MHz
HD	256 GB SSD Samsung 860 Evo

The CPU was released in early 2017, and it represents commodity hardware that is representative of standard office computer, not a specialized high-performance system.

The software stack is standard for a Python/Numpy environment in 2019, the main software packages used are shown in table 2.2.

Table 2.2: Software versions used for the benchmarks in this thesis

Software	**Version**
Linux Kernel	4.15.0-76-generic
OS	Linux Mint 19
CPython	3.7.6
Numpy	1.17.3
MKL	2019.4
Numba	0.47.0
LLVM	8.0.1

Chapter 3

Principles of Power Flow Computations

In this chapter, the formulation of the power flow problem is revisited, including the formation of the grid model, and the considerations that lead to that specific model. A simple example grid is defined and its power flow model is derived in detail. The formulation in terms of vectors and matrices is introduced, which is crucial for the design of computationally performant solution methods in Chapter 4. The handling of slack nodes is presented in a separate section, as well as the possible convergence criteria that indicate a correct power flow solution.

3.1 The Power Flow Model

The foundation of any power flow model is the equivalent electrical circuit of the considered power grid. The formulation of the equivalent circuit transforms the real-world physical problem of electrical quantities into a mathematical model. As with all mathematical models of real-world systems, there are considerations to be made about which physical effects to take into account and which ones to ignore or simplify. If all known physical effects were fully considered, the result would be a mathematical model of fantastical complexity. In contrast, a simplistic model might not be useful if it does not mirror the relevant characteristics of the real-world system.
A power flow model therefore needs to strike a balance between *com-*

plexity and *computability*. In general, the complexity of the model needs to be as low as possible, but as high as necessary, whereas the computability needs to be as high as possible, but as low as necessary.
The required complexity of a power flow model is directly related to the requirements of the simulation it is used in. Large-scale economic studies frequently use simple grid models, modeling entire regions as single nodes or ignoring line impedances altogether ("copper plate"). On the other hand, simulations which examine the reliability of individual grids under stress or before major switching operations use more complex grid models which contain every single line and grid element.
The computability of a model can be a problem either due to the required computational effort or, more frequently, due to a lack of appropriate input data. For instance, the temperature of a conductor affects its resistance, and so this is one of the physical effects that influences the power flow (and the power flow in turn influences the temperature, which further complicates matters). The complexity of considering this in the mathematical model is manageable, but the correct ambient temperature data and material parameters are rarely available. In fact, even the base resistance of a line is mostly just estimated using its length and ratings, ignoring influences like contact resistances, material imperfections and ageing. In most situations, the consideration of temperature dependency would therefore only consider a minuscule effect on a quantity that is not precisely known anyway.
This problem occurs with many of the physical effects that could be taken into account. The usefulness of many complicated models is severely limited by the availability and precision of the necessary input data. The data that is available is usually the impedances of the lines and the loads at the nodes in terms of powers. A 'normal' power flow model therefore takes exactly these values into account.
In many practical simulations, this is acceptable, because the purpose of power flow computations is hardly ever to achieve perfectly accurate results on the basis of perfectly available input data. The purpose of most power flow computations is the discovery of trends, the investigation of worst-case situations or the comparison of scenarios concerning generation, consumption, or grid expansion.
In these practical investigations, the actual power flow computations often make up only a part of the simulation. The input data, like impedances $\underline{\mathbf{Z}}$, powers $\underline{\mathbf{S}}$ and the slack voltage(s) $\underline{U}_0$ can be the result of measurements, assumptions, predictions, or statistical considerations. They might represent a real, past or momentary situation or an as-

sumed situation in a near or far future. The output of the power flow computation is just the vector of voltages $\underline{\mathbf{U}}$. The currents $\mathbf{I}$ flowing through the lines and the power flowing through the transformer $\underline{\mathbf{S}}_0$ can be computed from those. The consequence of the simulation then depends on those values and can be an immediate action, a long-term strategic decision or just a statistic. This chain of steps with the power flow computation at its center is outlined in Figure 3.1. A frequent subsequent processing step is the determination of compliance, i.e. a check if all voltages and currents are within the bounds of the applicable norms.

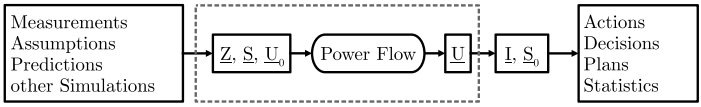


Figure 3.1: Logical Structure of Power Flow Computations

In order to introduce the basic notion of a power flow problem and establish the notation and conventions used in this thesis, a very simple three-node power flow problem is described in the remainder of this section.
Consider the electric grid symbolically pictured in Figure 3.2. Two household consumers are connected to the low voltage grid, which is connected to the overlaying medium voltage grid via a transformer.

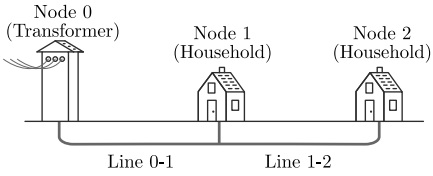


Figure 3.2: Transformer station with two households connected by cables

This is one of the simplest possible grid configurations and it yields one of the simplest power flow problems. However, it highlights which electric effects are usually ignored or simplified in such simple computations. In order to construct the mathematical model of this grid, the following assumptions are made:

Steady State

The power flow computations discussed in this thesis all deal with the *steady state*, which means that all currents, voltages, and powers are assumed to be perfectly sinusoidal with a fixed frequency and therefore defined by their RMS value and phase angle. All currents, voltages, and powers in this thesis are expressed in terms of those two components as complex numbers, unless otherwise stated. Under this assumption, the complex power is $\underline{S} = \underline{U}\,\underline{I}^*$. Effects like harmonics or transient voltage changes are not taken into account and the models and algorithms described in this thesis are not suitable to deal with them without modifications.

Loads

The household loads are all modeled as a fixed complex power $\underline{S} = P + jQ$. This is not entirely correct for e.g. simple heating elements, which, barring temperature influences, have a fixed impedance or LED lighting, which are constant current sinks. However, a major reason to use fixed power is again the availability of data compared to current or impedance, because power is what is actually measured and billed in all residential, commercial, and industrial buildings. In reality, these usually represent many individual loads with complex behaviors under voltage changes, often employing a mix of all three characteristics (constant load, constant current, constant impedance). If the data is available, this can be accounted for by using *ZIP load models* (see section 4.6.1).

Slack Node

The supplying transformer is modeled as a *slack node*, an ideal voltage source with a given, fixed RMS value and angle. It represents the transformer, the supplying medium voltage grid and all other connected electric grids. In reality, the current drawn by the grid influences the transformer and the connected grids, but in the model, this is usually neglected. Therefore, all calculated voltage drops have to be seen in relation to the slack node. This is acknowledged in many grid regulations which specify a tolerated voltage drop relative to the source voltage at the transformer.

Line Model

The impedances and capacitances of overhead lines and cables are simplified. In theory, the line properties are in effect at every infinitesimal piece of the line, which is reflected by the *telegraphers' equations* [75]. These are a system of partial differential equations with electrical quantities expressed as dependent both on time and location on the line. Short of electrodynamic finite element simulations, they represent the most sophisticated model of electric transmission. But even with the assumption of steady-state voltage and current, the resulting equations are too complex for the use in power flow computations, and the potential gains in accuracy are often completely overshadowed by the uncertainty in the values of the given impedances. Lines are therefore usually modeled as one impedance, with stray capacitances distributed onto the endpoints, if available. This leads to the Π-line model shown in figure 3.3.

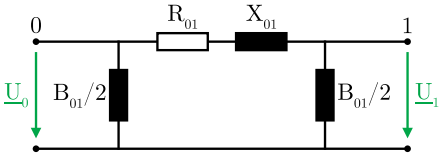


Figure 3.3: Individual line model

In low voltage grids, the parallel reactances are usually neglected, so that a single impedance $Z = R + jX$ per phase remains. For the electric grid pictured in Figure 3.2, the assumptions of steady state, the representation of the transformer as a slack node, the assumption of loads as constant power sinks and the simplified line model yields the three-phase equivalent electrical circuit pictured in Figure 3.4.

Symmetry

Under the additional assumption of *symmetry*, the three-phase system can be further reduced to a one-phase system without any loss of information. The three-phase system is said to be symmetric when the impedances of the lines are identical (which is usually the case in all grids), and the loads at the nodes are the same for all three phases (which is usually approximatively the case in HV and MV grids, but not necessarily in LV grids). In this case, the currents in the neutral con-

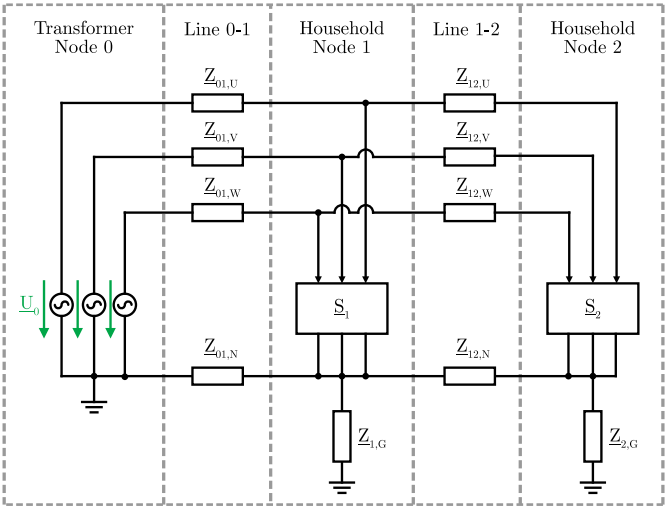


Figure 3.4: Equivalent circuit for the grid pictured in Figure 3.2 under the assumption of steady-state, representation of household loads as constant power sinks, the transformer as a constant AC voltage source, and the lines as single impedances

ductor add up to zero and its impedance can be neglected. The result is a model as pictured in Figure 3.5.

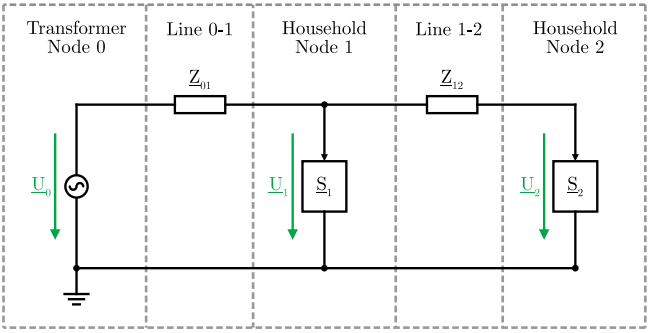


Figure 3.5: Equivalent circuit of the grid in Figure 3.2 under the additional assumption of symmetry

The final step of the power flow model formulation is the translation of this electrical circuit into a mathematical model to which the solution methods can be applied. In order to derive a mathematical model from any electrical circuit, one of the two *Kirchhoff laws* has to be employed. The application of Kirchhoff's Current Law (KCL), which states that all currents flowing into or out of a node of an electric network must sum up zero, gives rise to the Y_{BUS} formulation of the power flow model, which is developed in the next section.
In theory, it is also possible to construct a valid power flow model starting from Kirchhoff's Voltage Law (KVL), which leads to the Y_{LOOP} formulation of the power flow model [102]. This formulation was used in the earliest power flow computations, but it proved to be very complicated for meshed networks. After the first experiences with power flow computations, it quickly became apparent that the Y_{BUS} formulation is the superior method for systematically setting up the mathematical power flow model.

3.2 The Y_{BUS} Formulation of the Power Flow Model

Figure 3.6 pictures the same equivalent circuit as Figure 3.5, but shows only the mathematically relevant parts together with the occurring cur-

rents and voltages. The constant power sinks are represented as current sinks, where the current depends on the voltage at the node.

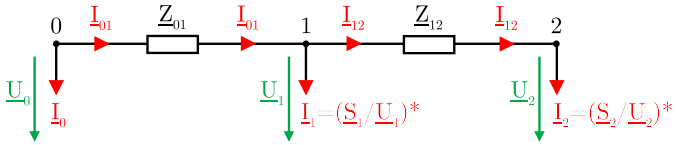


Figure 3.6: Reduced diagram of the equivalent circuit with occurring voltages and currents

Among these values, the impedances $\underline{Z}_{01}$ and $\underline{Z}_{12}$, the powers $\underline{S}_1$ and $\underline{S}_2$, and the slack voltage $\underline{U}_0$ are known, the rest are unknown. The goal of the power flow computation is the calculation of $\underline{U}_1$ and $\underline{U}_2$, from which the unknown currents in the circuit can be computed.
The application of Kirchhoff's current law ($\sum I = 0 \Rightarrow I_{in} = I_{out}$) to the nodes 0, 1, and 2 in the equivalent circuit in Figure 3.6 yields

$$\begin{aligned} 0 &= \underline{I}_0 + \underline{I}_{01} \\ \underline{I}_{01} &= \underline{I}_1 + \underline{I}_{12} \\ \underline{I}_{12} &= \underline{I}_2 \end{aligned} \tag{3.1}$$

or, reordered to separate the load currents $\underline{I}_0$, $\underline{I}_1$, and $\underline{I}_2$:

$$\begin{aligned} & \underline{I}_0 = -\underline{I}_{01} \\ \Rightarrow \quad & \underline{I}_1 = \underline{I}_{01} - \underline{I}_{12} \\ & \underline{I}_2 = \underline{I}_{12}. \end{aligned} \tag{3.2}$$

The ultimate goal of the following operations is to transform these equations so that they contain the known quantities, the power at the nodes $\underline{S}$ and the impedances $\underline{Z}$ as well as the desired result, the voltages at the nodes $\underline{U}$. It should also be possible to set up the equations in a systematic and automated fashion.

The impedances and voltages are directly related to the currents over the lines $\underline{I}_{01}$ and $\underline{I}_{12}$ by Ohm's law:

$$\begin{aligned} \underline{I}_{01} &= \frac{\underline{U}_0 - \underline{U}_1}{\underline{Z}_{01}} \\ \underline{I}_{12} &= \frac{\underline{U}_1 - \underline{U}_2}{\underline{Z}_{12}}. \end{aligned} \tag{3.3}$$

For the model formulation, it is more practical to express the line impedances $\underline{Z}$ as line admittances $\underline{Y} = {}^1\!/\!_{\underline{Z}}$, because a missing line can then easily be described as having an admittance of 0. An impedance of infinity is hard to work with numerically, and would have to be specially accounted for in the equations and the solution algorithms. The substitution of $\underline{Y} = {}^1\!/\!_{\underline{Z}}$ yields

$$\begin{aligned} \Rightarrow \quad \underline{I}_{01} &= \underline{Y}_{01}\,(\underline{U}_0 - \underline{U}_1) \\ \Rightarrow \quad \underline{I}_{12} &= \underline{Y}_{12}\,(\underline{U}_1 - \underline{U}_2)\,. \end{aligned} \tag{3.4}$$

In order to group the voltages as a separate vector, the equations are reordered as

$$\begin{aligned} \Rightarrow \quad \underline{I}_{01} &= \underline{Y}_{01}\underline{U}_0 - \underline{Y}_{01}\underline{U}_1 \\ \Rightarrow \quad \underline{I}_{12} &= \underline{Y}_{12}\underline{U}_1 - \underline{Y}_{12}\underline{U}_2. \end{aligned} \tag{3.5}$$

Equations 3.5 can now be inserted into equations 3.2, relating the load currents to the voltages and impedances:

$$\begin{aligned} \underline{I}_0 &= -\,(\underline{Y}_{01}\underline{U}_0 - \underline{Y}_{01}\underline{U}_1) \\ \underline{I}_1 &= (\underline{Y}_{01}\underline{U}_0 - \underline{Y}_{01}\underline{U}_1) - (\underline{Y}_{12}\underline{U}_1 - \underline{Y}_{12}\underline{U}_2) \\ \underline{I}_2 &= (\underline{Y}_{12}\underline{U}_1 - \underline{Y}_{12}\underline{U}_2)\,. \end{aligned} \tag{3.6}$$

By reordering the terms and sorting by the voltages, the final matrix-vector structure of the equations becomes visible:

$$\begin{array}{rccccccc} & \underline{I}_0 & = & -\underline{Y}_{01}\,\underline{U}_0 & & +\underline{Y}_{01}\,\underline{U}_1 & & \\ \Rightarrow & \underline{I}_1 & = & \underline{Y}_{01}\,\underline{U}_0 & -(\underline{Y}_{01}+\underline{Y}_{12})\,\underline{U}_1 & & +\underline{Y}_{12}\,\underline{U}_2 \\ & \underline{I}_2 & = & & \underline{Y}_{12}\,\underline{U}_1 & & -\underline{Y}_{12}\,\underline{U}_2. \end{array} \tag{3.7}$$

Expressed with matrices and vectors, the equations consist of a current vector, a voltage vector, and a symmetric admittance matrix:

$$\Rightarrow \begin{bmatrix} \underline{I}_0 \\ \underline{I}_1 \\ \underline{I}_2 \end{bmatrix} = \begin{bmatrix} -\underline{Y}_{01} & \underline{Y}_{01} & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \tag{3.8}$$

The symmetric admittance matrix in equation 3.8 encodes the entire information about the modeled electric grid, including the topology and the values of the individual lines in one structure. This is important for the efficient power flow solutions later, as this matrix has to be constructed only once for each individual grid configuration.
For other grid topologies, the admittance matrix can be set up directly, by employing a simple procedure:

1. If nodes i and j are connected by a line with admittance $\underline{Y}_{ij}$, enter this value into positions ij and ji in the matrix. Repeat for all lines.

2. Set the diagonal of the matrix, the positions ii, to the negative of the sum of the corresponding line i in the matrix.

This procedure will yield the admittance matrix even for larger, more complicated meshed grids. The following steps can then be carried out as explained here for the simple three-node grid.
As the complex power $\underline{S}$ of an individual node is defined by $\underline{S} = \underline{U}\,\underline{I}^*$, the power can be incorporated into Equation 3.8.
Again, this step is only necessary because $\underline{S}$ is usually a value that is readily available, whereas the current $\underline{I}$ is not. This is especially true for the complex parts: Active and non-active power are routinely measured and recorded, the phase angle of the current in relation to the voltage, which would be required here, is not.
After introducing $\underline{S}$, the resulting equation is now non-linear with respect to $\underline{U}$:

$$\Rightarrow \begin{bmatrix} \underline{S}_0^*/\underline{U}_0^* \\ \underline{S}_1^*/\underline{U}_1^* \\ \underline{S}_2^*/\underline{U}_2^* \end{bmatrix} = \begin{bmatrix} -\underline{Y}_{01} & \underline{Y}_{01} & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \tag{3.9}$$

The voltage can be transferred to the right side using the *Hadamard product*, which represents the element-wise multiplication of two vectors:

$$\Rightarrow \begin{bmatrix} \underline{S}_0^* \\ \underline{S}_1^* \\ \underline{S}_2^* \end{bmatrix} = \begin{bmatrix} \underline{U}_0^* \\ \underline{U}_1^* \\ \underline{U}_2^* \end{bmatrix} \odot \begin{bmatrix} -\underline{Y}_{01} & \underline{Y}_{01} & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \quad (3.10)$$

Finally, the complex conjugate is usually switched, which is possible because of the involutory property of the complex conjugate:

$$\begin{bmatrix} \underline{S}_0 \\ \underline{S}_1 \\ \underline{S}_2 \end{bmatrix} = \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix} \odot \begin{bmatrix} -\underline{Y}_{01} & \underline{Y}_{01} & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix}^* \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}^*. \quad (3.11)$$

The same equation in vector-matrix notation is

$$\underline{\mathbf{S}} = \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^* \, \underline{\mathbf{U}}^*. \quad (3.12)$$

This formulation is known as the Y_{BUS} *formulation* of the power flow model [102]. The equation contains the vector of complex powers $\underline{\mathbf{S}}$, the admittance matrix $\underline{\mathbf{Y}}$, and the vector of voltages $\underline{\mathbf{U}}$. The task of power flow computations is now the calculation of a voltage $\underline{\mathbf{U}}$ for given complex power $\underline{\mathbf{S}}$ and admittance matrix $\underline{\mathbf{Y}}$.

3.3 Handling of Slack Nodes

Slack nodes in power flow computations are nodes for which a fixed voltage amplitude and angle is specified, instead of an active and reactive power. Mathematically, they represent a boundary condition for the voltages in the grid. Without such a condition, there would be an infinite number of solutions for the power flow, and the problem as formulated by equation 3.11 would not be well posed. Intuitively, such a power flow problem does not contain any information about the voltage level of the grid at all. At least one slack node is required in every classical power flow computation. From the point of view of electrical engineering, slack nodes serve three purposes:

- They anchor the voltage of other nodes with their constant voltage amplitude, which is invariant to changes in current flows or voltage amplitudes at other nodes. The voltages of slack nodes therefore define the actual voltage level a grid is on.

- Because of the constant voltage angle, they provide the angle *reference*. As the voltage angle is a relative property, a voltage angle can never be evaluated in isolation. It only has meaning as a voltage angle difference between two nodes. There is no inherent absolute voltage angle in any node in a grid, so one has to be arbitrarily provided.

- Because of their unspecified power, they provide the power balance in the grid by compensating for the mismatch of consumption and generation of active and reactive power.

Slack nodes are a "fictitious concept" [104] and have no perfect equivalent in the real world. However, in real world problems, the role of slack node has to be taken by real grid elements. Plausible choices are dependent on the type of grid:
In most low and medium voltage distribution grids, the supply of the grid is usually handled by one single transformer, which naturally acts as the single slack node of the grid. It is technically plausible that this node has a constant voltage, provides a voltage angle reference, and delivers or consumes the power mismatch. In fact, a single supplying transformer would not be modeled any other way.
In high voltage distribution grids, it is common to have several supplying transformers at different points in the grid and therefore a necessity to compute power flow problems with multiple slack nodes. This is possible, but it can complicate the handling of the power flow equation and its solution considerably. Section 4.6.2 is dedicated to the solution of power flow problems with multiple slack nodes. Many papers on power flow and available power flow programs do not take multiple slack nodes into account.
In transmission grids, the situation is fundamentally different, as there are no supplying transformers fed from an overlaying grid. The role of slack node in these grids is different and motivated by the mathematical necessity rather than a sensible technical equivalent. Power stations in

transmission grids are usually modeled as *generator nodes*, a third type of node which provides voltage magnitude anchoring through a given voltage magnitude and active power. The slack node is therefore only necessary as a voltage angle reference and for power mismatch compensation. One of the generators has to be 'promoted' to be the slack node, which 'trades' its given active power injection for a constant voltage angle. A single node with a strong generator has to be chosen as slack, although the balancing of load and generation is the task of all controlled generators in the grid. This can be accounted for in an outer optimization loop, which distributes the load mismatch at the single slack node to multiple other nodes which are modeled as load or generator nodes [104].
Distribution grids which are islanded and supplied by distributed generation pose the same problem concerning suitable slack nodes, so research around this topic has arisen with the increased interest around microgrids [92], [59].
In order to account for slack nodes in power flow methods, the equations themselves or the solution method must ensure that during the solution of the power flow, the voltage of the slack node or nodes never changes. For the Y_{BUS} formulation, there are two ways of incorporating a slack node and keeping its voltage invariant directly in the power flow model:

Slack Handling - Option 1

Assuming a slack node at index 0, if the entire line of the admittance matrix $\underline{\mathbf{Y}}$ and the corresponding entry of $\underline{\mathbf{S}}$ is changed as in equation 3.13, the voltage $\underline{U}_0$ becomes invariant:

$$\begin{bmatrix} (\underline{U}_0^* \underline{U}_0)/\underline{U}_0^* \\ \underline{S}_1^*/\underline{U}_1^* \\ \underline{S}_2^*/\underline{U}_2^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01}+\underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \tag{3.13}$$

Mathematically, by setting $\underline{S}_0 = \underline{U}_0^* \underline{U}_0$, the first line of this equation resolves to $\underline{U}_0 = \underline{U}_0$ and will therefore be solved by any value of $\underline{U}_0 \in \mathbb{C}$. In the iterative solution schemes that are presented in Chapter 4, the initial guess will fix $\underline{U}_0$, and repeated iterations to find a solution to equation 3.13 will always yield the initial value of $\underline{U}_0$. As the size of the matrix is unchanged, all other vectors like $\underline{\mathbf{U}}$ and $\underline{\mathbf{S}}$ also remain the same size. Because resizing matrices and vectors is a memory-intensive task, this option is preferred and used throughout the thesis, unless otherwise

stated.
There seems to be no precedent for this method in the literature, and none of the freely available implementations use it.

Slack Handling - Option 2

The line corresponding to the slack node can also be deleted from equation 3.9, yielding equation 3.14.

$$\begin{bmatrix} \underline{S}_1^* / \underline{U}_1^* \\ \underline{S}_2^* / \underline{U}_2^* \end{bmatrix} = \begin{bmatrix} \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \tag{3.14}$$

As $\underline{U}_0$ does not appear on the left side of the equation anymore, it is guaranteed to stay constant. From a performance standpoint, this method has the theoretical advantage of fewer computations due to the missing terms, but the major disadvantage that two versions of the voltage vector $\underline{\mathbf{U}}$, one with and one without $\underline{U}_0$ have to be stored and synced throughout the computation. In addition, this version of the admittance matrix $\underline{\mathbf{Y}}$ can not be inverted.

3.4 Convergence Criteria

The nonlinear equations that arise from the power flow model can usually not be solved directly in a closed form. Instead, a solution is reached iteratively, which requires a criterion of convergence, or breaking or stopping condition, to decide when the result is sufficiently precise and acceptable as a solution. There are three main possible convergence conditions [63]:

Option 1: Power Residual

The power residual $\underline{\mathbf{S}}_R^{(m)}$ of a solution candidate in the mth iteration $\underline{\mathbf{U}}^{(m)}$ is

$$\underline{\mathbf{S}}_R^{(m)} = \underline{\mathbf{S}} - \underline{\mathbf{U}}^{(m)} \odot \underline{\mathbf{Y}}^* \underline{\mathbf{U}}^{(m)*}. \tag{3.15}$$

Intuitively, this power residual is the discrepancy between the given powers at the nodes, and the powers that would be delivered to the nodes

when $\underline{\mathbf{U}}^{(m)}$ is applied to the network. One possible convergence criterion is then

$$\|\underline{\mathbf{S}}_R^{(m)}\|_{max} < \varepsilon_S \tag{3.16}$$

where $\varepsilon_S \in \mathbb{R}$ is the convergence threshold and $\|\cdot\|_{max}$ is the *maximum norm* (also called infinity norm), which represents the entry inside the vector with the greatest absolute value. Another possible convergence criterion is

$$\|\mathrm{Re}(\underline{\mathbf{S}}_R^{(m)})\|_{max} < \varepsilon_P \wedge \|\mathrm{Im}(\underline{\mathbf{S}}_R^{(m)})\|_{max} < \varepsilon_Q \tag{3.17}$$

with separated real and imaginary parts.
The power residual can be directly related to the given powers at the nodes which are input variables to the computation. It makes sense to adjust the convergence criterion according to the precision of the input power $\underline{\mathbf{S}}$. If $\underline{\mathbf{S}}$ is given as integers, it makes sense to choose $\varepsilon_S = 1$ VA and not $\varepsilon_S = 0.001$ VA.
The power residual therefore can bring the computing precision in line with the data precision, which allows for an informed choice of the convergence condition.
The simple criterion 3.16 is used for all power flow computation in this thesis. For the sake of completeness, the other two possible convergence criteria are described below.

Option 2: Current Residual

Another, closely related convergence criterion uses the current residual

$$\underline{\mathbf{I}}_R^{(m)} = \frac{\underline{\mathbf{S}}^*}{\underline{\mathbf{U}}^{(m)*}} - \underline{\mathbf{Y}}\,\underline{\mathbf{U}}^{(m)}, \tag{3.18}$$

which leads to the convergence criterion

$$\|\underline{\mathbf{I}}_R^{(m)}\|_{max} < \varepsilon_I. \tag{3.19}$$

The current residual is obviously closely related to the power residual. For the same power flow problem, the two are approximately interchangeable via the relationship

$$\varepsilon_I = \frac{\varepsilon_S}{|\underline{U}_0|}. \tag{3.20}$$

The only reason to use the current residual is that $\underline{\mathbf{I}}_R$ occurs as an intermediate result during the computation of the Jacobi method as described in section 4.1 and can be directly used for the comparison with ε_I.

Option 3: Voltage Step

The last option for a convergence criterion uses the voltage step

$$\Delta\underline{\mathbf{U}}^{(m)} = \underline{\mathbf{U}}^{(m)} - \underline{\mathbf{U}}^{(m-1)}, \tag{3.21}$$

i.e. the difference in voltage between two iterations. This leads to the convergence criterion

$$\|\Delta\underline{\mathbf{U}}^{(m)}\|_{max} < \varepsilon_U. \tag{3.22}$$

Checking for the step size of the solution variable between iterations is a generally applicable convergence criterion in numerics and is related to the Cauchy convergence test for infinite series. It is usually only used when no other alternative method is available.
This condition is much harder to relate to the other two residuals, as the size of the voltage step does not generally signify convergence to the solution. It could also be a local phenomenon far away from the actual solution. However, a small enough $\Delta\underline{\mathbf{U}}^{(m)}$ usually indicates that the solution is close to convergence. Also, the choice of ε_U does not directly relate to the precision that the iterations ultimately achieve. When convergence according to this criterion has been reached, the actual, precise solution might still be more than ε_U away.

Chapter 4

Power Flow Solution Methods

Based on the Y_{BUS} formulation, six power flow solution methods are presented in this chapter. This includes the mathematical derivation, the development of optimized algorithms formulations and a discussion about the general properties of the methods. The focus is always on the computational performance, as outlined in Chapter 2. This means that the mathematical formulations are at times not the most elegant nor the most straightforward. The presented methods can be divided into three classes:

1. The Y_{BUS} Jacobi, Gauss-Seidel, and Relaxation methods arise from a common fixed-point approach. They are computationally simple but algorithmically inefficient.

2. The Y_{BUS} Newton-Raphson method arises from a root-finding approach to the power flow equations. It is computationally more complex but algorithmically more efficient.

3. The Z_{BUS} Jacobi method is another fixed-point method using an inverted admittance matrix. It can be potentially both algorithmically and computationally efficient. The Backward-Forward sweep method can be seen as a special type of Z_{BUS} method where, in the case of a single-feeder grid, all the computations can be done

using vector algebra.

Most reported 'novel' power flow computation methods can be traced back to these three classes of methods. The relatively new and unique 'holomorphic embedded load flow' (HELM) [111], [112], [91], is not discussed, as its computational speed is reportedly not competitive [96].
A number of complications in grids or the attached elements are also presented and their influence on the power flow model and solution methods as well as the impact on performance is discussed.
Finally, the topic of convergence including the mathematical background, reasons for divergence and possible remedies are discussed.

4.1 Y_{BUS} Fixed-Point Methods

4.1.1 Principle

The family of Y_{BUS} fixed point methods is the most straightforward solution method for the Y_{BUS} equations. In principle, they all consist of iterative update steps to the individual nodes, where the voltage of each node is temporarily brought into harmony with the conditions induced by the given powers, until the entire voltage profile satisfies the convergence criterion.
Mathematically, the solution methods can be directly derived from the original Y_{BUS} formulation. Starting with equation 3.9, it is possible to reformulate the first line as follows:

$$\frac{\underline{S}_0^*}{\underline{U}_0^*} = -\underline{Y}_{01}\underline{U}_0 + \underline{Y}_{01}\underline{U}_1 \tag{4.1}$$

$$\Rightarrow \quad -\underline{Y}_{01}\underline{U}_0 = \frac{\underline{S}_0^*}{\underline{U}_0^*} - \underline{Y}_{01}\underline{U}_1 \tag{4.2}$$

$$\Rightarrow \quad \underline{U}_0 = \frac{1}{-\underline{Y}_{01}} \left(\frac{\underline{S}_0^*}{\underline{U}_0^*} - \underline{Y}_{01}\underline{U}_1 \right). \tag{4.3}$$

In the last equation, $\underline{U}_0$ is isolated on the left side, but the right side still contains it. The other two lines in equation 3.9 can be reformulated accordingly, isolating $\underline{U}_1$ and $\underline{U}_2$, respectively. A formulation like this, or generally of the form $x = f(x)$, lends itself to a solution with *fixed point iterations*, which follow a simple scheme, where, after an initial guess $x^{(0)}$, the equation $x^{(m+1)} = f(x^{(m)})$ is solved repeatedly for $x^{(m+1)}$ un-

til a predefined convergence condition (see section 3.4) is met.
The theoretical conditions for convergence of general fixed point methods are given by the *Banach fixed point theorem*, which for this problem states that the method converges linearly if, among other conditions, $f(x)$ is a *contraction mapping*. For most practical power flow problems in distribution grids, convergence itself is not a problem. In the situations where it is, a multitude of techniques [2],[80] and entirely different algorithms [111] are available.
The general fixed point formulation for the Y_{BUS} formulation of a power flow problem with n nodes is

$$\underline{U}_i = \frac{1}{\underline{Y}_{ii}} \left(\frac{\underline{S}_i^*}{\underline{U}_i^*} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \underline{Y}_{ij} \underline{U}_j \right) \quad \forall\, i = 1 \ldots n. \tag{4.4}$$

Computationally, this equation is not very elegant, as it requires a loop for the sum and an additional condition to handle the exception for $j \neq i$. In a practical program, this would result in poor CPU utilization due to the branches and inevitable cache and branch misses. A better formulation can be reached by transforming the vectors $\underline{\mathbf{S}}$ and $\underline{\mathbf{U}}$ and the matrix $\underline{\mathbf{Y}}$ to contain those loops and exceptions implicitly in vector-vector and vector-matrix operations.
Starting with equations 4.4, there are two ways to achieve this:

Vectorization - Option 1
First, equation 4.4 can be augmented with effectively 0 as follows:

$$\underline{U}_i = \frac{1}{\underline{Y}_{ii}} \left(\frac{\underline{S}_i^*}{\underline{U}_i^*} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \underline{Y}_{ij} \underline{U}_j \right) - \left(\frac{\underline{Y}_{ii} \underline{U}_i}{\underline{Y}_{ii}} \right) + \left(\frac{\underline{Y}_{ii} \underline{U}_i}{\underline{Y}_{ii}} \right) \quad \forall\, i = 1 \ldots n. \tag{4.5}$$

The negative augmenting term then corresponds exactly to the term that is excluded in the sum and eliminates the exception for $j = i$. The

positive term remains. $\underline{Y}_{ii}$ can be eliminated and the entire term can be pulled to the front:

$$\underline{U}_i = \underline{U}_i + \frac{1}{\underline{Y}_{ii}} \left(\frac{\underline{S}_i^*}{\underline{U}_i^*} - \sum_{j=1}^{n} \underline{Y}_{ij} \underline{U}_j \right) \quad \forall\, i = 1 \dots n. \tag{4.6}$$

Using the diagonal of the admittance matrix $\underline{\mathbf{Y}}$ as the vector $\underline{\mathbf{Y}}_{\text{diag}}$, this equation can be vectorized and put into an iterative form as

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{U}}^{(m)} + \left(\left(\underline{\mathbf{S}}^* \oslash \underline{\mathbf{U}}^{(m)*} - \underline{\mathbf{Y}}\, \underline{\mathbf{U}}^{(m)} \right) \oslash \underline{\mathbf{Y}}_{\text{diag}} \right). \tag{4.7}$$

Vectorization - Option 2
Alternatively, the condition for $i \neq j$ can be eliminated by applying the substitutions

$$\hat{\underline{Y}}_{ij} = \begin{cases} \frac{\underline{Y}_{ij}}{\underline{Y}_{ii}}, & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases} \qquad \forall\, i, j = 1 \dots n, \tag{4.8}$$

and

$$\hat{\underline{S}}_i = \frac{\underline{S}_i^*}{\underline{Y}_{ii}} \quad \forall\, i = 1 \dots n. \tag{4.9}$$

Both of these substitutions can be applied before any other computations take place.
The fixed point formulation can then be expressed as

$$\underline{U}_i = \frac{\hat{\underline{S}}_i}{\underline{U}_i^*} - \sum_{j=1}^{n} \hat{\underline{Y}}_{ij} \underline{U}_j \quad \forall\, i = 1 \dots n, \tag{4.10}$$

which can also be expressed in a vectorized form as

$$\underline{\mathbf{U}}^{(m+1)} = \hat{\underline{\mathbf{S}}} \oslash \underline{\mathbf{U}}^{(m)*} - \hat{\underline{\mathbf{Y}}}\, \underline{\mathbf{U}}^{(m)}. \tag{4.11}$$

The first option introduces the vector $\underline{\mathbf{Y}}_{\text{diag}} \in \mathbb{C}^n$, but otherwise preserves the structure and contents of all other variables. The easier handling compared to option 2 gives option 1 a practical advantage. The additional addition and Hadamard division do not result in a measurable performance difference, probably because the runtime is dominated by the matrix-vector multiplication $\underline{\mathbf{Y}}\, \underline{\mathbf{U}}^{(m)}$. Therefore, all

algorithms devised and used in this chapter use option 1.
The vectorized fixed-point formulation in equation 4.7 can now be solved with a multitude of so called *splitting methods*. In the following sections, the *Jacobi method*, *Gauss-Seidel method* and the *Relaxation method* are presented and discussed.

4.1.2 The $\mathbf{Y_{BUS}}$ Jacobi Method

The Jacobi Method (also called the *Gauss iterative method* in [102], although no such 'Gauss method' exists) is the simplest and most straightforward solution method for the Y_{BUS} fixed point formulation. It consists of the iterative solution of the fixed-point equation in unchanged order. After one iteration, where every $\underline{U}_i$ has been updated once, the convergence has to be checked using any of the three methods (see section 3.4), and the iteration loop can be exited when convergence is reached.
The Y_{BUS} Jacobi method can profit from the first vectorization option using equation 4.7, because the equation

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{U}}^{(m)} + \left(\left(\underline{\mathbf{S}}^{*} \oslash \underline{\mathbf{U}}^{(m)*} - \underline{\mathbf{Y}}\,\underline{\mathbf{U}}^{(m)}\right) \oslash \underline{\mathbf{Y}}_{\mathrm{diag}}\right)$$

explicitly contains the current residual

$$\underline{\mathbf{I}}_R^{(m)} = \underline{\mathbf{S}}^{*} \oslash \underline{\mathbf{U}}^{(m)*} - \underline{\mathbf{Y}}\,\underline{\mathbf{U}}^{(m)}. \tag{4.12}$$

$\underline{\mathbf{I}}_R^{(m)}$ can therefore be precomputed as an intermediate result to be used in the ongoing iterations as well as the convergence criterion. This leads to algorithm 1, which describes the Y_{BUS} Jacobi method in terms of only vector-vector operations and one matrix-vector multiplication. It contains no branches except for the inevitable convergence check.
The algorithm consists of the allocation of $\underline{\mathbf{Y}}_{\mathrm{diag}}$ in line 1, which is required only once per grid, and an explicit iteration loop containing the precomputation of $\underline{\mathbf{I}}_R$ in line 3, the convergence criterion in lines 4 and 5, and the voltage update step in line 6.
If the check for the convergence criterion was conducted in the head of a while loop, the residual current $\underline{\mathbf{I}}_R$ would have to be initialized to a "safe" value, e.g. $\underline{I}_{R,i} = 2\epsilon_s \underline{U}_i^{(0)}\ \forall i$. Also, this formulation with the separate breaking condition offers the fastest possible code path in a case where the voltage is already in a converged state.
The inner iteration loop of the algorithm (lines 3-6) contains only one

Algorithm 1 Y_{BUS} Jacobi method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, ϵ_S

$\underline{\mathbf{Y}}_{diag} = \texttt{diag}(\underline{\mathbf{Y}})$

loop

  $\underline{\mathbf{I}}_R = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^* - \underline{\mathbf{Y}}\,\underline{\mathbf{U}}$

  if $\|\underline{\mathbf{I}}_R^* \odot \underline{\mathbf{U}}\|_\infty < \epsilon_S$ **then**

    break loop

  $\underline{\mathbf{U}} = \underline{\mathbf{U}} - \underline{\mathbf{I}}_R \oslash \underline{\mathbf{Y}}_{\text{diag}}$

return $\underline{\mathbf{U}}$

matrix-vector multiplication, two element-wise vector divisions and two vector subtractions per loop and contains only five variables, three of which ($\underline{\mathbf{S}}, \underline{\mathbf{Y}}, \underline{\mathbf{Y}}_{diag}$) don't change between iterations. The If-condition leads to one branch per iteration.

Due to the sparse nature of the admittance matrix $\underline{\mathbf{Y}}$, the Jacobi method, like all Y_{BUS} fixed point methods, is a *local* voltage update method in the sense that in each step, the voltage at a node can only be influenced by adjacent nodes. In practice, this means that information about voltage changes has to be 'propagated' through the network, moving at one node per iteration. Figure 4.2 shows the voltage profiles of the first 4 iterations for a very simple problem shown in Figure 4.1 which highlights the propagating voltage updates. The example grid is similar to the minimal example in section 3.1, but contains one more node and one more line.

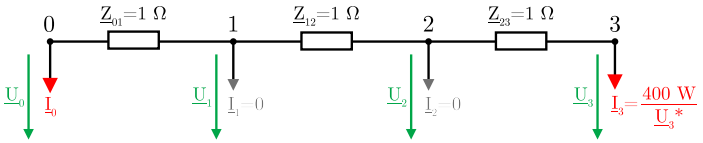


Figure 4.1: Example grid and load scenario for the investigation of the general iteration behavior in Figures 4.2, 4.3, 4.4, 4.5, 4.6, and 4.8.

Iteration 0 shows the initial values for $\underline{\mathbf{U}}$, a 'flat start' with all voltages set to $\underline{U} = \underline{U}_0 = 400\text{V}$, the slack voltage. Because the load only affects one node, in this case node 3, the first iteration leads to a voltage update at only that node. In the second iteration, the voltage at the adjacent node 2 is updated, and in the third, the voltage node 1 is updated

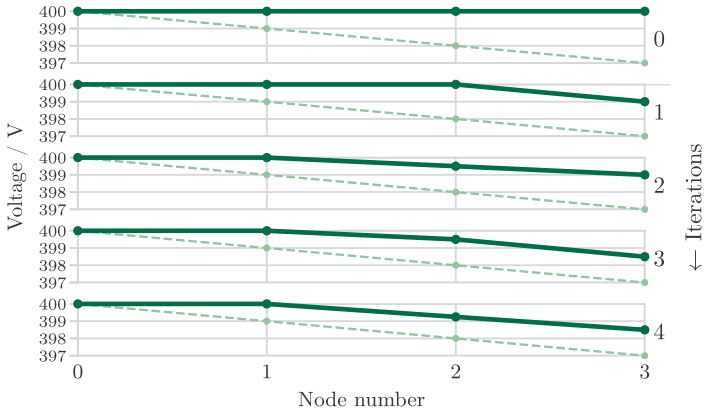


Figure 4.2: Voltage profiles of the first 4 iterations of the Y_{BUS} Jacobi method for a single feeder grid in Figure 4.1. The final solution is shown as the dashed green line.

for the first time together with node 3, which is adjusted because its 'local reference voltage' at node 2 changed. For $\epsilon_S = 1\,\text{VA}$, the method converges after 39 iterations. The example single feeder grid represents a worst-case situation for the Jacobi method, but the same effect leads to high iteration numbers in many situations, especially in large grids with some nodes with very high loads.

The computational performance of the method can be further analyzed by laying out the indivual operations that occur during an iteration in terms of BLAS, LAPACK, and MKL VM routines. Table 4.1 shows all numeric routines that are called by the Y_{BUS} Jacobi method. Futher details about the routines can be found in Appendix A.

Table 4.1: Numerical routines and FLOP estimates for the Y_{BUS} Jacobi method.

Line	Operation	Numerical Interface	Numerical Routine	Total FLOPs Est.
1	$\underline{\mathbf{Y}}_{diag} = \texttt{diag}(\underline{\mathbf{Y}})$			–
3	$\underline{\mathbf{Y}}\,\underline{\mathbf{U}}$	LAPACK	`zgemv()`	$12n^2$
3	$\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}$	MKL VM	`vzDiv()`	$11n$
3	$(\underline{\star})^*$	LAPACK	`zlacgv()`	n
3	$\underline{\star} - \underline{\star}$	MKL VM	`vzSub()`	$2n$
6	$\underline{\mathbf{I}}_R \oslash \underline{\mathbf{Y}}_{diag}$	MKL VM	`vzDiv()`	$11n$
6	$\underline{\mathbf{U}} - \underline{\star}$	MKL VM	`vzSub()`	$2n$
Inner Loop:				$12n^2 + 27n$

Computationally, the time per iteration is the lowest of all methods in this thesis. There are comparatively few operations, and the matrix-vector multiplication in line 3, which dominates the theoretical FLOPs count, contains a high number of zeroes that get immediately resolved in the FPU. The copy operation in line 1 has no FLOPs associated to it and occurs only once for every grid.

4.1.3 The $\mathbf{Y_{BUS}}$ Gauss-Seidel Method

The *Gauss-Seidel method* is an algorithmic refinement of the Jacobi method. Instead of updating the entire voltage vector $\underline{\mathbf{U}}$ all at once after a complete iteration through all lines, the respective value is changed immediately in $\underline{\mathbf{U}}$ and is used for the computations of the following lines. The iteration rule for the Y_{BUS} Gauss-Seidel method is

$$\underline{U}_i^{(m+1)} = \underline{U}_i^{(m)} + \frac{1}{\underline{Y}_{ii}} \left(\frac{\underline{S}_i^*}{\underline{U}_i^{(m)*}} - \sum_{j=1}^{i-1} \underline{Y}_{ij} \underline{U}_j^{(m+1)} - \sum_{j=i}^{n} \underline{Y}_{ij} \underline{U}_j^{(m)} \right) \tag{4.13}$$

$$\forall\ i = 1 \ldots n.$$

In a practical program, the Gauss-Seidel method is implemented with an explicit inner loop which updates one entry of the vector $\underline{\mathbf{U}}$ during every pass. The Gauss-Seidel method is more algorithmically efficient than the Jacobi method, but this inner for-loop is a problem for the

computational efficiency of the method in practice (see section 4.5). Algorithm 2 shows the algorithm for Y_{BUS} Gauss-Seidel using the power residual as a convergence criterion.

Algorithm 2 Y_{BUS} Gauss-Seidel method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, ϵ_S

1: **while** $\|\underline{\mathbf{U}} \odot (\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* - \underline{\mathbf{S}}\|_\infty > \epsilon_S$ **do**

2: **for** i=1...n **do**

3: $\underline{\mathbf{U}}[i] = \underline{\mathbf{U}}[i] - (\underline{\mathbf{Y}}[i,:] \cdot \underline{\mathbf{U}}[:] - (\underline{\mathbf{S}}[i]/\underline{\mathbf{U}}[i])^*)$

return $\underline{\mathbf{U}}$

In this algorithm, the convergence criterion is evaluated in the head of a while-loop, because no part of the convergence check can be precomputed.
The notation in line 3 follows the notation in Python and its numerical library *Numpy*, where $\underline{\mathbf{U}}[i]$ means *"the ith entry of vector* $\underline{\mathbf{U}}$*"*, and $\underline{\mathbf{Y}}[i,:]$ means *"the complete ith line of matrix* $\underline{\mathbf{Y}}$*"*.
Compared to the Jacobi method, every individual iteration is more effective, but computationally slower.
Figure 4.3 shows the voltage profiles afters each of the first four iterations of a trivially simple power flow problem with only on load at node 3, analogous to Figure 4.2.

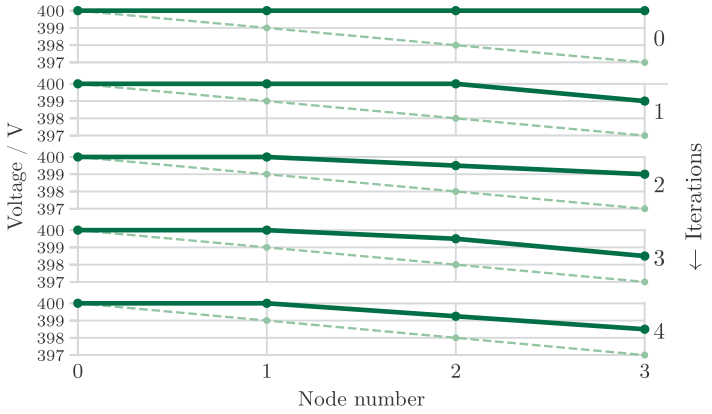


Figure 4.3: Voltage profiles of the first 4 iterations of a Y_{BUS} Gauss-Seidel method for a single feeder grid, 4 nodes, 400 W of load applied to the node 3. The final solution is shown as the dashed green line.

Each iteration of the Y_{BUS} Gauss-Seidel method is more effective than those of the Y_{BUS} Jacobi method - after four iterations, the voltage profile is closer to the eventual solution. Notably, the voltage at node 3 changes in every iteration, which is not true for the Jacobi method. The power flow converges in 20 iterations - roughly half of the iterations required by the Jacobi method for this particular problem.
The computational properties of the Y_{BUS} Gauss-Seidel method are hard to analyze. Most of the individual operation are scalar. All numerical operations are oulined in table 4.2.

Table 4.2: Numerical routines and FLOP estimates for the Y_{BUS} Gauss-Seidel method.

Line	Operation	Numerical Interface	Numerical Routine	Total FLOPs Est.
3	$\underline{\mathbf{S}}[i]/\underline{\mathbf{U}}[i]$	-	-	11
3	$(\underline{\star})^*$	-	-	1
3	$\underline{\mathbf{Y}}[i,:] \cdot \underline{\mathbf{U}}[:]$	MKL VM	`vzMul()`	$6n$
3	$\underline{\star} - \underline{\star}$	-	-	2
3	$\underline{\mathbf{U}}[i] - \underline{\star}$	-	-	2
Inner Loop (n times line 3):				$6n^2 + 16n$

Although the theoretical number of FLOPs is lower, and the iteration count is guaranteed to be equal or lower, the Y_{BUS} Gauss-Seidel method is slower than the Y_{BUS} Jacobi method in practice (see section 4.5). This shows the insignificance of FLOP counts on modern CPUs and the importance of auxiliary CPU features like caching and intrinsic vectorization.
The Y_{BUS} Gauss-Seidel method can be refined by static acceleration (see section 5.1), and multiple other enhancements have been proposed [87], [110].

4.1.4 The Y_{BUS} Relaxation Method

The relaxation method was first described as a solution method for power flow problems by R.H. Jordan in 1957 [57]. The exact name was probably introduced by Stagg and El-Abiad [102].
In principle, the Y_{BUS} Relaxation Method represents an extension of the Y_{BUS} Gauss-Seidel method. Instead of blindly updating line by line of

the power flow equation, this method includes a step in which the node with the biggest deviation from convergence is determined, and only the corresponding line is updated. This yields a more directed iteration approach, at the cost of an extra search for the biggest deviation.
Algorithm 3 shows the entire extent of the operations.

Algorithm 3 Y_{BUS} Relaxation method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, ϵ_S
1: **loop**
2: $\quad \underline{\mathbf{I}}_R = \underline{\mathbf{Y}}\,\underline{\mathbf{U}} - (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
3: $\quad \underline{\mathbf{S}}_R = \underline{\mathbf{U}} \odot \underline{\mathbf{I}}_R$
4: $\quad i_{\max} = \texttt{argmax}(|\underline{\mathbf{S}}_R|)$
5: $\quad$ **if** $|\underline{\mathbf{S}}_R\,[i_{\max}]| < \epsilon_S$ **then**
6: $\quad\quad$ **break loop**
7: $\quad \underline{\mathbf{U}}[i_{\max}] = \underline{\mathbf{U}}[i_{\max}] - \underline{\mathbf{I}}_R[i_{\max}] \,/\, \underline{\mathbf{Y}}[i_{\max}, i_{\max}]$
return $\underline{\mathbf{U}}$

Like the Y_{BUS} Jacobi algorithm, this algorithm also uses the current residual $\underline{\mathbf{I}}_R$ precomputed in line 2 in multiple places. The function $\texttt{argmax}(|\underline{\mathbf{S}}_R|)$ in line 4 yields the index of the largest entry in $|\underline{\mathbf{S}}_R|$ and is directly available in Pythons *Numpy* and other numerical libraries.
The convergence criterion in lines 5 and 6 can directly use the computed $\underline{\mathbf{S}}_R$ and i_{max}. The voltage update step in line 7 then only targets the line with the largest deviation from convergence.
It is noteworthy that one iteration of this method only changes one node voltage, which makes it hard to compare the iteration counts to other methods. If one voltage update at one node is counted as one iteration (which would make one loop of the Jacobi and Gauss-Seidel methods count as n iterations), then the Y_{BUS} relaxation method requires the least number of iterations, but the computational runtime per iteration is much higher.

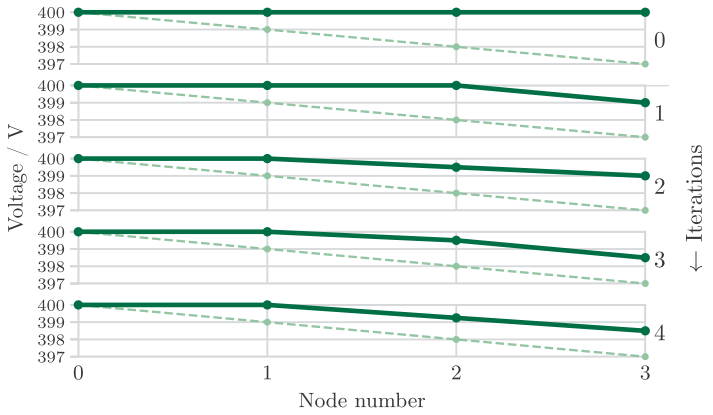


Figure 4.4: Voltage profiles of the first 4 iterations of a Y_{BUS} Relaxation method for a single feeder grid, 4 nodes, 400 W of load applied to the node 3. The final solution is shown as the dashed green line.

The progression of the voltage profile through the iteration looks roughly like the one of the Y_{BUS} Jacobi method in Figure 4.2. However, only one voltage is updated each iteration, first node 3, then node 2, then 3, and then 2 again.

Table 4.3: Numerical routines and FLOP estimates for the Y_{BUS} Relaxation method.

Line	Operation	Numerical Interface	Numerical Routine	Total FLOPs Est.
2	$\underline{\mathbf{Y}}\,\underline{\mathbf{U}}$	LAPACK	`zgemv()`	$12n^2$
2	$\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}$	MKL VM	`vzDiv()`	$11n$
2	$(\underline{\star})^*$	LAPACK	`zlacgv()`	n
2	$\underline{\star} - \underline{\star}$	MKL VM	`vzSub()`	$2n$
3	$\underline{\mathbf{U}} \odot \underline{\mathbf{I}}_R$	MKL VM	`vzMul()`	$3n$
4	$\lvert\star\rvert$	MKL VM	`vzAbs()`	$3n$
4	`argmax`$(\star)$	MKL VM	`izamax()`	n
7	$\underline{\mathbf{I}}_R[i_{\max}]/\underline{\mathbf{Y}}[i_{\max}, i_{\max}]$	-	-	11
7	$\underline{\mathbf{U}}[i] - \underline{\star}$	-	-	2
Inner Loop:				$12n^2 + 21n + 13$

Computationally, the Y_{BUS} Relaxation methods ends up as the least efficient of the Y_{BUS} fixed-point methods. As seen in table 4.3, it can make use of a decent number of vectorized methods, but it is not enough to overcome the algorithmic deficiencies in most practical scenarios. The usefulness of the Y_{BUS} Relaxation method is limited to special situations and conditions, like optimization problem on heavily memory-constrained systems.

4.2 The Y_{BUS} Newton-Raphson Method

The Newton-Raphson method is a widely applied general solution method for nonlinear equations. In its simplest form, the Newton-Raphson method is an iterative solution method for *root-finding problems* of the form

$$f(x) \stackrel{!}{=} 0, \quad x \in \mathbb{R}, f : \mathbb{R} \mapsto \mathbb{R} \tag{4.14}$$

with a known function f and an unknown scalar x. The method defines the iteration scheme

$$x^{(m+1)} = x^{(m)} - \frac{f(x^{(m)})}{f'(x^{(m)})}, \tag{4.15}$$

which, starting with a suitable initial guess $x^{(0)}$, iteratively approximates the x that solves equation 4.14.
For multi-variate problems of the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{x} \in \mathbb{R}^n, \mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n, \tag{4.16}$$

the derivative of $\mathbf{f}$ must be expressed as the *Jacobi matrix* $\mathbf{J}$, which contains all partial derivatives of $\mathbf{f}$ with respect to $\mathbf{x}$, and the iteration scheme therefore is

$$\mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} - \mathbf{J}(\mathbf{x}^{(m)})^{-1}\, \mathbf{f}(\mathbf{x}^{(m)}). \tag{4.17}$$

By introducing an update step $\Delta\mathbf{x}^{(m)} = \mathbf{x}^{(m+1)} - \mathbf{x}^{(m)}$, the iteration scheme can be formulated as

$$\Delta\mathbf{x}^{(m)} = -\mathbf{J}(\mathbf{x}^{(m)})^{-1}\, \mathbf{f}(\mathbf{x}^{(m)}). \tag{4.18}$$

This equation can then be solved for $\Delta\mathbf{x}^{(m)}$ by solving the system of linear equations

$$-\mathbf{J}(\mathbf{x}^{(m)})\Delta\mathbf{x}^{(m)} = \mathbf{f}(\mathbf{x}^{(m)}). \tag{4.19}$$

The solution of linear equations of the form $\mathbf{Ax} = \mathbf{b}$ is a highly optimized function of many numerical libraries (most prominently LAPACK) and therefore directly used in algorithm 4 as the operation $\mathbf{x} = \texttt{solve(A, b)}$. In order to apply the Newton-Raphson method to the power flow problem, the problem has to be reformulated as a root-finding problem by adapting equation 3.12 to

$$\underline{\mathbf{S}}_R = \underline{\mathbf{S}} - \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^* \underline{\mathbf{U}}^* \stackrel{!}{=} \underline{\mathbf{0}}. \tag{4.20}$$

The function $\underline{\mathbf{S}}_R(\underline{\mathbf{U}}) = 0$ is now in the form of equation 4.16. The power residual or *power mismatch* $\underline{\mathbf{S}}_R$ is a function of the state $\underline{\mathbf{U}}$ and has to approach $\mathbf{0}$ for a correct solution.
The complex function is separated into its real and imaginary parts:

$$\underline{\mathbf{S}}_R \stackrel{!}{=} 0 \Leftrightarrow \begin{bmatrix} \mathrm{Re}(\underline{\mathbf{S}} - \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^* \, \underline{\mathbf{U}}^*) \\ \mathrm{Im}(\underline{\mathbf{S}} - \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^* \, \underline{\mathbf{U}}^*) \end{bmatrix} = \begin{bmatrix} \mathbf{P}_R \\ \mathbf{Q}_R \end{bmatrix} = \mathbf{f}(\mathbf{x}) \stackrel{!}{=} \mathbf{0}. \tag{4.21}$$

The voltage $\underline{\mathbf{U}}$ is also separated into its phase ϕ and absolute value $|\underline{\mathbf{U}}|$ to form the state vector

$$\mathbf{x} = \begin{bmatrix} \phi \\ |\underline{\mathbf{U}}| \end{bmatrix} \in \mathbb{R}^{2n}, \quad \phi, \mathbf{U} \in \mathbb{R}^n. \tag{4.22}$$

With these definitions, the Newton-Raphson power flow algorithm can be outlined as follows:

Algorithm 4 $\mathrm{Y_{BUS}}$ Newton-Raphson method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, ϵ_S

loop
  $\underline{\mathbf{S}}_R = \underline{\mathbf{U}} \, (\underline{\mathbf{Y}} \, \underline{\mathbf{U}})^* - \underline{\mathbf{S}}$
  if $\|[\mathrm{Re}(\underline{\mathbf{S}}_R), \mathrm{Im}(\underline{\mathbf{S}}_R)]\|_{max} < \epsilon_S$ **then**
    break loop
  $\mathbf{J} = \texttt{calcJacobi}\ (\underline{\mathbf{Y}}, \underline{\mathbf{U}})$
  $[\Delta\phi, \Delta|\underline{\mathbf{U}}|] = \texttt{solve(} -\mathbf{J}, [\mathrm{Re}(\underline{\mathbf{S}}_R), \mathrm{Im}(\underline{\mathbf{S}}_R)]\texttt{)}$
  $\underline{\mathbf{U}} = (|\underline{\mathbf{U}}| + \Delta|\underline{\mathbf{U}}|) \, e^{j(\phi + \Delta\phi)}$

return $\underline{\mathbf{U}}$

The algorithm starts with the explicit computation of $\underline{\mathbf{S}}_R$ for the currently given $\underline{\mathbf{U}}$. After the convergence criterion in lines 3 and 4, the Jacobi matrix is computed in line 5, the system of linear equation is solved in line 6, and the complex voltage is updated in line 7.
The missing piece of the algorithm is the computation of the Jacobi matrix $\mathbf{J}$ in `calcJacobi()`. From the standpoint of computational performance, this is the step of the Newton-Raphson method with the greatest optimization potential. For many physical problems, the Jacobi matrix has to be computed or approximated using *automatic* or *numerical differentiation*, but because the mathematical model of the power flow is so well known and relatively simple, the Jacobi matrix can be explicitly and analytically correctly computed at a given state $\mathbf{x}$.
For the power flow problem, the Jacobi matrix $\mathbf{J}$ is the matrix that satisfies

$$\delta\mathbf{f}(\mathbf{x}) = \mathbf{J}\delta\mathbf{x} \tag{4.23}$$

$$\begin{bmatrix}\delta\mathbf{P_R}\\ \delta\mathbf{Q_R}\end{bmatrix} = \mathbf{J}\begin{bmatrix}\delta\phi\\ \delta|\mathbf{U}|\end{bmatrix} \tag{4.24}$$

$$\mathbf{J} \in \mathbb{R}^{2n\times 2n}. \tag{4.25}$$

In 1959, J. van Ness gave explicit formulas [115] for $\mathbf{J}$ in the form

$$\begin{bmatrix}\delta\mathbf{P_R}\\ \delta\mathbf{Q_R}\end{bmatrix} = \begin{bmatrix}\mathbf{J_{11}}\mathbf{J_{12}}\\ \mathbf{J_{21}}\mathbf{J_{22}}\end{bmatrix}\begin{bmatrix}\delta\phi\\ \delta\frac{|\mathbf{U}|}{\underline{\mathbf{U}}}\end{bmatrix} \tag{4.26}$$

$$\mathbf{J}_{11}, \mathbf{J}_{12}, \mathbf{J}_{21}, \mathbf{J}_{22}, \in \mathbb{R}^{n\times n}. \tag{4.27}$$

The punch-card computers of the time could not deal with complex numbers directly, and so all expressions had to be completely deconstructed into scalar values. The substitution $\delta|\underline{U}|/\underline{U}$ in the state vector made the formulation of the submatrices easier. This method is still the most popular in the literature and available implementations, although a better way to compute the Jacobi matrix exists. Today, computations with complex numbers are supported by low-level numerical libraries, and thus the complex differentiation with subsequent separation into real and complex parts is computationally more efficient. The result is

a Jacobi matrix that is separated into two submatrices $\underline{\mathbf{J}}_{\phi}$ and $\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}$,

$$\begin{bmatrix} \delta \underline{\mathbf{S}}_{\mathbf{R}} \end{bmatrix} = \begin{bmatrix} \underline{\mathbf{J}}_{\phi} & \underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} \end{bmatrix} \begin{bmatrix} \delta\phi \\ \delta|\underline{\mathbf{U}}| \end{bmatrix} \tag{4.28}$$

$$\underline{\mathbf{J}}_{\phi}, \underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} \in \mathbb{C}^{2n \times n}, \tag{4.29}$$

which can then be further separated into real and imaginary parts to be identical to 4.26.

The analytical expressions for $\underline{\mathbf{J}}_{\phi}$ and $\underline{\mathbf{J}}_{|(U)|}$ are given by Zimmerman [125], but without a derivation or further context. The expressions are used in his popular open-source packages MATPOWER and PYPOWER, but most literature on the $\mathrm{Y}_{\mathrm{BUS}}$ Newton-Raphson method still uses the old expressions from van Ness.

The following analytical derivation of the Jacobi submatrices makes use of the unity matrix $\mathbf{E}$ to bring the values of a vector onto the diagonal of an appropriately sized square matrix, so that e.g. $\mathbf{E}\,\underline{\mathbf{U}}$ means 'the matrix that has the elements of $\underline{\mathbf{U}}$ on its diagonal and 0 everywhere else'. This also allows for the elimination of the Hadamard product.

At every point during the iterations, the power mismatch $\underline{\mathbf{S}}_R$ is the difference between the power at the nodes computed with the voltage vector $\mathbf{x}$ in that iteration and the power $\underline{\mathbf{S}}$ given as an input value. That given power is independent of changes in the voltage, and the differentiation of the power mismatch with respect to the voltage can therefore ignore it:

$$\underline{\mathbf{J}}_{\phi} = \frac{\delta \underline{\mathbf{S}}_{\mathbf{R}}}{\delta\phi} = \frac{\delta\left(\underline{\mathbf{S}} - \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^{*}\underline{\mathbf{U}}^{*}\right)}{\delta\phi} = \frac{\delta\left(\underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^{*}\underline{\mathbf{U}}^{*}\right)}{\delta\phi} \tag{4.30}$$

$$\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} = \frac{\delta \underline{\mathbf{S}}_{\mathbf{R}}}{\delta|\underline{\mathbf{U}}|} = \frac{\delta\left(\underline{\mathbf{S}} - \underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^{*}\underline{\mathbf{U}}^{*}\right)}{\delta|\underline{\mathbf{U}}|} = \frac{\delta\left(\underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^{*}\underline{\mathbf{U}}^{*}\right)}{\delta|\underline{\mathbf{U}}|}. \tag{4.31}$$

For the symbolic derivation of equation 4.30, the product rule of differentiation can be applied to the Hadamard product, yielding

$$\underline{\mathbf{J}}_{\phi} = \frac{\delta\left(\underline{\mathbf{U}} \odot \underline{\mathbf{Y}}^{*}\underline{\mathbf{U}}^{*}\right)}{\delta\phi} = \mathbf{E}\,\left(\underline{\mathbf{Y}}\,\underline{\mathbf{U}}\right)^{*}\frac{\delta\underline{\mathbf{U}}}{\delta\phi} + \mathbf{E}\,\underline{\mathbf{U}}\frac{\delta\left(\underline{\mathbf{Y}}\,\underline{\mathbf{U}}\right)^{*}}{\delta\phi}. \tag{4.32}$$

$\delta\underline{\mathbf{U}}/\delta\phi$ is 0 everywhere except on the diagonal, where it is

$$\frac{\delta \underline{U}_i}{\delta \phi_i} = \frac{\delta\left(|\underline{U}_i| \cdot e^{j\phi_i}\right)}{\delta \phi_i} = |\underline{U}_i| \cdot j \cdot e^{j\phi_i} = j\underline{U}_i \quad \forall\, i = 1...n. \tag{4.33}$$

The first partial derivation in equation 4.32 can therefore be expressed as

$$\frac{\delta \underline{\mathbf{U}}}{\delta \phi} = j\mathbf{E}\,\underline{\mathbf{U}}. \tag{4.34}$$

The partial derivative of the second part of equation 4.32 with respect to the voltage angle is

$$\frac{\delta(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*}{\delta \phi} = \underline{\mathbf{Y}}^* \frac{\delta \underline{\mathbf{U}}^*}{\delta \phi} = \left(\underline{\mathbf{Y}}\,(j\mathbf{E}\,\underline{\mathbf{U}})\right)^* = -j\left(\underline{\mathbf{Y}}\,(\mathbf{E}\,\underline{\mathbf{U}})\right)^*. \tag{4.35}$$

The extra parentheses here are necessary because matrix-vector multiplications are not commutative.
Equation 4.32 can now be written as

$$\begin{aligned} \underline{\mathbf{J}}_\phi &= \mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*\, j\mathbf{E}\,\underline{\mathbf{U}} - \mathbf{E}\,\underline{\mathbf{U}}\, j\left(\underline{\mathbf{Y}}\,(\mathbf{E}\,\underline{\mathbf{U}})\right)^* && (4.36)\\ &= j\mathbf{E}\,\underline{\mathbf{U}}^* \left(\mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* - \left(\underline{\mathbf{Y}}\,(\mathbf{E}\,\underline{\mathbf{U}})\right)^*\right). && (4.37) \end{aligned}$$

The computation of $\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}$ works along the same way:

$$\begin{aligned} \underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} = \frac{\delta \underline{\mathbf{S}}_\mathbf{R}}{\delta |\underline{\mathbf{U}}|} &= \frac{\delta\left(\underline{\mathbf{U}} \odot (\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*\right)}{\delta |\underline{\mathbf{U}}|} && (4.38)\\ &= \mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* \frac{\delta \underline{\mathbf{U}}}{\delta |\underline{\mathbf{U}}|} + \mathbf{E}\,\underline{\mathbf{U}} \frac{\delta\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*}{\delta |\underline{\mathbf{U}}|} && (4.39)\\ &= \mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*\,\mathbf{E} + \mathbf{E}\,\underline{\mathbf{U}}\,\underline{\mathbf{Y}}^* && (4.40)\\ &= \mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* + \mathbf{E}\,\underline{\mathbf{U}}\,\underline{\mathbf{Y}}^*. && (4.41) \end{aligned}$$

To summarize, the Jacobi matrix can be computed with

$$\begin{aligned} \underline{\mathbf{J}}_\phi &= j\mathbf{E}\,\underline{\mathbf{U}}^* \left(\mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* - \left(\underline{\mathbf{Y}}\,(\mathbf{E}\,\underline{\mathbf{U}})\right)^*\right) && (4.42)\\ \underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} &= \mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^* + \mathbf{E}\,\underline{\mathbf{U}}\,\underline{\mathbf{Y}}^* && (4.43) \end{aligned}$$

$$\mathbf{J} = \begin{bmatrix} \mathrm{Re}\left(\underline{\mathbf{J}}_\phi\right) & \mathrm{Re}\left(\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}\right) \\ \mathrm{Im}\left(\underline{\mathbf{J}}_\phi\right) & \mathrm{Im}\left(\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}\right) \end{bmatrix} \tag{4.44}$$

This series of algebraic matrix operations is computationally faster than the explicit computation of the 4 submatrices proposed by van Ness. The expressions $\mathbf{E}\,(\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*$ and $\mathbf{E}\,\underline{\mathbf{U}}$ can be precomputed and used in multiple places in the computation.
The complete $\mathrm{Y_{BUS}}$ Newton-Raphson algorithm is outlined below:

Algorithm 5 $\mathrm{Y_{BUS}}$ Newton-Raphson method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, ϵ_S

loop
  $\underline{\mathbf{I}} = (\underline{\mathbf{Y}}\,\underline{\mathbf{U}})^*$ ▷ precomputation
  $\underline{\mathbf{S}}_R = \underline{\mathbf{U}} \odot \underline{\mathbf{I}} - \underline{\mathbf{S}}$
  if $\|[\mathrm{Re}(\underline{\mathbf{S}}_R), \mathrm{Im}(\underline{\mathbf{S}}_R)]\|_{max} < \epsilon_S$ **then**
    break loop
  $\underline{\mathbf{I}}_M = \mathbf{E}\,\underline{\mathbf{I}}$ ▷ precomputation
  $\underline{\mathbf{U}}_M = \mathbf{E}\,\underline{\mathbf{U}}$ ▷ precomputation
  $\underline{\mathbf{J}}_\phi = j\underline{\mathbf{U}}^*_M \left(\underline{\mathbf{I}}_M - (\underline{\mathbf{Y}}\,\underline{\mathbf{U}}_M)^*\right)$
  $\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|} = \underline{\mathbf{I}}_M + \underline{\mathbf{U}}_M\,\underline{\mathbf{Y}}^*$
  $\underline{\mathbf{J}} = [\mathrm{Re}\left(\underline{\mathbf{J}}_\phi\right), \mathrm{Re}\left(\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}\right); \mathrm{Im}\left(\underline{\mathbf{J}}_\phi\right), \mathrm{Im}\left(\underline{\mathbf{J}}_{|\underline{\mathbf{U}}|}\right)]$
  $[\Delta\phi, \Delta|\underline{\mathbf{U}}|] = \mathtt{solve}(-\mathbf{J}, [\mathrm{Re}(\underline{\mathbf{S}}_R), \mathrm{Im}(\underline{\mathbf{S}}_R)])$
  $\underline{\mathbf{U}} = (|\underline{\mathbf{U}}| + \Delta|\underline{\mathbf{U}}|)\, e^{j(\phi+\Delta\phi)}$
return $\underline{\mathbf{U}}$

Algorithmically, the Newton-Raphson method is locally quadratically convergent and usually converges in much fewer steps than $\mathrm{Y_{BUS}}$ fixed-point methods. Due to the dense Jacobi matrix, the $\mathrm{Y_{BUS}}$ Newton-Raphson method is a method with 'global' convergence properties. In one iteration, the influence of all node loads on all voltages is considered and all voltages are updated. For the simple example problem with 4 nodes, the $\mathrm{Y_{BUS}}$ Newton method converges in one step.

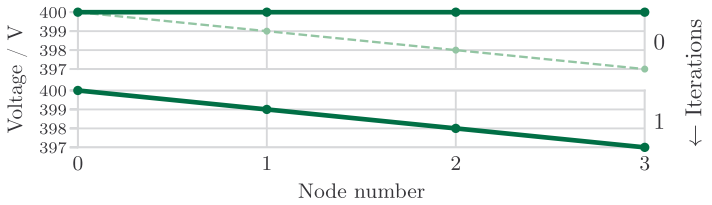


Figure 4.5: Voltage profiles of the first iteration of a Y_{BUS} Newton method for a single feeder grid, 4 nodes, 400 W of load applied to the node 3. The final solution is shown as the dashed green line.

Computationally, the Y_{BUS} Newton-Raphson algorithm involves significantly more variables and memory in the inner loop than the other methods outlined in this thesis. The detailed analysis of the numerical routines in table 4.4 reveals many routines that have high complexities and touch a lot of memory.
The convergence criterion in lines 4 and 5 can be ignored for complexity considerations, as it will be mostly skipped by the branch prediction. The copy operation in line 10 carries no algorithmic complexity and no FLOPs, but has a significant impact on real-world performance.
The exponential function in line 12 that recreates the complex number from its polar coordinates is a special case from a computational point of view. Its algorithmic and computational properties depend entirely on the exact parameters and are highly unpredictable [58].

Table 4.4: Numerical routines and FLOP estimates for the Y_{BUS} Newton-Raphson method.

Line	Operation	Numerical Interface	Numerical Routine	Total FLOPs
2	$\underline{\mathbf{Y}}\,\underline{\mathbf{U}}$	BLAS	`zgemv()`	$12n^2$
2	$()^*$	LAPACK	`zlacgv()`	n
3	$\underline{\mathbf{U}} \odot \underline{\mathbf{I}}$	MKL VM	`vzMul()`	$6n$
3	$\underline{\mathbf{U}}_{calc} - \underline{\mathbf{S}}$	MKL VM	`vzSub()`	$2n$
6	$\mathbf{E}\,\underline{\mathbf{I}}$	BLAS	`zgemv()`	$12n^2$
7	$\mathbf{E}\,\underline{\mathbf{U}}$	BLAS	`zgemv()`	$12n^2$
8	$\underline{\mathbf{Y}}\,\underline{\mathbf{U}}_M$	BLAS	`zgemm()`	$12n^3$
8	$()^*$	LAPACK	`zlacgv()`	n^2
8	$\underline{\mathbf{I}}_M - \underline{\star}$	MKL VM	`zomatadd()`	$2n^2$
8	$\underline{\mathbf{U}}_M\,\underline{\star}$	BLAS	`zgemm()`	$6n^3$
9	$\underline{\mathbf{Y}}^*$	LAPACK	`imatcopy()`	n^2
9	$\underline{\mathbf{U}}_M\,\underline{\mathbf{Y}}$	BLAS	`zgemv()`	$6n^3$
9	$\underline{\mathbf{I}}_M + \underline{\star}$	MKL VM	`zomatadd()`	$2n^2$
11	`solve(`$\star$`)`	LAPACK	`dgesv()`	$8/3n^3 + 7n^2 + 7/3n$
12	$\star$`exp(`$\star$`)`	MKL VM	`vzExp()`	–
Inner Loop:				$26\frac{2}{3}n^3 + 49n^2 + 11\frac{1}{3}n$

When applied to transmission grids, the Y_{BUS} Newton-Raphson method can be accelerated without a signifant loss of accuracy by neglecting the influence of active power on the absolute voltage and the influence of reactive power on the voltage angle difference. This simplification is approximately valid in grids which very long line lengths roughly in the range of more than 100 km. Looking at the formulation of the Jacobi matrix in equation 4.26, this means that J_{12} and J_{21} are set to 0. The original system of linear equation with $2n$ lines is thus split into two linear system of size n, which is a significant advantage. The model reduction and subsequent simplified solution is called *decoupled power flow*.

This approach can be further optimized by applying the Jacobi matrix computed in the first iteration for the subsequent iterations, thereby performing the computationally expensive Jacobi computation only once per load. This approach is called *Fast Decoupled Load Flow* [103].

There are also 'inexact' Newton-Raphson methods [34], [67], which sub-

stitute the exact computation of the Jacobi matrix for an iterative or approximative approach.

4.3 The Z_{BUS} Jacobi Method

The Z_{BUS} formulation and associated solution methods arise from a simple modification to the basic Y_{BUS}-equations. The definition of the admittance matrix (equation 3.8) is

$$\underline{\mathbf{I}} = \underline{\mathbf{Y}}\,\underline{\mathbf{U}}. \tag{4.45}$$

If the admittance matrix $\underline{\mathbf{Y}}$ is invertible, the voltage on the right side can be isolated by inverting it and multiplying it from the left:

$$\underline{\mathbf{Y}}^{-1}\underline{\mathbf{I}} = \underline{\mathbf{Y}}^{-1}\underline{\mathbf{Y}}\,\underline{\mathbf{U}} \tag{4.46}$$

$$\Rightarrow \underline{\mathbf{Y}}^{-1}\underline{\mathbf{I}} = \underline{\mathbf{U}}, \tag{4.47}$$

which would lead to the simple fixed point iteration scheme

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{Y}}^{-1}\left(\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}^{(m)}\right)^{*}. \tag{4.48}$$

This scheme looks very promising - the inverted admittance matrix needs to be computed only once for a grid, and all subsequent iterations can be carried out using only simple vector-matrix multiplications and vector operations.
However, the admittance matrix as shown in equation 4.45 is not generally invertible, as the relationship between $\underline{\mathbf{U}}$ and $\underline{\mathbf{I}}$ is not generally injective. The physical system that the equation describes is only well and unambiguously defined after a slack node has been included, which defines the voltage level that the grid is on. Both options for slack handling outlined in section 3.3 lead to an invertible matrix and to a correctly working Z_{BUS} Jacobi method in different ways. Option 1 involved a changed line and entry in $\underline{\mathbf{S}}$ for every slack node as shown again in equation 4.49 for a 3-node, single feeder grid:

$$\begin{bmatrix} (\underline{U}_0^*\underline{U}_0)/\underline{U}_0^* \\ \underline{S}_1^*/\underline{U}_1^* \\ \underline{S}_2^*/\underline{U}_2^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \underline{Y}_{01} & -(\underline{Y}_{01}+\underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix}. \tag{4.49}$$

An admittance matrix $\underline{\mathbf{Y}}_{\text{mod}}$ modified in this fashion is invertible, and equation 4.47 then leads to a simple fixed point iteration scheme:

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{Y}}_{\text{mod}}^{-1} \left(\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}^{(m)} \right)^* . \tag{4.50}$$

Using this method, additional complexities can be incorporated into the admittance matrix as normal.
The other option for slack handling outlined in section 3.3 requires the deletion of the line corresponding to one slack node:

$$\begin{bmatrix} \underline{S}_1^* / \underline{U}_1^* \\ \underline{S}_2^* / \underline{U}_2^* \end{bmatrix} = \begin{bmatrix} \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \end{bmatrix} . \tag{4.51}$$

The resulting admittance matrix in this case is not square and therefore not conventionally invertible by definition. However, the matrix can be squared using a simple transformation: If there are no shunt admittances in the grid, the power flows computed during an iteration are not dependent on the absolute voltage level, only on the voltage differences between the nodes. The voltage vector used for the computation of 4.45 can be shifted up or down by any value, and the equation stays correct. So, by subtracting $\underline{U}_0$ from $\underline{\mathbf{U}}$, equation can be formulated as

$$\begin{bmatrix} \underline{S}_1^* / \underline{U}_1^* \\ \underline{S}_2^* / \underline{U}_2^* \end{bmatrix} = \begin{bmatrix} \underline{Y}_{01} & -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_0 - \underline{U}_0 \\ \underline{U}_1 - \underline{U}_0 \\ \underline{U}_2 - \underline{U}_0 \end{bmatrix} . \tag{4.52}$$

The first column of $\underline{\mathbf{Y}}$ in equation 4.52 is now always multiplied by 0 and can therefore be deleted without losing any information, which leads to

$$\begin{bmatrix} \underline{S}_1^* / \underline{U}_1^* \\ \underline{S}_2^* / \underline{U}_2^* \end{bmatrix} = \begin{bmatrix} -(\underline{Y}_{01} + \underline{Y}_{12}) & \underline{Y}_{12} \\ \underline{Y}_{12} & -\underline{Y}_{12} \end{bmatrix} \begin{bmatrix} \underline{U}_1 - \underline{U}_0 \\ \underline{U}_2 - \underline{U}_0 \end{bmatrix} . \tag{4.53}$$

The resulting square admittance matrix $\underline{\mathbf{Y}}_{\text{red}}$ is conventionally invertible again. Equation 4.47 is now

$$\underline{\mathbf{Y}}_{\text{red}}^{-1} \underline{\mathbf{I}} = \underline{\mathbf{U}} - \underline{\mathbf{U}}_0, \tag{4.54}$$

where $\underline{\mathbf{U}}_0$ is a vector with $\underline{U}_0$ as every entry and the size of every vector is reduced by one. In practice, the slack voltage can be added or subtracted

to or from the vector $\underline{\mathbf{U}}$ as a scalar value via the BLAS Level 1 routine `zaxpy()`. This leads to the fixed point iteration scheme

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{Y}}_{\text{red}}^{-1}\,\underline{\mathbf{I}}^{(m)} + \underline{\mathbf{U}}_0 \tag{4.55}$$

$$\Rightarrow \underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{Y}}_{\text{red}}^{-1}\left(\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}^{(m)}\right)^* + \underline{\mathbf{U}}_0, \tag{4.56}$$

In the following, $\underline{\mathbf{U}}_0$ is more generally denoted as $\underline{\mathbf{U}}_{\text{slack}}$.
If there are shunt impedances in the grid, they can not be incorporated into the admittance matrix as outlined in section 4.6.4, as the admittance matrix is multiplied with the shifted voltage vector and not the real voltages occurring at the nodes. Instead, their effect must be integrated using *current compensation* by computing the current flowing through the shunt and adding it to the computed node current $\underline{\mathbf{I}}^{(m)}$ as in

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{Y}}_{\text{red}}^{-1}\left(\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}^{(m)} + \underline{\mathbf{Y}}_{\text{shunts}} \odot \underline{\mathbf{U}}^{(m)}\right)^* + \underline{\mathbf{U}}_{\text{slack}}. \tag{4.57}$$

Both variants of the $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi method lead to mathematically equivalent iterations. In the remainder of the thesis, the second option is used, as it is more prevalent in the existing literature.

For the Jacobi method, analogous to the $\mathrm{Y}_{\mathrm{BUS}}$ Jacobi method, the resulting algorithm 6 for the second option is very concise and contains only simple matrix and vector operations apart from the inversion of $\underline{\mathbf{Y}}$ at the very start. The given admittance matrix is in this case the reduced matrix $\underline{\mathbf{Y}}_{\text{red}}$.

Algorithm 6 $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi Method

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{U}_{\text{slack}}$, ϵ_S
1: $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$ ▷ precomputation
2: **loop**
3: $\quad\underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
4: $\quad$**if** $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **then**
5: $\quad\quad$**break loop**
6: $\quad\underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{\text{slack}}$
return $\underline{\mathbf{U}}$

The explicit matrix inversion in line 1 has to be carried out only once for every grid. The current is explicitly computed as an intermediate result

in line 3 because it is required for the convergence criterion in line 4 as well as the actual voltage update step that happens in line 6.
Like the Y_{BUS} Newton method, the Z_{BUS} Jacobi method algorithmically has 'global' iteration properties and solves the simple example problem in one step.

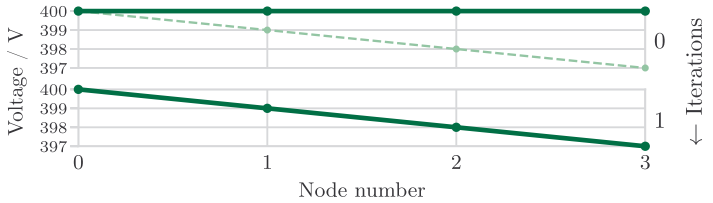


Figure 4.6: Voltage profiles of the first iteration of a Z_{BUS} Jacobi method for a single feeder grid, 4 nodes, 400 W of load applied to the node 3. The final solution is shown as the dashed green line.

Table 4.5: Numerical Routines and complexities for the Z_{BUS} Jacobi method. The number of FLOPs is given in a mathematical sense only.

Line	Operation	Numerical Interface	Numerical Routine	Total FLOPs Est.
1	$\underline{\mathbf{Y}}^{-1}$	LAPACK	`zgetrf(); zgetri()`	$6\frac{2}{3}n^3 - 5n^2 + 8\frac{1}{3}n$
3	$\underline{\mathbf{S}} \oslash \underline{\mathbf{U}}$	MKL VM	`vzDiv()`	$11n$
3	$()^*$	LAPACK	`zlacgv()`	n
6	$\underline{\mathbf{Z}}\,\underline{\mathbf{I}}$	LAPACK	`zgemv()`	$12n^2$
6	$\Delta\underline{\mathbf{U}} + \underline{U}_{\text{slack}}$	MKL VM	`zaxpy()`	n
Inner Loop:				$12n^2 + 13n$

Computationally, this method plays very well to the strengths of modern CPUs and its runtime is often very competitive. The numerical routines used and their complexities are outlined in table 4.5. The costly computation of the matrix inverse $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$ occurs only once for every unique grid. The memory footprint of the entire inner loop is $16\text{Bytes} \cdot (n^2 + 3n + 1.5)$. The inherent density of $\underline{\mathbf{Z}}$ renders sparse matrix techniques inefficient except for very large, radial grids. In the 60s and 70s, when most of the groundwork for power flow computations was laid, the storage demands for the full matrix $\underline{\mathbf{Z}}$ were a deal breaker, and the method was dismissed (e.g., [104]). With today's computers,

the storage demand is no longer a problem for all but massive grids, but the method still seems to be largely unused.
As the Z_{BUS} approach also leads to a fixed-point formulation, the Gauss-Seidel and Relaxation methods could also be applied here. However, due to the dense matrix $\underline{\mathbf{Z}}$ and the therefore inherently global iterations (all nodes are corrected in every iteration step), the main disadvantage of the Y_{BUS} Jacobi method does not apply here, and the Z_{BUS} Jacobi method is sufficient.

4.4 Backward/Forward Sweep Method

In very simple electrical grids with only one straight path (sometimes called *feeders*, although this term is sometimes used for all low-voltage grids, regardless of grid topology, in the following called *single feeders*), the application of Kirchhoffs Voltage Law (KVL) and Kirchhoffs Current Law (KCL) leads to an algorithm that uses only vector arithmetic and no matrix multiplication.
Figure 4.7 shows the equivalent electrical circuit of a single feeder grid with three nodes, with three closed loops in which, according to KVL, the voltages must add up to zero.
These loops give rise to the following equations:

$$\begin{aligned} &\text{Loop 1:} && \underline{U}_0 = \underline{U}_{01} + \underline{U}_1 \\ &\text{Loop 2:} && \underline{U}_0 = \underline{U}_{01} + \underline{U}_{12} + \underline{U}_2 \\ &\text{Loop 3:} && \underline{U}_0 = \underline{U}_{01} + \underline{U}_{12} + \underline{U}_{23} + \underline{U}_3. \end{aligned} \tag{4.58}$$

They can be reordered to explicitly describe the node voltages $\underline{U}_1$, $\underline{U}_2$, and $\underline{U}_3$:

$$\begin{aligned} \underline{U}_1 &= \underline{U}_0 - \underline{U}_{01} \\ \underline{U}_2 &= \underline{U}_0 - \underline{U}_{01} - \underline{U}_{12} \\ \underline{U}_3 &= \underline{U}_0 - \underline{U}_{01} - \underline{U}_{12} - \underline{U}_{23}. \end{aligned} \tag{4.59}$$

These equations can be expressed in a vectorized form by introducing the vector of line voltages

$$\underline{\mathbf{U}}_L = \begin{bmatrix} \underline{U}_{01} & \underline{U}_{12} & \underline{U}_{23} \end{bmatrix}^T, \tag{4.60}$$

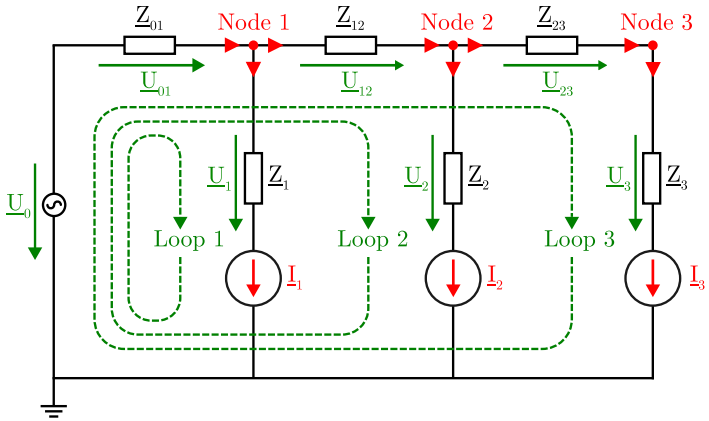


Figure 4.7: Feeder Grid with three consumer nodes. The dashed lines show three closed loops for Kirchhoffs Voltage Law, the red dots mark three nodes for the application of Kirchhoffs Current Law

another vector of the same dimension, which contains the transformer or slack voltage

$$\underline{\mathbf{U}}_0 = \begin{bmatrix} \underline{U}_0 & \underline{U}_0 & \underline{U}_0 \end{bmatrix}^T, \tag{4.61}$$

and the operation

$$\begin{aligned} \texttt{cumsum(X)} &= \begin{bmatrix} x_1 & x_1 + x_2 & x_1 + x_2 + x_3 & \dots \end{bmatrix}, \\ \text{where } \mathbf{X} &= \begin{bmatrix} x_1 & x_2 & x_3 & \dots \end{bmatrix}. \end{aligned} \tag{4.62}$$

The vectorized form of equations 4.59 is then

$$\underline{\mathbf{U}} = \underline{\mathbf{U}}_0 - \texttt{cumsum}(\underline{\mathbf{U}}_L). \tag{4.63}$$

The line voltage vector $\underline{\mathbf{U}}_L$, in this case the voltage drops $\underline{U}_{01}$, $\underline{U}_{12}$ and $\underline{U}_{23}$, can be expressed in terms of the line impedances and the line

currents by applying Ohm's Law:

$$\underline{U}_{01} = \underline{Z}_{01}\,\underline{I}_{01} \tag{4.64}$$
$$\underline{U}_{12} = \underline{Z}_{12}\,\underline{I}_{12} \tag{4.65}$$
$$\underline{U}_{23} = \underline{Z}_{23}\,\underline{I}_{23}, \tag{4.66}$$

or, in a vectorized form,

$$\underline{\mathbf{U}}_L = \underline{\mathbf{Z}}_L \odot \underline{\mathbf{I}}_L. \tag{4.67}$$

The line impedance vector $\underline{\mathbf{Z}}_L$ contains all the complex impedances in one vector:

$$\underline{\mathbf{Z}}_L = \begin{bmatrix} \underline{Z}_{01} & \underline{Z}_{12} & \underline{Z}_{23} \end{bmatrix}^T, \tag{4.68}$$

The line currents $\underline{\mathbf{I}}_L$ can, in the special case of a single feeder grid, easily be expressed in terms of the node currents $\underline{I}_0$, $\underline{I}_1$ and $\underline{I}_2$ by applying KCL:

$$\texttt{Node 3:} \quad \underline{I}_{23} = \underline{I}_3 \tag{4.69}$$
$$\texttt{Node 2:} \quad \underline{I}_{12} = \underline{I}_2 + \underline{I}_{23} = \underline{I}_2 + \underline{I}_3 \tag{4.70}$$
$$\texttt{Node 1:} \quad \underline{I}_{01} = \underline{I}_1 + \underline{I}_{12} = \underline{I}_1 + \underline{I}_2 + \underline{I}_3 \tag{4.71}$$

The line current vector

$$\underline{\mathbf{I}}_L = \begin{bmatrix} \underline{I}_{01} & \underline{I}_{12} & \underline{I}_{23} \end{bmatrix}^T \tag{4.72}$$

can therefore be expressed in terms of the node currents as

$$\underline{\mathbf{I}}_L = \texttt{reversed(cumsum(reversed(}\underline{\mathbf{I}}\texttt{)))}, \tag{4.73}$$

where the operation `reverse()` results in a new vector with inverted order.
Lastly, the node currents $\underline{\mathbf{I}}$ can be calculated using the complex power

given at every node:

$$\underline{I}_1 = \underline{S}_1^* / \underline{U}_1^* \tag{4.74}$$
$$\underline{I}_2 = \underline{S}_2^* / \underline{U}_2^* \tag{4.75}$$
$$\underline{I}_3 = \underline{S}_3^* / \underline{U}_3^*, \tag{4.76}$$

or, in a vectorized form, using the Hadamard division analogous to section 3.2:

$$\underline{\mathbf{I}} = \underline{\mathbf{S}}^* \oslash \underline{\mathbf{U}}^* \tag{4.77}$$

With this, the initial KVL equations in 4.58 have been transformed to express the node voltages $\underline{\mathbf{U}}$ in terms of known quantities: the transformer voltage $\underline{\mathbf{U}}_0$, the impedances $\underline{\mathbf{Z}}_L$, the consumed powers $\underline{\mathbf{S}}$, and an initial guess for the voltages $\underline{\mathbf{U}}^{(0)}$. The full implicit formulation is

$$\underline{\mathbf{U}}^{(m+1)} = \underline{\mathbf{U}}_0 - \texttt{cumsum}(\underline{\mathbf{Z}}_L \odot \texttt{reversed}(\texttt{cumsum}(\texttt{reversed}(\underline{\mathbf{S}}^* \oslash \underline{\mathbf{U}}^{(m)*})))) \tag{4.78}$$

The algorithm for the backwards/forwards sweep power flow (BFS) is outlined in algorithm 7.

Algorithm 7 Backward/Forward Sweep Power Flow for Feeders

Input: $\underline{\mathbf{Z}}_L$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{\mathbf{U}}_0$, ϵ_S

1: **loop**
2: $\quad\underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
3: $\quad$**if** $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty > \epsilon_S$ **then**
4: $\quad\quad$**break loop**
5: $\quad\underline{\mathbf{U}} = \underline{\mathbf{U}}_0 - \texttt{cumsum}(\underline{\mathbf{Z}}_L \odot \texttt{reversed}(\texttt{cumsum}(\texttt{reversed}(\underline{\mathbf{I}}))))$

return $\underline{\mathbf{U}}$

With the exception of the input variables, the algorithm looks similar to the other fixed point methods, especially the $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi method. In fact, the BFS algorithm can be seen as a special case of the $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi method. The only difference is the computation of the 'correction voltage' $\underline{\mathbf{U}} - \underline{\mathbf{U}}_0$, which involves a matrix-vector multiplication for the $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi method, and vector operation for BFS. The `cumsum()` and `reversed()` methods are available in most higher-level numeric li-

braries.
Notably, the BFS algorithm contains only vector operations and does not require any matrix-vector multiplications. It is supremely fast, but only applicable to single feeder grids with one straight line of lines and nodes.
Just like the Y_{BUS} Newton-Raphson and Z_{BUS} Jacobi methods, the Backward-Forward-Sweep method solves the simple example problem in Figure 4.1 in one step:

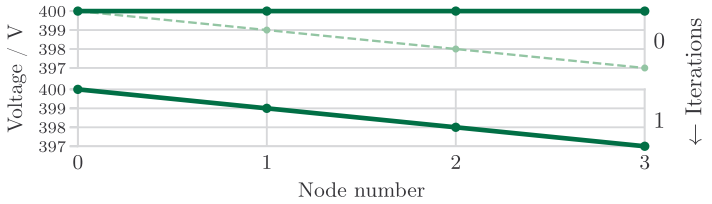


Figure 4.8: Voltage profiles of the first iteration of a Backward-Forward-Sweep method for a single feeder grid, 4 nodes, 400 W of load applied to the node 3. The final solution is shown as the dashed green line.

The memory requirements for this method are also very low, which could be an advantage on lower end or microprocessor-based hardware.
There have been attempts (e.g., [42] [37] [69] [90] [98]) to apply the Backward / Forward - Sweep method to other grid topologies by describing the current-to-voltage-relationship in a matrix with graph-theory tools, but these approaches necessarily lead to methods that are mathematically equivalent to the Z_{BUS} method.

4.5 Performance Characteristics

In order to get a general impression of the performance characteristics, this section features a detailed performance breakdown of one specific power flow problem solved by each presented method. The chosen problem consists of $10,000$ power flows with different, randomly chosen loads $\underline{\mathbf{S}}$ in a low-voltage, single feeder (non-meshed) grid with 20 nodes. This grid topology represents a worst-case scenario for the performance of power flow solution methods. The convergence criterion is the power residual with $\epsilon_S = 1\,\text{VA}$ for each method.
The metrics shown here are specifically chosen to demonstrate the

difference between the algorithmic efficiency and the computational performance of the methods, as introduced in section 2.2.

Iterations and Runtime

First, the average number of iterations per load for every method is plotted in Figure 4.9.

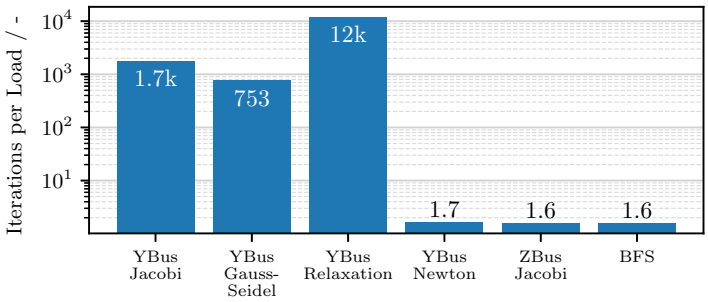


Figure 4.9: Average number of iterations per load for each method - lower is better

The number of iterations is purely a measure of the quality of the algorithmic efficiency, not of the computational performance.
In the case of the Y_{BUS} Relaxation method, the number is however slightly misleading, as only the voltage at one node is updated per iteration as opposed to all other methods, which can theoretically update all node voltages in every step. The differences are enormous, with the three Y_{BUS} fixed point methods requiring considerably more iterations to converge. Between the Y_{BUS} Jacobi method and the Y_{BUS} Gauss-Seidel method, the latter is more algorithmically effective and convergences in much fewer iterations.
The three other methods with their more global adjustments convergence in one or two iterations, which shows their algorithmic superiority.
Figure 4.10 shows the average time per iteration for each of the six methods. This is not the CPU time, but the 'wall time', the actual time elapsed.

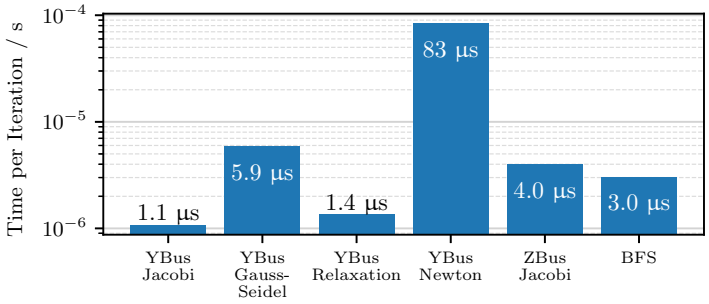


Figure 4.10: Average times for a single iteration for each method - lower is better

The Y_{BUS} Jacobi method is more than 5 times faster per iteration compared to the Y_{BUS} Gauss-Seidel method. This makes the Y_{BUS} Jacobi method faster overall.
The Y_{BUS} Relaxation method has a higher time per iteration than the Y_{BUS} Jacobi method and, as shown above, a higher iteration count. This makes the it uncompetetive in almost all situations, the only potential exception being optimization problems with tiny, localized changes in power between runs.
The Y_{BUS} Newton method has the highest runtime per iteration, due to the required construction of $\mathbf{J}$ and the solution of the system of a linear equation in every iteration.
For both the Z_{BUS} Jacobi and Y_{BUS} Jacobi methods, the dominating operation is a matrix-vector multiplication, the only difference being the sparsity of $\underline{\mathbf{Y}}$ compared to $\underline{\mathbf{Z}}$, which still yields a considerable advantage. Surprisingly, the advantage of the Backward-Forward-Sweep (BFS) method is only 25%, despite it featuring only vector-vector multiplications.
For the Z_{BUS} Jacobi method, the time shown includes the one-time matrix inversion of the admittance matrix, distributed onto the roughly 16,000 individual iterations. The inversion of the admittance matrix in this grid with 20 nodes takes around 17 µs on its own.
Figure 4.11 shows all total time durations for the solution of the 10,000 power flows. This time is the number of iteration in Figure 4.9, multiplied with the time per iteration in Figure 4.10 and 10,000, the number of loads.

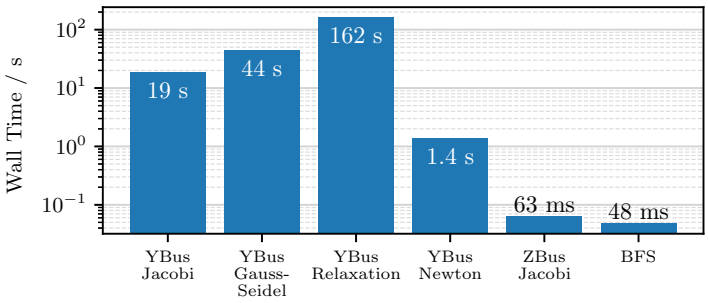


Figure 4.11: 'Wall Time' - real time the computer needed for the solution - lower is better

The total runtime clearly shows that the Y_{BUS} fixed point methods can not compete with the other methods. The Y_{BUS} Newton method is more than 10 times faster than the Y_{BUS} Jacobi method, and the two impedance-based methods are faster still by a factor of more than 20.
These huge performance differences diverge from the information in many publications, which often cite the Newton-Raphson as the fastest and most efficient method. [79], [20]. This at last shows where the findings about the computational performance of power flow computations from the 60s and 70s are severely outdated. The Z_{BUS} Jacobi method is not held back by its memory demands anymore and is - by these numbers - by far the most performant generally applicable power flow method.
The other metrics discussed in section 2.2 offer a more detailed view into the computational performance of the implementations.

L1 Cache Loads and Misses

Figure 4.12 shows the number of data loads from the L1 cache. This number is a measure of the memory accesses of the program.
The number of L1 cache loads is unsurprisingly roughly proportional to the runtime in Figure 4.11, no method shows a particular advantage here. Of these cache load requests, a certain percentage fail and are delegated to the other, slower caches and finally to memory. Figure 4.13 shows the percentage of L1 cache misses.
This percentage is a measure of the computational performance independent of the algorithmic complexity.

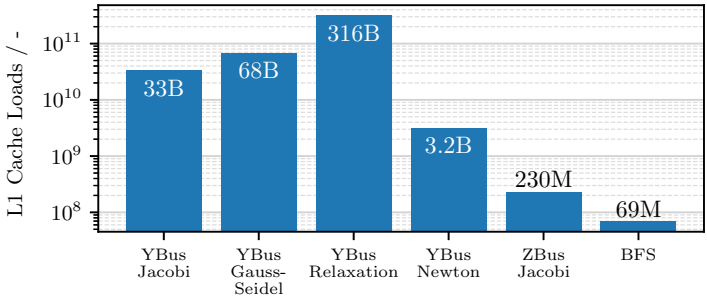


Figure 4.12: Number of L1 cache loads - lower is better

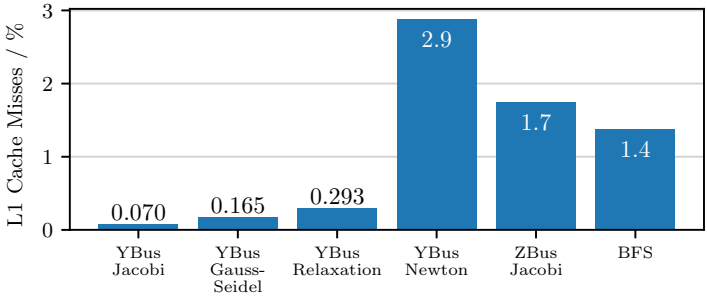


Figure 4.13: Percentage of L1 cache load misses - lower is better

The percentage of cache misses of the Y_{BUS} fixed-oint methods is a magnitude lower than of the other three methods. Part of the reason is that the Y_{BUS} fixed-point methods take much longer in total, which gives the prefetching algorithms inside the CPU more time and data to adapt to the memory access patterns. Another reason is the algorithmic simplicity, especially of the Y_{BUS} Jacobi method. On the other hand, the Y_{BUS} Newton-Raphson method features many variables and complex memory access patterns, and the result is a comparably high number of cache misses.
Although a high number of cache misses is detrimental for the computational performance, it also means that there is optimization potential in the implementation and the CPU itself. For example, the Y_{BUS} Newton-Raphson method can probably perform better on a CPU with more or faster cache. The low numbers of the Y_{BUS} fixed-point methods

show that this potential is largely already exploited.

Branch Misses

Another important metric for efficient programs is the number of branch misses, as introduced in section 2.2, shown in Figure 4.14. A branch miss can interrupt the instruction pipeline and stall the CPU. Again, the Y_{BUS} fixed point methods are very well suited for the branch prediction. The other three methods produce significantly more branch misses, which on the one hand is explained by the low number of iterations, but on the other hand promises some optimization potential.

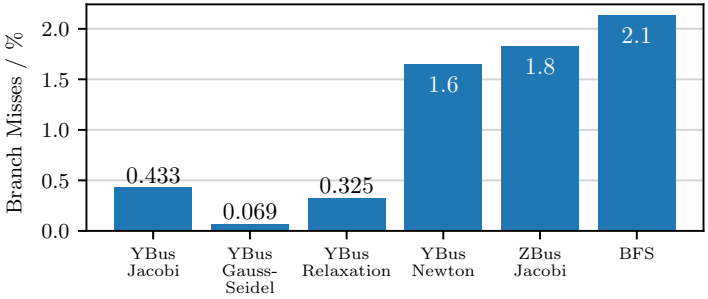


Figure 4.14: Percentage of branch misses for each power flow method - lower is better

Instructions per Cycle

Lastly, Figure 4.15 shows the number of executed instructions per CPU cycle, which shows the effects of µops-fusing on the one hand - allowing a combination of multiple instruction into one and therefore a number higher than one even without explicit parallelization - and CPU stalling on the other hand - leading to a underutilization of CPU resources. The Y_{BUS} Newton-Raphson method with its low computational performance stands out with a particularly low value, which is directly caused by the high number of cache misses as shown in Figure 4.13.

These more detailed measurement results allow for a deeper understanding of the wall time results in Figure 4.11. The Y_{BUS} fixed point methods are very well suited for modern CPU architectures, but their algorithms

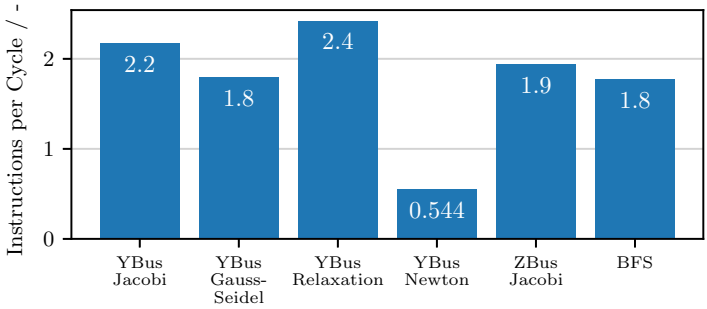


Figure 4.15: Average number of instructions executed per CPU cycle - higher is better

are too inefficient and require too many iterations to converge - this has not changed since the inception of the methods in the 60s.
The Z_{BUS} Jacobi method causes slightly more cache and branch misses, but the algorithm itself is so good in terms of number iterations required that it beats all other methods except for the highly specialized and limited BFS method. The memory requirements that led to unacceptable runtimes per iteration in the 60s and 70s are no problem for modern systems, and the memory access patterns are even more uniform than of the Y_{BUS} Newton-Raphson method, which makes the Z_{BUS} Jacobi method much faster.
The Y_{BUS} Newton-Raphson method has been the most popular power flow method since the 70s because of its efficient algorithm, not because it particularly well suited to modern CPUs. The solution of a system of linear equations in every iteration does not allow the CPU to perform to its maximum potential and frequently stalls it to load from cache or memory or revert to a branch after a misprediction.
The results also show the performance ceilings - what limits the performance - of every method. The Y_{BUS} fixed point methods are clearly CPU-bound and would only profit from a faster CPU clock. The Y_{BUS} Newton-Raphson method on the other hand would mainly profit from a bigger cache.
These performance measurements were conducted for a single scenario with a simple grid. Other grid sizes and topologies may lead to slightly different results, but the overall assessment is largely independent of grid shape except for extreme or very specific scenarios, e.g. 10,000s of nodes or close-to-blackout conditions. Chapter 6 contains three more

specific examples of more practical applications of these power flow solution methods.

4.6 Power Flow Complications

The definition of the Y_{BUS} formulation in section 3.1 and the explanations of the power flow solution methods in this chapter all assumed a fairly simplified electrical grid model: The loads were all elements with constant power, the grid was limited to one voltage level, was single-phase (symmetric), contained no shunts or transformer impedances, and there was only one slack node. In this section, the consequences of removing these model simplifications is discussed in terms of impact on the mathematical model, the implementation and the computational runtime.

4.6.1 Load Characteristics

The notion of a load in a power system is an abstraction. A load always represents an electrical network which consists of many electrical elements [51], which, until now, has been modeled as one electrical element that draws constant power $\underline{S}$ in this thesis. Specifically, the load has been assumed to be independent of voltage deviations or other environmental changes. Real electrical devices usually have different characteristics regarding their behavior under voltage changes. For example, a conventional heating element is better represented by a constant impedance, so the power drawn by the device is not constant with respect to the voltage, but is described by

$$\underline{S} = \underline{U}\,\underline{I}^* = \underline{U}\,\frac{\underline{U}^*}{\underline{Z}^*} = \frac{|\underline{U}|^2}{\underline{Z}^*}. \tag{4.79}$$

In this case, the load is quadratically dependent on the absolute voltage. Load can also resemble a constant current sink, in which case the power is linearly dependent on the voltage. Figure 4.16 shows how the resulting power of each of the three load models changes with the voltage.

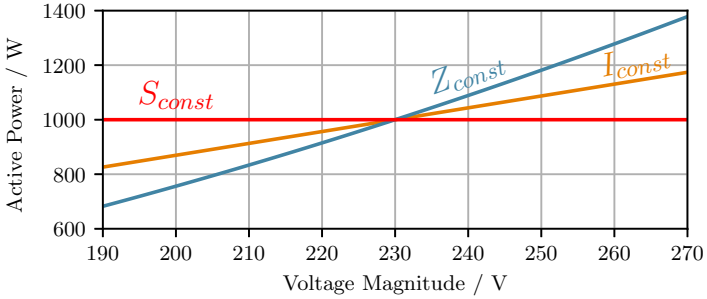


Figure 4.16: Resulting equivalent power for different load models under varying voltage in a purely active-power-case

Real household loads are usually composed of multiple devices that all represent one of the three load characteristics or a mixture between them. In medium or high voltage grids, a load consists of one or more entire grids and the households connected to them, including transformers.
For most practical problems, the exact load characteristics are nearly impossible to determine, as only power measurements, and not voltage or current, let alone impedance measurements are readily available. Luckily, their influence on the result is often negligible in steady-state simulations. CIGRE Working Group C4.605 performed a large-scale survey on the usage of load models by utilities all around the world and found that 84% of the participants use a constant power $\underline{S}$ as load model [18],[72] for static, steady-state simulations.
If the need arises, using another load model is straightforward in all power flow methods. In fact, using a pure constant current model severely simplifies and reduces them to direct, non-iterative equations. A pure constant impedance model requires the substitution of $\underline{\mathbf{I}} = (\underline{\mathbf{S}}/\underline{\mathbf{U}})^*$ with $\underline{\mathbf{I}} = \underline{\mathbf{U}}/\underline{\mathbf{z}}$.
When mixed load models are used, they are usually expressed in the *polynomial form*

$$\underline{S} = P_0 \left[a_1 \left(\frac{|\underline{U}|}{U_0} \right)^2 + a_2 \left(\frac{|\underline{U}|}{U_0} \right) + a_3 \right] + jQ_0 \left[a_4 \left(\frac{|\underline{U}|}{U_0} \right)^2 + a_5 \left(\frac{|\underline{U}|}{U_0} \right) + a_6 \right], \tag{4.80}$$

also called the *ZIP model*, or the *exponential form*

$$P = P_0 \left(\frac{|U|}{U_0} \right)^{n_P} \tag{4.81}$$

$$Q = Q_0 \left(\frac{|U|}{U_0} \right)^{n_Q} . \tag{4.82}$$

U_0 in this case signifies the voltage at which the nominal values P_0 and Q_0 are given. The coefficients a_1 - a_6 or n_P and n_Q need be measured in a laboratory environment, deduced by evaluating long-term measurements, or assumed.
Computationally, the integration of a mixed load model requires the computation of the load function in every iteration step.
The technique of adjusting the current in every iteration step to replicate the behavior of a component model can also be applied to slack nodes (see section 4.6.2), synchronous generators (see section 4.6.3), and more [12], [13].
If appropriate data is available, load models can be even more complex and take into account frequency dependencies or active control models. However, none of these modifications should influence the computational efficiency of the power flow methods beyond a constant overhead per iteration.

4.6.2 Multiple Slack Nodes

European low voltage grids usually have only one supplying transformer. Medium and high voltage grids as well as low voltage grids in other regions, however, can have multiple transformers connecting them to the overlaying grid or even different grids, and so the corresponding power flow model needs to accommodate multiple slack nodes. For transmission grids, this feature is unnecessary (see section 3.3). Therefore, it was not an issue in the early development of power flow algorithms and is not mentioned by many existing papers, books, or libraries.

Multiple Slack Nodes with Identical Voltage

If there are multiple slack nodes in the grid which all share the same voltage magnitude and angle, the equations can be trivially reduced to the one-slack-formulation by deleting all slack nodes except one and

connecting the now loose connections to that single slack. The remaining slack node can now be treated as a single slack node as explained in section 3.3.

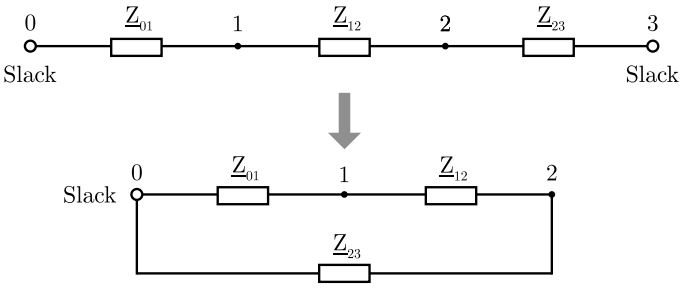


Figure 4.17: Reduction of multiple slack nodes with identical voltages

Multiple Slack Nodes with Different Voltage

If the slack nodes have different voltage magnitudes or angles, which is common when the supplying transformers have a means of voltage control like tap changers, the nodes and their corresponding lines in the equation have to be kept in place, while ensuring that the voltage never changes.
For the Y_{BUS} methods, the additional slack nodes can be integrated into the equations similarly to a single slack node if the first option for integrating the slack as explained in section 3.3 is chosen. Then, just like the first slack node, the lines corresponding to the other slack nodes in the admittance matrix have to be changed to 'unity' and the corresponding power S_i, which is not given in advance for a slack bus, to $U_i U_i^*$, which is. This modification ensures a constant voltage regardless of any load changes and thusly ensures the correct solution with any

Y_{BUS} method. For the simple grid in Figure 4.17, a simple feeder grid with 4 nodes and 3 lines, the power flow equation looks as follows:

$$\begin{bmatrix} \underline{U}_0\underline{U}_0^* \\ \underline{S}_1 \\ \underline{S}_2 \\ \underline{U}_3\underline{U}_3^* \end{bmatrix} = \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \\ \underline{U}_3 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 0 & 0 \\ \underline{Y}_{01} & -\underline{Y}_{01}-\underline{Y}_{12} & \underline{Y}_{12} & 0 \\ 0 & \underline{Y}_{12} & -\underline{Y}_{12}-\underline{Y}_{23} & \underline{Y}_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}^* \begin{bmatrix} \underline{U}_0 \\ \underline{U}_1 \\ \underline{U}_2 \\ \underline{U}_3 \end{bmatrix}^* . \tag{4.83}$$

The same does not work for the Z_{BUS} Jacobi method. The slack voltage is an inherent part of the algorithm, and all node voltages are computed relative to the single slack voltage. A direct integration of multiple slack nodes is therefore impossible. However, the method can be augmented to model the slack nodes as normal load nodes, and update the load at those nodes in every iteration to mimic the behavior of slack nodes. This general compensation approach can also be used to replicate other node behaviors (see section 4.6.3).

If there are two slack nodes, this involves a few more arithmetic operations before the voltage update step:

Let $i_{slack2} \in \mathbb{N}$ be the index of the additional slack node and $\underline{U}_{slack2} \in \mathbb{C}$ the desired, second slack voltage. Then $\underline{\mathbf{U}}[i_{slack2}]$ should converge towards $\underline{U}_{slack2}$, and the iterations should not conclude before this requirement is not fulfilled to at least some degree. This requires the definition of a secondary convergence criterion $\epsilon_{U,slack2} = \underline{\mathbf{U}}[i_{slack2}] - \underline{U}_{slack2}$. Before every voltage update step, the corresponding entry in the current vector $\underline{\mathbf{I}}[i_{slack2}]$ should then be modified so that the voltage update steps yields a voltage equal or at least as close as possible to $\underline{U}_{slack2}$ at position i_{slack2}. This is ensured by changing $\underline{\mathbf{I}}[i_{slack2}]$ so that

$$\underline{U}_{slack2} - \underline{U}_{slack} \overset{!}{=} \underline{\mathbf{Z}}[i_{slack2}, :] \cdot \underline{\mathbf{I}}. \tag{4.84}$$

Mathematically, this is solved by

$$\underline{\mathbf{I}}[i_{slack2}] = \frac{\underline{U}_{slack2} - \underline{U}_{slack} - \sum_{i \neq i_{slack2}} \underline{\mathbf{Z}}[i_{slack2}, i]\, \underline{\mathbf{I}}[i]}{\underline{\mathbf{Z}}[i_{slack2}, i_{slack2}]} \tag{4.85}$$

After that, the power vector $\underline{\mathbf{S}}$ should also be updated to ensure that the changes are carried over to the next iteration by setting

$$\underline{\mathbf{S}}[i_{slack2}] = \underline{\mathbf{U}}[i_{slack2}] \cdot \underline{\mathbf{I}}[i_{slack2}]^*. \tag{4.86}$$

Numerically, equation 4.85 is inelegant and requires a loop implementation because of the exception for $i \neq i_{slack2}$. If, however, $\underline{\mathbf{I}}[i_{slack2}]$ is set to 0 before this operation, the sum can be eliminated without changing the result of the expression:

$$\underline{\mathbf{I}}[i_{slack2}] = \frac{\underline{U}_{slack2} - \underline{U}_{slack} - \underline{\mathbf{Z}}[i_{slack2}, :] \cdot \underline{\mathbf{I}}}{\underline{\mathbf{Z}}[i_{slack2}, i_{slack2}]}. \tag{4.87}$$

This does no harm, because $\underline{\mathbf{I}}[i_{slack2}]$ is reassigned immediately anyway. $\underline{\mathbf{Z}}[i_{slack2}, :]$ could be assigned to its own variable, but this did not improve the runtime with the programming toolchain used in this thesis.
Algorithm 8 summarizes the steps for the $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi method with 2 slack nodes.

Algorithm 8 $\mathrm{Z}_{\mathrm{BUS}}$ Jacobi Method with 2 slack nodes

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{\mathbf{U}}_{slack}$, i_{slack2}, $\underline{U}_{slack2}$, ϵ_S, $\epsilon_{U,slack2}$

$\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$
$\underline{U}_{slack} = \underline{\mathbf{U}}_{slack}[0]$
loop
  $\underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
  $\underline{\mathbf{I}}[i_{slack2}] = 0$
  $\underline{\mathbf{I}}[i_{slack2}] = (\underline{U}_{slack2} - \underline{U}_{slack} - \underline{\mathbf{Z}}[i_{slack2}, :] \cdot \underline{\mathbf{I}}[:]) \,/ \underline{\mathbf{Z}}[i_{slack2}, i_{slack2}]$
  $\underline{\mathbf{S}}[i_{slack2}] = \underline{\mathbf{U}}[i_{slack2}]\, \underline{\mathbf{I}}[i_{slack2}]^*$
  if $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **and** $\underline{\mathbf{U}}[i_{slack2}] - \underline{U}_{slack2} < \epsilon_{U,slack2}$ **then**
    break loop
  $\underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{slack}$
return $\underline{\mathbf{U}}$

The operations in algorithm 8 are all vectorized and do not introduce many additional variables, so the runtime is not heavily impacted.
If the grid model has three or more slack nodes, the same solution principle applies. In this case, let m be the number of additional slack nodes, $\mathbf{i}_{slacks} \in \mathbb{N}^m$ a vector with the indices of the additional slack nodes, and $\underline{\mathbf{U}}_{slacks} \in \mathbb{C}^m$ a vector of the corresponding slack voltages. The computation of the current $\underline{\mathbf{I}}$ that yields the correct voltages at the slack nodes now requires the solution of a system of m linear equations. Specifically, equation 4.84 now involves m lines of $\underline{\mathbf{Z}}$.
The required operations could be expressed using an extended index notation, where $\underline{\mathbf{Z}}[\mathbf{i}_{slacks}, :]$ denotes a new matrix which contains only those lines of $\underline{\mathbf{Z}}$ of which the index is in $\mathbf{i}_{slacks}$. In numerical libraries, this is

called *fancy indexing* and is supported by Pythons Numpy and others. With this, the operation to find $\underline{\mathbf{I}}[\mathbf{i}_{slacks}]$ is

$$\underline{\mathbf{I}}[\mathbf{i}_{slacks}] = \texttt{solve}(\underline{\mathbf{Z}}[\mathbf{i}_{slacks}, \mathbf{i}_{slacks}], \underline{\mathbf{U}}_{slacks} - \underline{\mathbf{U}}_{slack}[\mathbf{i}_{slacks}]) \tag{4.88}$$

In practice, it is more efficient to reorder the nodes and put all slack nodes at the top than to use fancy indexing. Then, using the number of additional slack nodes m, equation 4.88 can be expressed as

$$\underline{\mathbf{I}}[:m] = \texttt{solve}(\underline{\mathbf{Z}}[:m, :m], \underline{\mathbf{U}}_{slacks} - \underline{\mathbf{U}}_{slack}[:m]), \tag{4.89}$$

where the notation $\underline{\mathbf{I}}[:m]$ denotes 'the first m elements of $\underline{\mathbf{I}}$'.
The complete algorithm for the Z_{BUS} Jacobi method with three or more slack nodes is outlined in 9. $\underline{\mathbf{Y}}$ and $\underline{\mathbf{S}}$ are assumed to be ordered in a way that all m slack nodes are at the top.

Algorithm 9 Z_{BUS} Jacobi Method with 3 or more slack nodes

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{\mathbf{U}}_{slack}$, m, $\underline{\mathbf{U}}_{slacks}$, ϵ_S, $\epsilon_{U,slacks}$

$\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$
loop
  $\underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
  $\underline{\mathbf{I}}[:m] = \texttt{solve}(\underline{\mathbf{Z}}[:m, :m], \underline{\mathbf{U}}_{slacks} - \underline{\mathbf{U}}_{slack}[:m])$
  $\underline{\mathbf{S}}[:m] = \underline{\mathbf{U}}[:m] \odot \underline{\mathbf{I}}[:m]^*$
  if $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **and** $\|\underline{\mathbf{U}}[:m] - \underline{\mathbf{U}}_{slacks}[:m]\|_{max} < \epsilon_{U,slack}$ **then**
    break loop
  $\underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{slack}$
return $\underline{\mathbf{U}}$

The Backward-Forward-Sweep method could, in principle, be augmented the same way, but possible applications are probably very rare.

4.6.3 PV Nodes

If a node in a grid is a synchronous generator, as most power stations effectively are from a grid standpoint, the representation as a node with fixed active and reactive power is not suitable and does not do the actual behavior justice. A real power station generator is controlled by a multitude of control loops and additional overlaid control by transmission grid

operators. A first-order approximation to this behavior is a node with a fixed active power and a fixed voltage magnitude, where the reactive power and the voltage angle are variables instead of the two components of the complex voltage. These nodes are commonly called *PV Nodes*, *generator nodes*, or *voltage-controlled nodes*. In the iterative procedure, this means that the magnitude of the voltage must be kept constant, while the reactive power can be varied to bring the equations into harmony. Like with slack nodes, this could be achieved by deleting the corresponding lines in the admittance matrix. This procedure is favorable for Y_{BUS} methods, well described in the papers that introduced the methods [115] [102] and particularly simple for Y_{BUS} Newton-Raphson method, because the complex power is being split up into real and imaginary parts already. However, for the Z_{BUS} Jacobi method this procedure is detrimental to performance, as it necessarily interrupts the main complex matrix multiplication. There is a better method, whose principle was already described in 1963 [12].
The basic idea is again to treat the node as a PQ node with constant power and variable voltage and compensate appropriately in a separate step. Algorithm 10 shows the complete steps to compensate for PV nodes in the Z_{BUS} Jacobi method.

Algorithm 10 Z_{BUS} Jacobi Method with PV nodes

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\mathbf{U}_{pv}$, n_{pv}, ϵ_S, $\epsilon_{U,pv}$

1: $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$
2: $\underline{\mathbf{Z}}_{pv,diag,inv} = \underline{\mathbf{1}} \oslash \mathtt{diag}(\underline{\mathbf{Z}})[:n_{pv}]$
3: **loop**
4: $\quad \underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
5: $\quad \underline{\mathbf{I}}[:n_{pv}] = \underline{\mathbf{I}}[:n_{pv}] + \underline{\mathbf{Z}}_{pv,diag,inv} \odot (\mathbf{U}_{pv} + \mathtt{exp}(j \cdot \mathtt{arg}(\underline{\mathbf{U}}[:n_{pv}])) - \underline{\mathbf{U}}[:n_{pv}])$
6: $\quad \mathrm{Im}(\underline{\mathbf{S}}[:n_{pv}]) = \mathrm{Im}(\underline{\mathbf{S}}[:n_{pv}] - \underline{\mathbf{U}}[:n_{pv}] \odot \underline{\mathbf{I}}[:n_{pv}]^*)$
7: $\quad$ **if** $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **and** $\|\underline{\mathbf{U}}[:n_{pv}] - \mathbf{U}_{pv}\|_{max} < \epsilon_{U,pv}$ **then**
8: $\quad\quad$ **break loop**
9: $\quad \underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{slack}$

return $\underline{\mathbf{U}}, \underline{\mathbf{S}}$

This algorithm is modified compared to algorithm 6 in lines 2, 5, and 6. Line 2 just precomputes a section of $\underline{\mathbf{Z}}$ for later use in every iteration. In line 5, the node currents are corrected to mimic the conditions if the

voltage magnitudes of the PV nodes were correctly fixed to the values given in $\mathbf{U}_{pv}$. To that end, the term $\mathbf{U}_{pv} + \texttt{exp}(j \cdot \texttt{arg}(\underline{\mathbf{U}}[: n_{pv}]))$ represents the voltages with corrected magnitude, which is subsequently used in the product $\underline{\mathbf{Z}}_{pv,diag,inv} \odot (\star - \underline{\mathbf{U}}[: n_{pv}])$ to calculate the compensation currents that temporarily correct the further computations which do not take the constant voltage magnitude of the generator nodes into account. In line 6, the reactive power of the PV nodes is updated to reflect the new state and to be able to return the reactive power on exit.
During the iterations, the reactive power is an additional state variable and ideally converges to an eventual steady state, just like the voltage itself. This means that an additional convergence criterion $\epsilon_{U,pv}$ is needed in line 7. Also, a set of convergence-enhancing measures like acceleration factors (see section 5.1) can be applied to the iteration of the reactive power only.
Real synchronous generators have reactive power limits, over or under which their operation becomes unstable. In reality, the generator control system will prevent this state by adjusting the output voltage by means of the excitation system. This behavior could be modeled in addition, either in every iteration step or after convergence. A frequently recommended method is to convert the node to a PQ node with a reactive power at the edge of the stable operation.

4.6.4 Shunt Elements

Impedances connecting a node directly to ground are called *parallel* or *shunt impedances*. In a grid model, these impedances can appear for several reasons.

1. They can be part of a π-line-model which represents a line segment as one series impedance and two parallel impedances at the ends.

2. They can be discrete elements that are installed in the grid. In practice, shunt inductors and capacitors are used to compensate the voltage drop caused by long capacitive cables or inductive overhead lines, respectively.

3. They can be a part of a (industrial) customer installation, where shunt capacitors are often used to offset the lagging current caused by inductive loads and thereby increase the power factor. In this case, they are usually part of the customer load $\underline{S}$, but could also

be an explicit part of the grid model.

The integration of shunt capacitors or inductors is straightforward and does not affect the computation characteristics. Of course, shunt elements could render an otherwise stable power flow instance unstable and influence the solution time that way.

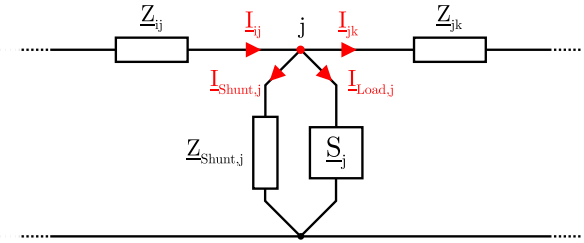


Figure 4.18: Equivalent electrical circuit with a shunt admittance at node j

Figure 4.18 shows a node j in an electrical network which has a load $\underline{S}_j$ and a parallel shunt impedance $\underline{Z}_{Shunt,j}$. The current through the shunt impedance is

$$\underline{I}_{Shunt,j} = \frac{\underline{U}_j}{\underline{Z}_{Shunt,j}}. \tag{4.90}$$

Using the admittance $\underline{Y}_{Shunt,j} = \frac{1}{\underline{Z}_{Shunt,j}}$ of the shunt, this fits into the standard admittance matrix $\underline{\mathbf{Y}}$ without any further modifications. The shunt admittance at the node is added to the corresponding entry of the diagonal of $\underline{\mathbf{Y}}$, so that the line j reads, in part,

$$\begin{bmatrix} \vdots \\ \underline{I}_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \\ \underline{Y}_{i,j} & \left(\underline{Y}_{Shunt,j} - \underline{Y}_{i,j} - \underline{Y}_{j,k}\right) & \underline{Y}_{j,k} \\ & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ \underline{U}_i \\ \underline{U}_j \\ \underline{U}_k \\ \vdots \end{bmatrix} \tag{4.91}$$

From, there, the computations in any of the $\mathrm{Y_{BUS}}$ methods or the $\mathrm{Z_{BUS}}$ Jacobi method continues as normal.

4.6.5 Asymmetry

As mentioned in section 3.1, the assumption of symmetry entails

1. the equal allocation of all loads onto the three phases,
2. identical line impedances in all phases, and
3. a neglect of the capacitances between the phases.

If one of those requirements can not be maintained, the grid model needs to be formulated as a three-phase model. This can complicate the power flow model and the subsequent solution considerably [1].
Problems with the first assumption, the equality of loads on the phases, most frequently arise in low-voltage grids, where individual devices are almost always connected to a single phase. In Germany, household and commercial buildings are usually wired to distribute the loads as balanced as possible, but sometimes, larger loads like electric vehicles or PV installations are connected to only one phase, which can skew the load distribution on the phases. This skew can only be simulated with a three-phase model [21]. In less densely populated areas around the world, a fully developed three-phase system is uneconomical, and so one-phase systems are common, sometimes mixed with three-phase systems in a single grid. A combined simulation including both system then also needs to consider an asymmetric grid.
Figure 4.19 shows a three-phase representation of a line and a connected node with three loads. The parallel line impedances $\underline{Z}_{j,UV}$, $\underline{Z}_{j,VW}$, and $\underline{Z}_{j,UW}$ can be used to model the capacitances between individual conductors in a cable or between separate cables or lines. Using the admittances, the relationship between the voltages, powers and impedances at the node j can be expressed in matrix form:

$$\begin{bmatrix} \underline{S}^*_{j,U}/\underline{U}^*_{j,U} \\ \underline{S}^*_{j,V}/\underline{U}^*_{j,V} \\ \underline{S}^*_{j,W}/\underline{U}^*_{j,W} \end{bmatrix} = \begin{bmatrix} \underline{Y}_{ij,U} & 0 & 0 & -\sum \underline{Y} & \underline{Y}_{j,UV} & \underline{Y}_{j,UW} & \underline{Y}_{jk,U} & 0 & 0 \\ 0 & \underline{Y}_{ij,V} & 0 & \underline{Y}_{j,UV} & -\sum \underline{Y} & \underline{Y}_{j,VW} & 0 & \underline{Y}_{jk,V} & 0 \\ 0 & 0 & \underline{Y}_{ij,W} & \underline{Y}_{j,UW} & \underline{Y}_{j,VW} & -\sum \underline{Y} & 0 & 0 & \underline{Y}_{jk,W} \end{bmatrix} \begin{bmatrix} \underline{U}_{i,U} \\ \underline{U}_{i,V} \\ \underline{U}_{i,W} \\ \underline{U}_{j,U} \\ \underline{U}_{j,V} \\ \underline{U}_{j,W} \\ \underline{U}_{k,U} \\ \underline{U}_{k,V} \\ \underline{U}_{k,W} \end{bmatrix} \tag{4.92}$$

The sum of the admittances connected to a node $\sum \underline{Y}$ is not written out here. It can include the shunt admittances of each phase as well.
The resulting admittance matrix is in principle built like the single-phase

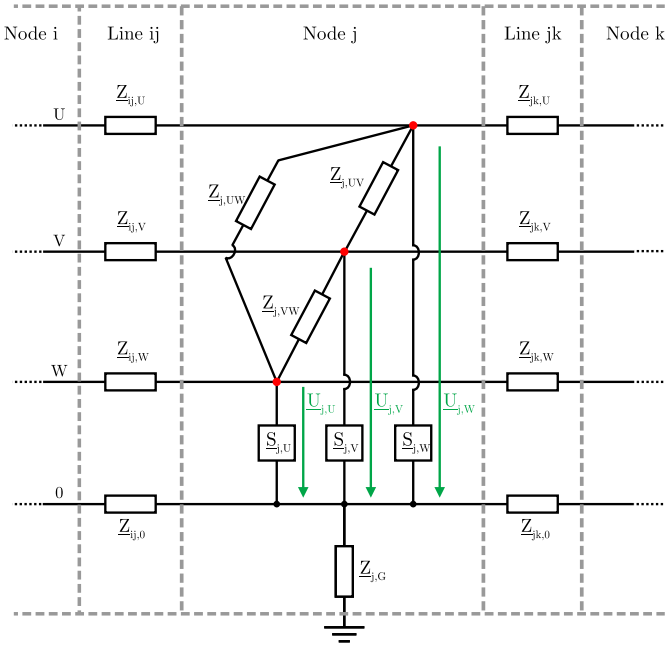


Figure 4.19: Three-phase representation of a node j and two connected lines ij and jk

admittance matrix in section 3.2, except that every entry is replaced with a 3×3 submatrix.

This augmentation takes care of the three-phase line model with only modifications to the admittance matrix and with no further requirements regarding the solution methods. However, with the assumption of symmetry retired, the nature and values of the grounding at the nodes becomes important. If the loads on the three phases are asymmetric, the resulting current in the neutral conductor (the *zero-sequence current*) is non-zero and flows back to the source (unless the neutral point is isolated), often both through the grid via the neutral line impedances represented by $\underline{Z}_{ij,0}$ and $\underline{Z}_{jk,0}$ and through the earth via the earthing resistance represented by $\underline{Z}_{j,G}$. Table 4.6 shows the values of the impedance between the neutral point of the node and the common ground of the

grid $\underline{Z}_G$ and the impedance between neutral points of connected nodes $\underline{Z}_0$ for different earthing strategies in the electric grid.

Table 4.6: Properties of the four dominant earthing strategies in electric grids

Earthing	$\mathbf{\underline{Z}_G}$	$\mathbf{\underline{Z}_0}$	**Application**
Direct / Solid Earthing	$\underline{Z}_G \approx 0$	$\underline{Z}_0 \approx 0$	HV & EHV grids, TN systems in LV grids
Low Impedance Earthing	$R_G > 0$	$\underline{Z}_0 \approx 0$	One point in MV cable grids
Peterson Coil	$X_G > 0$	$\underline{Z}_0 \approx 0$	One point in MV overhead line grids
Isolated Neutral	$\underline{Z}_G = \infty$	$\underline{Z}_0 = \infty$	Small area, high reliability grids, (IT system), delta-connected loads

A value of 0 for the impedance Z means it can not be directly translated to an admittance Y, in this case the line has to be ignored altogether and the two nodes connected by it have to be treated as equipotential. The case of direct earthing ($\underline{Z}_G \approx 0, \underline{Z}_0 \approx 0$) is the most frequent case in practice. In low voltage grids, the majority of grids in Europe uses a TN-S or TN-C-S earthing system, the difference being an additional path between neutral point and source through the earth for the TN-C-S system, which is shown in figure 4.20. The relevant impedance between the neutral points is therefore formed by the impedance of the *PEN* conductor and the impedance through earth in parallel.

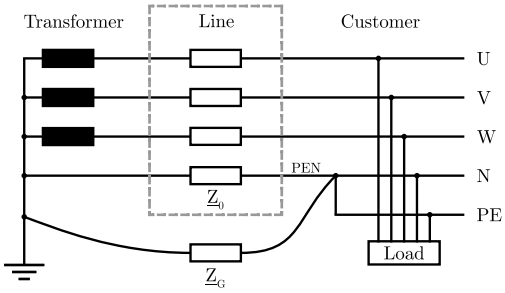


Figure 4.20: Schematic of a low-voltage grid using the TN-C-S system

In medium and high voltage grids, the asymmetry of the loads is usually small and the neutral current therefore negligible. Luckily, this case is also the easiest to integrate into the power flow model. The earthing doesn't need to be considered at all, and the three lines in equation 4.92 model the behavior of the node completely.
All other earthing variants lead to an additional line in the admittance matrix. A correct modeling of the earthing impedance in this case can be very complicated, because the return path may lead through earthing paths of other grids [114]. Also, the exact inductance value of Petersen coils may change during operation, as the inductance is kept in resonance with the line capacities.
The powers at the nodes are necessarily seperated onto the three phases, and so three-phase power data is required.
Finally, a three-phase model necessarily leads to three separate slack nodes. The voltages of the phases differ at least in their phase, but may also have different amplitudes. This means that the restrictions for grids with multiple slack nodes mentioned in section 4.6.2 all apply to three-phase grids as well.

4.6.6 Multiple Voltage Levels

The integration of multiple voltage levels into one power flow model can be achieved without much friction by referencing all physical values to a *base value*. The resulting quantities have no physical unit and are usually referred to as *per-unit-values*. The first step of a conversion to the per-unit system is the choice of a *base voltage* U_{Base}, which defines 1 *per unit*, or 1 p.u. The base voltage U_{base} is frequently chosen equal to

the slack voltage $\underline{U}_{slack}$, but this is not a mathematical requirement.
In order to coherently scale voltages, currents, powers, and impedances, one additional base value must be chosen, usually a base power S_{Base}. The choice of value for the second base value is completely arbitrary, but it makes sense to choose a roughly suitable power in the operating range as base power. With these, I_{Base} and Z_{Base} can be computed with

$$I_{\text{Base}} = \frac{S_{\text{Base}}}{\sqrt{3}\, U_{\text{Base}}} \tag{4.93}$$

$$Z_{\text{Base}} = \frac{U_{\text{Base}}}{\sqrt{3}\, I_{\text{Base}}} \tag{4.94}$$

Many papers and books use *pu*-values and therefore take the possibility of multiple voltages in a grid into account, although many practical power flow computations do not require it. The computation of the pu-values is an unnecessary step in these cases.
However, beyond a constant overhead for the scaling to *pu*-values and back, the performance of all power flow methods should be unaffected. Rounding errors due to overall lower values are not a problem if standard floating point types are used, which have a nonlinear distribution of possible values along their range.
When multiple voltage levels are involved in a grid model, the modeling of the transformer is usually the more complicated task.

4.6.7 Transformers

In many power flow simulations, the transformers are the boundaries of the grid model and simply modeled as slack nodes. This ignores the voltage drop that is caused by the impedances of the transformer itself. In many practical applications, this is not a problem, e.g. when the voltage profiles of different scenarios are compared. However, when the power flow model should emulate the real grid as closely as possible, the transformer can be included in the model. If the model spans multiple voltage levels, the integration of a transformer model is inevitable.
Including a transformer only affects the power flow model, not the solution methods themselves. Figure 4.21 shows a common transformer model [79], [81] which features six individual parameters that can be combined into three impedances. In the power flow model, the whole transformer model can be represented by 2 lines and 1 node.

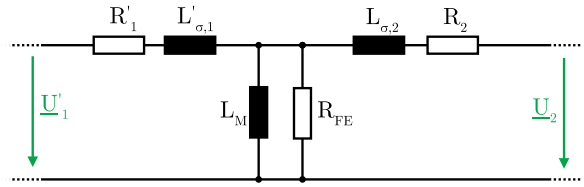


Figure 4.21: Equivalent circuit of a common transformer model

The physical values of one of the transformer coils need to be scaled by n^2 to use them in a grid model with a different operating voltage. In the example in Figure 4.21, the values U_1', R_1', and $L_{\sigma,1}'$ on the primary side are scaled from the original values.
The impedances of the lines and the shunt impedances are then

$$\underline{Z}_{T,1} = R_1' + 2\pi j f L_{\sigma,1}' \tag{4.95}$$

$$\underline{Z}_{T,2} = R_2 + 2\pi j f L_{\sigma,2} \tag{4.96}$$

$$\underline{Z}_{T,Shunt} = \frac{1}{\frac{1}{R_{FE}} + \frac{1}{2\pi f L_M}}. \tag{4.97}$$

In the node-based grid model, the transformer is then represented by the two lines and one node in Figure 4.22.

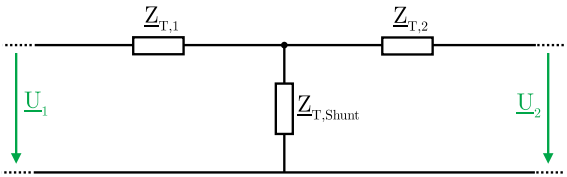


Figure 4.22: Equivalent circuit of a common transformer model

The shunt impedance can be integrated into the admittance matrix $\underline{\mathbf{Y}}$ as explained in section 4.6.4.
An even simpler transformer model is a single line with a combined impedance $\underline{Z}_T = \underline{Z}_{T,1} + \underline{Z}_{T,2}$, ignoring the shunt impedance. The model could also effecitvely be transformed into a pi-model by distributing $\underline{Z}_{T,Shunt}$ on the outer nodes.

The computational overhead caused by inclusion of a simple transformer model is minimal. However, some transformers can have additional features like tap-changers or variable phase-shifting capabilities [84], [32]. These can add even more complexity to the grid model, but the added model fidelity is rarely needed in steady-state applications, especially in distribution grids.

4.7 Handling of Convergence Problems

The term *convergence* has been introduced in section 3.4 and is part of every iterative power flow algorithm as the 'convergence criterion' which stops the iterations if the calculated $\underline{\mathbf{U}}^{(k)}$ solves the power flow equation 3.12 to a pre-chosen accuracy ϵ. The mathematically correct definition of convergence is however much more rigorous and very hard to apply to the power flow problem in a general way, for two reasons:
First, convergence is technically a property of an *infinite sequence*. The results of iterations like the ones during a power flow solution form a sequence, and if that sequence gets arbitrarily close to a certain value as the iterations go on infinitely, the sequence is called *convergent*. If the sequence of iterations results grows towards ∞ or $-\infty$ or reaches an undamped oscillation, the sequence is called *divergent*. The requirement of these properties for *infinite* lengths of the sequence and *arbitrary* closeness makes these conditions impossible to verify using numerical tools.
Second, since the convergence of a power flow algorithm applies to a sequence of vectors (e.g. $\underline{\mathbf{S}}_R^{(k)} \in \mathbb{C}^n$), the aforementioned 'closeness' is usually evaluated using a vector norm (e.g., the maximum norm $\|\cdot\|_\infty$). The associated convergence is called *norm convergence*, as opposed to the more mathematically strict *pointwise convergence*. This makes the convergence behavior of power flow algorithms even harder to reason about in a general, mathematically rigorous way for both fixed-point and root-finding methods.
Luckily, in real, practical applications, a mathematically rigorous check for convergence is almost never required. The pragmatic convergence criteria listed in section 3.4 are widely accepted to indicate a 'correct' solution. In much of the available literature on power flow, a power flow instance with a satisfied convergence criterion is colloquially denoted as convergent, and a power flow solution sequence that does not satisfy the convergence criterion after a certain, freely chosen number of iterations

as divergent.
An instance of a power flow computation can, in this colloquial sense, diverge for three reasons:
First, if the initial solution $\underline{\mathbf{U}}^{(0)}$ is chosen badly, an existing solution might not be found using the standard procedure of iterations. This indicates that the power flow function is not generally convex and that a *region of attraction* around the eventual solution exists. There exist numerous theoretical examinations of this phenomenon [123] [76], but in practice it is often sufficient to employ an additional *initialization step* to find a new, better initial value $\underline{\mathbf{U}}^{(0)}$ for the actual power flow iterations. Probably the most popular method for the initialization is *DC power flow*, which reduces the power flow equations to the relation between voltage angle and active power flows [105]. This model reduction implies the assumption of dominating line reactances over resistances, which is usually true for transmission grids, but does not hold for lower voltage distribution grids. For distribution grids with a non-negligible resistance, another simple possibility to reach the region of attraction is to reduce the load on the system by a certain factor and bringing the factor back to 1 over the course of the iterations [119].
The second reason for divergence during a power flow computation is the non-existence of a solution. Physically, this means that the instance represents a case where the voltage collapses, leading to a local blackout. There is of course nothing to be done to 'help' convergence in this case. It is generally desirable to recognize a case of voltage collapse as early as possible, which can be achieved with additional 'divergence criteria', such as $\mathtt{min}(|\underline{\mathbf{U}}^{(k)}|) < 0$. Iwamoto [55] presents a method to use the line-search parameter of the Newton-Raphson method (see Subsection 5.1.2) as an indicator for divergence. However, an additional criterion introduces a constant overhead and an additional branch into the program. In practice, a maximum number of iterations after which the iterations are stopped, and the solution is flagged for further processing, is a sufficient solution and implemented by most available power flow packages.
Finally, the third reason for non-convergence is numerical problems internal to the method. As with all numerical computations, the discrete nature of the floating-point representation inside the computer almost certainly leads to minor inaccuracies. These inaccuracies can be greatly amplified by numerically *ill-conditioned* operations during the computation. While every mathematical operation has a numerical conditioning associated to it, the operation where it is most likely a problem is the solution of a linear system $\mathbf{Ax} = \mathbf{b}$, which occurs during every itera-

tion step of the Newton-Raphson method. In this case, the *condition number* κ of the Jacobi matrix is a measure for the amplification of rounding errors. The potential error introduced by ill-conditioned operations is related to the rounding that actually takes place, which is in turn determined by the floating-point accuracy of the computation. The first power flow computations in the sixties were conducted using 10 bit floating-point numbers [6], which provided roughly two to three significant digits. With condition numbers greater than around $\kappa = 1000$, which can occur in admittance matrices derived from real grids with both large and small impedances, the inaccuracies introduced by this rounding could indeed lead to problems with convergence. Modern software uses at least 32 bit floating-point numbers by default, which have around 8 significant digits and can use 64 bit numbers as an option, which have around 15 significant digits. A lack of floating-point precision was for example identified as the cause for non-convergence of the IEEE 43 bus test system [55]. Using 64 bit floats, the solution is found without problems. With these modern floating-point formats, problems with numerical condition can only occur with Jacobi matrix condition numbers in the range of 10^8 and 10^{15}, respectively. These is only the case in huge grids or grid situations with extreme differences in impedances or wildly different voltages in the same grid, which can occur in situations close to voltage collapse.

In 2001, Wang et al. [119] investigated five instances of problems with the numerical condition reported by other papers. Two of the cases converge without any problems, suggesting that the original authors had errors in their programs, another case has a grave data parsing error, and one case represents a completely overloaded system which has no solution. The remaining case, from [113], is found to develop a Jacobian with a high condition number of the course of the iterations and, in fact, fails to converge. However, the chosen load in the system is extremely close to the point of overloading (99.8%) and represents a case right on the edge of stable operation where the Jacobi matrix becomes close to singular. In such a case, it is not clear whether the numerical effect of the condition has an impact on the convergence or divergence behavior at all.

Modern CPU hardware has 64 bit floating-point units as standard, as well as BLAS and LAPACK packages with built-in condition control including automatic pre-conditioning, partial pivoting as well as corrections for numerical stability. Using these tools, internal numerical problems can almost always be avoided without explicit changes to the Y_{BUS}

Newton-Raphson method. Power flow methods which do not employ the solution of a system of linear equations, like all the other methods outlined in this thesis, are not impacted by problems with numerical condition at all.
In practice, the optimal design of methods for convergence checking and control is a complex problem which depends greatly on the nature of the power flow problem at hand. If no specific issue about the problem is known, the best approach from a performance point of view is to just count the iterations and break out of the function after an arbitrarily preset number of maximum iterations is reached. Problems which are flagged in this way can then be further processed using different approaches to attempt to find a solution if one exists. When the power flow is running, any further checks would exponentially increase the chance of cache misses, especially in the TLB, branch misprediction and other penalties for CPU performance outlined in chapter 2.

Chapter 5

Computational Optimization Approaches

After the basic power flow solution methods have been covered in the previous chapter, this chapter introduces several modifications to the methods that promise further performance optimization. In general, these techniques can be classified into *lossless* and *lossy* optimizations:

- **Lossless Optimizations:**
  These are modifications to the model or the algorithms that aim to improve the runtime without a loss in precision of the solutions compared to the non-modified algorithms. However, they might also make the solution slower for some power flow tasks, if the problem is not well suited to the technique. Specifically, the presented approaches are:
  - Acceleration Factors and Line Search (Section 5.1),
  - Sparse Matrices (Section 5.2),
  - Lossless Grid Reduction (Section 5.3.1),
  - Parallelization (Section 5.5).
- **Lossy Optimizations:**
  These are modifications or complete replacements of algorithms that aim to trade an - hopefully acceptable - amount of precision for improvements in runtime. These should only be applied when

a confident estimation for the required precision is available. The presented lossy approaches are:

- Lossy Grid Reduction (Section 5.3.2),
- Weak Load Detection (Section 5.4).

A factor to consider in advance for every optimization approach is the constant overhead associated with many of the optimization approaches: if the problem to be solved involves a grid with few nodes and only a few separate load scenarios, an extensive analysis of the optimization potential might not be worth it.
However, the potential of these optimization can be substantial if the problem has to be solved continuously as part of online system, especially on lower-end hardware, e.g. on an industrial computer in a substation.

5.1 Acceleration Factors

5.1.1 Acceleration of the $\mathbf{Y_{BUS}}$ Gauss-Seidel method

The concept of acceleration factors was applied to the Y_{BUS} Gauss-Seidel method since its inception [39] [14], as it is a popular optimization approach for Gauss-Seidel methods in general. It was found that the rate of convergence can be increased by augmenting the Y_{BUS} Gauss-Seidel iteration rule 4.13 with a factor $\alpha \in \mathbb{R}, \alpha > 1$ as follows:

$$\underline{U}_i^{(m+1)} = \underline{U}_i^{(m)} + \frac{\alpha}{\underline{Y}_{ii}} \left(\frac{\underline{S}_i^*}{\underline{U}_i^{(m)*}} - \sum_{j=1}^{i-1} \underline{Y}_{ij} \underline{U}_j^{(m+1)} - \sum_{j=i}^{n} \underline{Y}_{ij} \underline{U}_j^{(m)} \right), \tag{5.1}$$

$$\forall\ i = 1 \dots n.$$

This is analog to an extrapolation of each voltage update step by a constant factor. Figure 5.1 shows the potential reduction in iteration count for a simple power flow problem with four single-feeder grids with 5, 10, 20, and 30 nodes. This is a worst-case scenario for Y_{BUS} fixed-point methods, as their convergence speed is related to the diagonal dominance of the matrix $\underline{\mathbf{Y}}$ [79] [63], which is worst for single-feeder grids, as they have the lowest possible number of entries in $\underline{\mathbf{Y}}$. The minimal number of iterations for each node count is indicated with a larger circle.

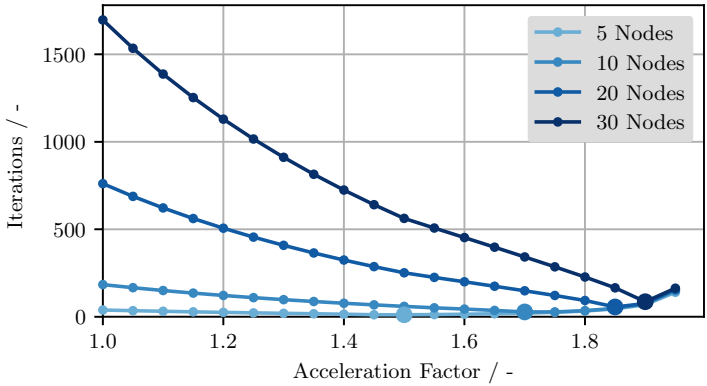


Figure 5.1: Number of iterations for different acceleration factors α in a single-feeder grid with varying number of nodes

The potential for optimization is profound: In the 30-node single-feeder grid, the necessary number of iterations drops by a factor of 21.4, which directly translates to an equivalent speedup, as shown in figure 5.2. This is however a best-case scenario, the improvement is usually smaller in more complex grid topologies.

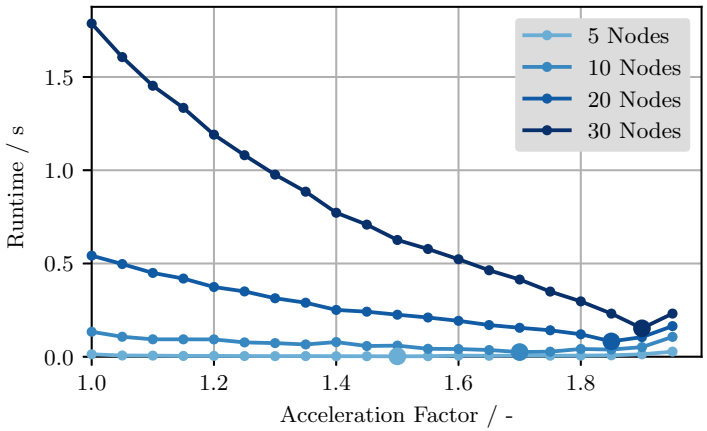


Figure 5.2: Runtime in seconds for different acceleration factors α in a feeder grid with varying number of nodes

The correct choice of acceleration factor therefore can have a significant impact on the runtime, however, it is not obvious how such a choice should be made. If α is too large (in the example case, $\alpha > 1.9$), the iterations can overreach and start to oscillate, leading to a sharp increase in runtime or no convergence at all. A conservative approach is a safe, constant factor α, a frequently cited choice is $\alpha = 1.6$ [39] [14], although this can be too great for larger, more complex grids (see section 6.3).
A slightly different way to accelerate the Gauss-Seidel method was presented by J.Treece in 1969 [110]. Its core idea is the "boosting" of the power $\underline{\mathbf{S}}$, controlled by the power mismatch at every iteration step by replacing $\underline{\mathbf{S}}$ with

$$\underline{\mathbf{S}}^{(m)}_{\text{Boost}} = \underline{\mathbf{S}} + B\,\underline{\mathbf{S}}^{(m)}_{\text{R}}, \tag{5.2}$$

where $B \in \mathbb{R}$ is a predetermined scalar factor and $\underline{\mathbf{S}}_{\text{R}}$ is the difference between the given powers at the nodes and the powers that would result from the voltages in the current iteration

$$\underline{\mathbf{S}}^{(m)}_{\text{R}} = \underline{\mathbf{S}} - \underline{\mathbf{U}}^{(m)} \odot \underline{\mathbf{Y}}\,\underline{\mathbf{U}}^{(m)}. \tag{5.3}$$

This term is also used for convergence control when ϵ_S is used as the convergence criterion. The discussion on Treece's paper [64] - a series of comments from peers was printed together with every IEEE Transactions journal paper at that time - highlights some of the problems with the method. The choice of boost factor is very sensitive to the exact power flow problem and can lead to non-convergence very easily. The boost method works best in large, weakly loaded grids (> 500 nodes), a class of problems that is better served with a Y_{BUS} Newton-Raphson or Z_{BUS} Jacobi method.
The accelerated Gauss-Seidel method is also somewhat similar to the general *Successive Over-Relaxation* (SOR)-method [15], but the additional factor $(1 - \alpha)$ on the value $\underline{U}_i^{(m)}$ is missing.

5.1.2 Acceleration of the $\mathbf{Y_{BUS}}$ Newton-Raphson method

As the Newton-Raphson method is a popular general-purpose optimization method, there exist numerous general approaches to accelerate its convergence. The equivalent of acceleration factors for the Newton-Raphson method is *line search*. The rationale here is that, while the step *direction* found by an iteration of the Newton-Raphson method cannot

easily be improved, the step *size* can. This might lead to fewer iteration steps, but comes at the cost of a constant overhead per iteration.
In the Newton-Raphson method (algorithm 4), the optimization procedure for α has to occur before the final voltage update in line 6, which, including the acceleration factor, then reads

$$\underline{\mathbf{U}} = (|\mathbf{U}| + \alpha\Delta|\mathbf{U}|) \cdot e^{j(\phi+\alpha\Delta\phi)}. \tag{5.4}$$

The search for the optimal α is more complicated than for the $\mathrm{Y_{BUS}}$ Gauss-Seidel method and involves a separate optimization procedure for every iteration.
In practical power flow applications, the potential for runtime improvements using line search approaches is limited. The fundamental problem of line search for the $\mathrm{Y_{BUS}}$ Newton-Raphson method is that the method usually already needs very few iterations in the first place. As section 4.5 showed, the method converges in one or two steps even for non-trivial problems. The line search procedure would have to cut at least one iteration from that in order to provide a performance boost. This is only realistically possible for huge grid models.
As an example for such a huge model, Figure 5.3 shows the optimal α for every iteration during the solution of a power flow problem in a generic, weakly meshed medium voltage grid with 3000 nodes (see Appendix C.2 for more details). The solution with the $\mathrm{Y_{BUS}}$ Newton-Raphson method needs 7 iterations, but a line-search procedure does not accelerate it even in this case.

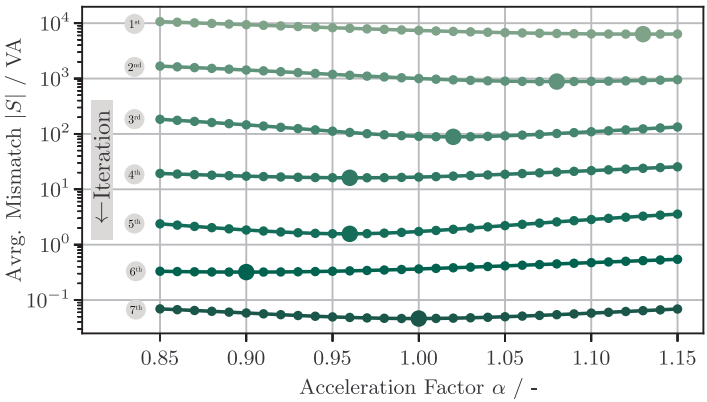


Figure 5.3: Potential of line-search optimization for the Newton-Raphson method for a generic, weakly meshed medium voltage grid with 3000 nodes

Every marker in this plot represents the average absolute power mismatch after an iteration using the acceleration factor α on the x-axis, and markers belonging to an iteration are grouped in a line. A non-accelerated Y_{BUS} Newton-Raphson method is on this plot represented by a progression on the vertical line at $\alpha = 1.00$ down towards an absolute power mismatch lower than the converge criterion $\epsilon_S = 0.1$VA.
The line search further optimizes every iteration step, leading to a slightly lower mismatch compared to the default choice $\alpha = 1$ (the lowest mismatch features a bigger marker). This procedure is only worth it if one iteration step can ultimately be skipped, and the power flow converges in fewer iterations. This is not the case even in the 3000-node computation in Figure 5.3.
However, although it does not help with performance, the line search procedure can be used to strengthen the robustness of the Y_{BUS} Newton-Raphson method [77] [94]. Iwamoto [55] used line search to establish a criterion for divergence.

5.2 Exploiting Sparsity

In theory, the maximum number of lines in an electric network with n nodes is $(n^2 - n)/2$. In reality, electric grids have considerably fewer lines and so the admittance matrix $\underline{\mathbf{Y}}$ contains a considerable amount of zeros. From very early on in the history of power flow computations,

researchers tried to exploit this fact by employing *sparse matrix storage schemes* [95]. Because the computers at the time had slow and very limited memory, this enabled faster computations, especially when applied to larger grid models.

The basic idea is intuitive: Store only the non-zero values plus information about their location in the matrix. Alongside the reduced storage space, sparse storage techniques can also lead to better performance of matrix operations, because the number of individual, scalar operations can be greatly reduced. However, these operations are usually more erratic from the CPUs point of view, and many of the automatic computational optimizations described in section 2.2 can not be applied as effectively. For computations involving very large sparse matrices (say, $n > 10{,}000$), sparse matrix techniques are essential, but for very small matrices (say, $n < 10$), they almost always lead to worse performance. Somewhere in between, there is a break-even point for sparse methods, where the reduced amount of individual operations makes up for the additional overhead and the less ideal use of the CPU pipelines and caches. For the computers used by the researchers who first explored sparse techniques in the 70s, this break-even point was probably very low, so that they experienced great speedups and recommended sparse matrix techniques for all power flow computations. Today, this is not true anymore, the efficiency of modern CPUs in uniform, predictable operations like matrix multiplication has pushed the break-even point considerably upwards.

There are many different sparse matrix storage schemes [86] [41]. The tested schemes were *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC), and *Coordinate List* (COO), all of which performed very similarly.

In the presented power flow algorithms, the only two matrix operations are matrix multiplication and the solution of a linear system in the Y_{BUS} Newton-Raphson algorithm. Even if these operations profit from a sparse matrix formulation, that does not automatically mean a performance boost for the entire algorithm. For smaller grids, the conversion overhead can be too high, and the efficiency of modern libraries and CPUs with uniform vector and matrix computations can ultimately mean a significantly better performance using the dense formulations. Where exactly the break-even point of sparse formulations for power flow computations lies, depends on the algorithm and the exact problem. The following benchmarks for the Z_{BUS} Jacobi method and the Y_{BUS}

Newton-Raphson method aim to give some indication.

5.2.1 Sparse Z_{BUS} Jacobi Method

The Z_{BUS} algorithm contains only one matrix multiplication, $\underline{\mathbf{Z}}\,\underline{\mathbf{I}}$, which is the computational centerpiece of the entire algorithm. Because $\underline{\mathbf{Z}}$ is constant between iterations and even between different invocations of the algorithm with different powers $\underline{\mathbf{S}}$, the conversion to a sparse formulation occurs only once and poses no performance problem. However, the inverted admittance matrix $\underline{\mathbf{Z}}$ is usually not very sparse. Single feeder or highly meshed grids can even lead to a completely filled $\underline{\mathbf{Z}}$, so that a sparse formulation makes little sense.
For a radial low voltage grid, Figure 5.4 shows the runtime of four different implementations of Z_{BUS} Jacobi: a standard, non-sparse version, and implementations using the CSR, CSC, and COO sparse matrix formulations for $\underline{\mathbf{Z}}$, respectively. The grid has a varying number of nodes, the loads were a set of randomly chosen powers from a standard beta function for households. The runtime is the total computation time for 35040 individual power flow problems.

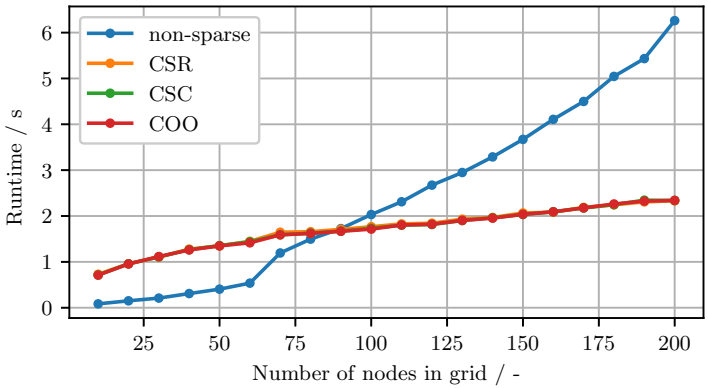


Figure 5.4: Runtimes of Z_{BUS} methods with different sparse matrix storage schemes in a radial grid with 35040 load cases. The CSR, CSC, and COO runtimes are nearly identical

For this best-case scenario, the results show that the non-sparse version is significantly faster up to around 80 nodes, at which point the advan-

tages of the sparse formulation overcome the raw optimization of the non-sparse version. This threshold is however dependent on the exact grid topology and is usually higher for more complicated meshed grids. The different formulations for sparse matrices differ very little from each other.

5.2.2 Sparse $\mathbf{Y_{BUS}}$ Newton-Raphson Method

For the Y_{BUS} Newton-Raphson method, the break-even point for sparse matrix storage methods occurs at even larger grid sizes. Two matrices inside the Newton-Raphson algorithm can be represented as sparse matrices: the admittance matrix $\underline{\mathbf{Y}}$ and the Jacobi matrix $\mathbf{J}$. $\underline{\mathbf{Y}}$ is used in the matrix multiplication $\mathbf{E}\,\underline{\mathbf{Y}}\,\underline{\mathbf{U}}$, which is performed once per iteration, and $\mathbf{J}$ is used to finally compute the iteration steps with the solution of the linear system $[\mathbf{P_{mis}}\ \mathbf{Q_{mis}}]^T = \mathbf{J}\,[\Delta\phi\ \Delta|\mathbf{U}|]^T$ in line 5 of algorithm 4. The COO-representation of sparse matrices does not directly permit a solution of linear systems, so it is omitted. Figure 5.5 shows the runtime of the sparse CSR and CSC representation against the non-sparse version. The grid is a radial low-voltage grid with an increasing number of nodes, and runtime was determined for 350 individual load scenarios.

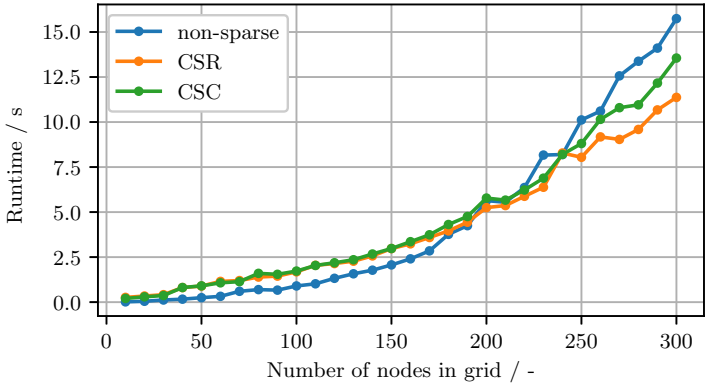


Figure 5.5: Runtime of Y_{BUS} methods with different sparse matrix storage schemes. Measured with a radial low-voltage grid and a time series with 350 loads

The break-even point for the sparse versions in this case is around 200 nodes. The CSR version has an advantage over the CSC version for

larger grids.
In summary, the sparse-matrix versions of the Z_{BUS} Jacobi and the Y_{BUS} Newton-Raphson method only have performance benefits if the grid is sufficiently large. For most distribution grids, the regular, dense formulation is more performant.

5.3 Grid Reduction Methods

Model reduction is an obvious candidate for a method to increase the computation speed of any simulation. As mentioned in section 3.1, the power flow model used in the previous chapters is already the result of several reductions in model order and complexity. A further reduction can lead to smaller vectors and matrices, therefore better usage of caches and fewer computations in general. There are general methods to reduce the order of arbitrary mathematical models, but for electric grids, several effective reduction schemes can be formulated by looking at the topology of the grid alone.
The most trivial of those schemes is the deletion of nodes that are completely irrelevant for the computation. Although it seems superfluous to have those nodes in the power flow model in the first place, they occur surprisingly frequently in grid models that stem from sources whose primary purpose is not the electrical accuracy, but e.g. the exact geopositions of the equipment. *Lossless grid reduction methods* which eliminate these nodes can change the number of nodes and the topology of the grid, but never the result of a power flow computation beyond the removal of redundant information.
Lossy grid reduction methods also aim to reduce the number of nodes and lines, but in turn sacrifice some accuracy. With these, as with all lossy methods, a cost-benefit assessment has to be conducted before applying the method.
In practice, the usage of a grid reduction method also means a little overhead for the reduction method itself and the re-ordering and reassignment of the nodes after the power flow computations, unless the subsequent operations can proceed with the reduced dataset.

5.3.1 Lossless Grid Reduction Methods

The lossless grid reduction methods have the goal of eliminating nodes and lines which contribute nothing to the solution of the power flow

and whose elimination leaves the relevant results of the computations unchanged. Basically, a node can be eliminated if it

1. has no load attached to it and
2. is not a slack node and
3. is not a junction (has three or more lines attached to it).

This leaves two variants of nodes: transit nodes (two lines attached) and end nodes (one line attached). Finding and eliminating these nodes using the admittance matrix as a data structure is not ideal, because it changes the size of the matrix multiple times, causing a number of costly copy operations in memory. If possible, these grid reduction methods should be carried out in a preliminary data structure. The exact algorithm formulation depends on that data structure.
The lossless reduction of these nodes can be carried out in multiple atomic steps, each removing only one transit or end node until no further reduction is possible.
Figure 5.6 shows the (trivial) atomic operation of end node removal.

Figure 5.6: End node reduction: removal of unnecessary nodes at the end of lines

Figure 5.7 shows the atomic operation of the transit node removal, where two lines are combined into a single line with impedance $\underline{Z}_{13} = \underline{Z}_{12} + \underline{Z}_{23}$.

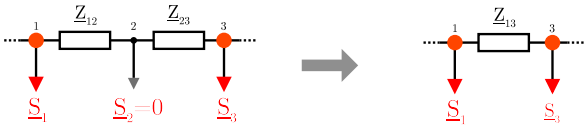


Figure 5.7: Transit node reduction: removal of unnecessary nodes in the middle of lines

Although it seems trivial, an efficient implementation involves several iterations through the nodes and the lines. The two operations can be performed together, so that each node is first checked for end node removal (one connected node), and then transit node removal (two connected nodes). The lossless reduction is applied in section 6.1.

5.3.2 Lossy Grid Reduction Methods

Lossy grid reduction methods employ similar atomic operations. However, the removed elements actually have an influence on the power flow. To minimize the error caused by this reduction, the powers at the remaining nodes can be modified.
Again, there are two atomic operations that can be applied iteratively until the grid is reduced to a desired degree. Figure 5.8 shows the atomic operation of lossy transit node reduction.

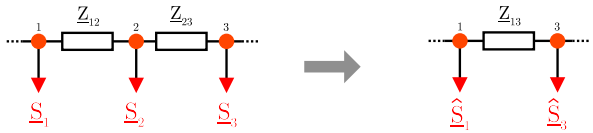


Figure 5.8: Atomic operation of lossy transit node reduction: A segment with three nodes and two lines is reduced to two nodes and one line

In order to eliminate the middle node 2 of the three-node segment, the impedances $\underline{Z}_{12}$ and $\underline{Z}_{23}$ need to be combined into one and the power $\underline{S}_2$ needs to be distributed to the remaining powers $\underline{S}_1$ and $\underline{S}_3$. The choice of

$$\underline{Z}_{13} = \underline{Z}_{12} + \underline{Z}_{23} \tag{5.5}$$

is rather trivial, as the reduced grid segment should behave identically to the rest of the grid as the non-reduced segment with two lines. By choosing e.g.

$$\hat{\underline{S}}_1 = \underline{S}_1 + \frac{\underline{Z}_{23}}{\underline{Z}_{12} + \underline{Z}_{23}} \underline{S}_2 \text{ , and} \tag{5.6}$$

$$\hat{\underline{S}}_3 = \underline{S}_3 + \frac{\underline{Z}_{12}}{\underline{Z}_{12} + \underline{Z}_{23}} \underline{S}_2 \text{ ,} \tag{5.7}$$

the load of node 2 is distributed to the two other nodes. The voltage drop over the reduced line segment is of course slightly different. This error is proportional to the power $\underline{S}_2$. When selecting which grid segment to reduce, the segment with the low power $\underline{S}_2$ should therefore be chosen.
Figure 5.9 shows the lossy end node reduction.

Figure 5.9: Atomic operation of end node reduction: a segment with one line and two nodes, one of which has no further connections, is reduced to one node

In this example, the power $\underline{S}_2$ of the deleted node as well as the power $\underline{S}_{Loss,12}$ that would be caused by the current flowing through the impedance $\underline{Z}_{12}$ needs to be represented as accurately as possible by $\hat{\underline{S}}_1$:

$$\hat{\underline{S}}_1 = \underline{S}_1 + \underline{S}_2 + \underline{S}_{Loss,12} \tag{5.8}$$

$$\hat{\underline{S}}_1 = \underline{S}_1 + \underline{S}_2 + \underline{Z}_{12} \left(\frac{\underline{S}_2}{\underline{U}_2} \right) \left(\frac{\underline{S}_2}{\underline{U}_2} \right)^* \tag{5.9}$$

$$\hat{\underline{S}}_1 = \underline{S}_1 + \underline{S}_2 + \underline{Z}_{12} \left(\frac{|\underline{S}_2|}{|\underline{U}_2|} \right)^2 \tag{5.10}$$

Of course, at the time of the grid reduction, $\underline{U}_2$ is not known and has to be approximated. The simplest approximation is $\underline{U}_2 = \underline{U}_0$, but a safer assumption would be setting $\underline{U}_2$ to the lowest point of acceptable operation, e.g. $\underline{U}_2 = 0.9\,\underline{U}_{slack}$. This ensures that the approximation made by the grid reduction operation does not qualitatively change the result in terms of lower limit norm violations: If the real voltage $\underline{U}_2$ is greater than $0.9\,\underline{U}_{slack}$, $\underline{S}_2$ will be overestimated, so that the voltage at node 1 is actually lower than the voltage at node 2 in the non-reduced grid, but never to the point of pushing $\underline{U}_2$ below the chosen boundary voltage. That way, the qualitative property of whether the grid voltages are above safe lower operational bounds is preserved.

In the case of negative loads and potential violation of upper voltage bounds, an approximation $\underline{U}_2 = 1.1\,\underline{U}_{slack}$ is also possible. If there is uncertainty about the nature of the load, the computation could also be executed twice, once for the approximation with a lower and once with an upper bound.
Additionally, the assumption for $\underline{U}_2$ can be used to estimate the voltage drop over the impedance $\underline{Z}_{12}$ and therefore a criterion for which node to reduce first via this method. With an assumption for the maximal load $\underline{S}_{2,max}$ at node 2, the expected voltage drop $\underline{U}_{exp}$ can be calculated to

$$\underline{U}_{exp} = \underline{Z}_{12} \left(\frac{\underline{S}_{2,max}}{\underline{U}_2} \right)^* . \tag{5.11}$$

Among the candidate nodes for end node reduction, the node with the lowest expected voltage drop should be reduced first.
With these two atomic operations, a grid reduction algorithm can be designed in multiple ways. The decision which nodes to reduce first in the grid and when to stop is an optimization problem in itself. In general, the transit node reduction is the less intrusive method and should be preferred whenever possible.
There is no hard limit to this lossy reduction until the grid consists of just one node. In most cases however, this will probably come at an unacceptable cost of accuracy. Therefore, a criterion for an acceptable degree of reduction is required. The simplest criterion is the number of nodes left (see also section 6.1), but more complex criteria like U_{exp} or an upper bound on the joined impedance Z_{13} could be chosen. The choice depends highly on the specific problem at hand.
Section 6.1 shows an application of lossless and lossy grid reduction methods and demonstrates the potential performance gains.

5.4 Weak Load Detection

In practice, power flow computations are often applied to time series in order to find violations of standards and the edges of safe operation. These critical situations usually only occur for a select few time steps, so the simulations to find them often contain a large set of load conditions which are far from critical and therefore uninteresting for the actual purpose of the simulation. This is frequently the case when the input loads are large time series datasets and include nighttime, weekends, and hol-

idays. The idea of weak load detection techniques is to identify those time steps and 'bypass' parts of or the entire power flow computation in these cases. Instead, these cases can be deleted entirely, or a completely 'flat' (all voltages $\underline{U} = \underline{U}_{slack}$) solution can be assumed. The latter has the practical advantage that the matrices containing the values do not need to be resized, and that the total number of solutions (i.e. voltage vectors) stays the same, which might be important for statistical considerations.
There are two ways to implement weak load detection: as a separate decision algorithm which sorts out problems before the power flow computation even starts, and as an additional convergence criterion, which terminates the algorithm prematurely when it is predictable that the resulting voltage will be within certain bounds. Both approaches need to be carefully applied with special attention to the interpretation of the raw results. In general, the second approach, a modified convergence criterion, is easier to correctly apply than the first, the separate detection.
Section 6.2 contains a practical application of a slightly modified separate weak load detection to reduce the runtime of a large-scale simulation.

5.4.1 Separate Weak Load Detection

The simplest way to implement a weak load detection method is as a separate procedure, which checks every time step of a time series computation before the power flow computation is run. The goal of that procedure is to quickly deduce from the given loads if the resulting voltages are safely within certain bounds or not. Algorithm 11 shows an implementation of this principle in lines 2 and 3, using a predetermined cutoff value S_{break} which is explained below.

Algorithm 11 Z_{BUS} Jacobi Method with separate weak load detection

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}_{all}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{\mathbf{U}}_{slack}$, S_{break}, ϵ_S

1: $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$
2: **for** $\underline{\mathbf{S}}$ **in** $\underline{\mathbf{S}}_{all}$ **do**
3: **if** `sum(abs(`$\underline{\mathbf{S}}$`))` $< S_{break}$ **then**
4: **return** $\underline{\mathbf{U}}^{(0)}$
5: **loop**
6: $\underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
7: **if** $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **then**
8: **break loop**
9: $\underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{slack}$

return $\underline{\mathbf{U}}$

The weak load criterion in line 2 (`sum(abs(`$\underline{\mathbf{S}}$`))` $< S_{break}$) is in this case a lower bound on the sum of the absolute loads, which is very fast to evaluate. If the sum of the loads is below the given bound S_{break}, the computation is stopped right there and the initial value of the voltage is returned. There are of course other plausible criteria. The chosen decision value, in this case S_{break}, must be chosen mindfully and ultimately depends on the topology and impedances of the grid. The best way to find a suitable value is either to look at previous simulation results in the same grid (see below) or to perform a dedicated simulation run with selected values to find an absolute bound on S_{break} (see section 6.2).
Regarding the correctness of the weak load criterion, the number of false negatives - load scenarios that are superfluously exactly computed although the voltage is within these safe bounds - should be minimized, but false positives - load scenarios which are erroneously detected as safe but are really not - need to be avoided at all cost. Ideally, the method should calculate an easy-to-compute indicator from the set of loads which has a high correlation to the "worst" voltage in the grid.
As an example, Figure 5.10 shows a cloud of points which each represent a load condition on the network. The y-coordinate is the sum of absolute loads - the weak load criterion S_{break} is then horizontal line on the graph, and all points below that line are bypassed by the algorithm. The x-coordinate is the minimal resulting voltage in the grid, which is one possible criterion for the correctness of the weak load criterion. The black line is positioned at 0.99 pu, or a maximal voltage drop of 1%.

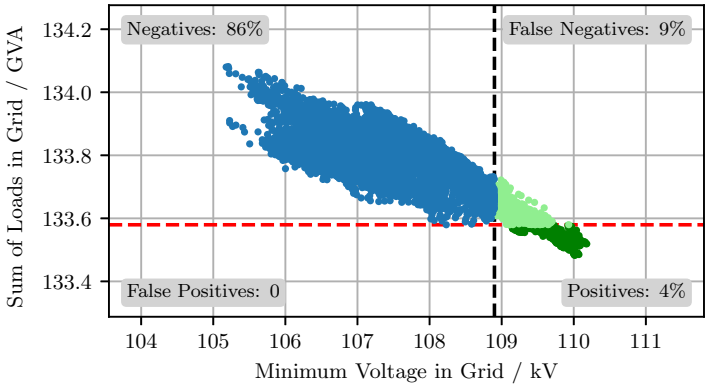


Figure 5.10: Correlation between the sum of all loads and the minimal voltage in the grid for a real high voltage grid with 110 nodes and 8064 loads

An ideal weak load detection would detect all points right of the black line, but in reality, the simple criterion used here can only detect the cases below the red line. This separates the cloud of points into four quadrants:

- The *Negatives* (upper left, blue points), load conditions which have a higher sum of loads than the bound and therefore are normally computed by the power flow method. There is no speed up for these loads, only a slight overhead for the evaluation of the weak load criterion.
- The *False Negatives* (upper right, light green points), load conditions which also have a higher sum of the loads than the bound and are therefore normally computed. Looking at the results however, these load conditions did not lead to a voltage drop of more than 1%, so in theory, the solution could have been bypassed. A better weak load criterion might be able to catch these load conditions.
- the *Positives* (lower right, dark green points), load conditions which were correctly identified by the weak load criterion as low-load scenarios which do not merit a full computation.
- The *False Positives*, (lower left, no points), load conditions which were wrongly identified by the weak load criterion, although their maximal voltage drop was greater than 1 %. In this example, the bound was intentionally chosen after the fact so that no points

are in this quadrant, in order to show the maximum potential of the method. An a priori choice of the weak load criterion is not straightforward.

As Figure 5.10 shows, the runtime optimization potential in this real example is just 4%, even if the constant overhead for the weak load criterion is ignored. Section 6.2 shows an example in which a separate weak load detection is applied to greater effect to a Monte-Carlo simulation.

5.4.2 Modified Convergence Criterion

The idea of bypassing a solution for obviously uncritical load scenario can also be directly incorporated into a power flow algorithm with a modified convergence criterion. If the computed voltages in the first iteration are already far from any critical region, this is unlikely to change in the remaining iterations. Of course, this highly depends on the power flow method used. In fact, in any other than the Y_{BUS} Newton and the Z_{BUS} Jacobi methods, the iteration progression is likely too uneven and unpredictable to prevent major problems with false positive solutions. Algorithm 12 shows how the Z_{BUS} Jacobi algorithm can be modified with an additional breaking condition in lines 7 and 8 using a cutoff value U_{break}.

Algorithm 12 Z_{BUS} Jacobi Method with modified convergence criterion

Input: $\underline{\mathbf{Y}}$, $\underline{\mathbf{S}}$, $\underline{\mathbf{U}}^{(0)}$, $\underline{\mathbf{U}}_{slack}$, U_{break}, ϵ_S
1: $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$
2: $m = 0$
3: **loop**
4: $\quad \underline{\mathbf{I}} = (\underline{\mathbf{S}} \oslash \underline{\mathbf{U}})^*$
5: $\quad$ **if** $\|\underline{\mathbf{U}} \odot \underline{\mathbf{I}}^* - \underline{\mathbf{S}}\|_\infty < \epsilon_S$ **then**
6: $\quad\quad$ **break loop**
7: $\quad$ **if** $m == 1$ **and** $|\texttt{min}(\underline{\mathbf{U}})| > U_{break}$ **then**
8: $\quad\quad$ **break loop**
9: $\quad \underline{\mathbf{U}} = \underline{\mathbf{Z}}\,\underline{\mathbf{I}} + \underline{\mathbf{U}}_{slack}$
10: $\quad m = m + 1$
return $\underline{\mathbf{U}}$

Figure 5.11 shows the results for the same 110-node high voltage grid and 8064 loads as used in Figure 5.10, using a cutoff value $U_{break} =$

$0.99\,U_{slack}$. The sum of absolute powers on the y-axis and the bound visualized by the red line in Figure 5.10 is not used in this algorithm.

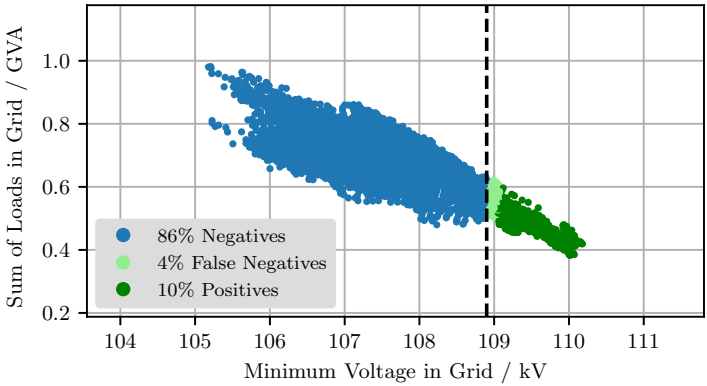


Figure 5.11: Example result of power flow "short-circuiting" with a modified convergence criterion in a Z_{BUS} power flow method: dark green points represent successfully detected weak load scenarios.

In the presented example, 10% of power flow computations could be bypassed, and the false negative-rate drops to 4%. In terms of iterations, the modified convergence criterion saves 2831 of a total of 44355 iterations, or 6.4%.

In this case, the method does not produce any false positives, but in theory, they can occur if a first iteration is totally unrepresentative of the final result of the power flow computation. In practical problems, this should be an extreme exception.

The choice of U_{break} depends again on the exact problem and the willingness to trade precision for speed, but a value like $U_{break} = 0.99\,U_{slack}$ is a conservative starting point.

In summary, the concept of weak load detection has its limits and must be rooted in a solid understanding of the goals and the scope of the simulation. A weak load detection can considerably accelerate the computation, but statistics like mean values of voltages and line currents might be skewed, and probabilistic interpretations of the results could be far off, as some of the percentiles are changed. There are many practical applications where this is not a problem, but it should be always kept in mind that the solutions of some individual scenarios might not be

technically correct.

5.5 Parallelization

Ever since CPUs have featured multiple cores, parallelization has been a straight-forward computational optimization strategy. Consumer-grade CPUs today (2020) commonly have between 4 and 16 cores which promises a straightforward way to speed up the computation by a factor that is equal to the number of cores. If even more performance is required, a parallelized power flow computation could also be distributed onto multiple machines. Leveraging a cloud provider, the performance therefore could, in theory, scale to all reasonable needs. That way, any size of problem can be solved in almost any time by renting appropriate server capacity.

As with many practical problems, power flow computations can be parallelized along different axes, i.e. the parallelization can be applied to different sections of code [19].

On the one end of the spectrum, the parallelized section can be very small, inside the BLAS or LAPACK routines, and therefore affect a single matrix multiplication or solution of a linear system. This requires the least customization (in most cases just a configuration setting), but it also offers the least performance increase, because the parallel processes need to be forked and joined frequently.

If the computation task involves more than one load scenario, the computations can also be parallelized by distributing a fraction of the number of load scenarios onto each process. In that case, each process executes a number of complete power flow computations, and the results are joined only after the whole set has been completed. This requires an explicit separation of the set of load scenarios, but the less frequent forks and joins improve performance. Results for the performance increase using this parallelization strategy are shown below.

Finally, if the practical problem is more complex than a simple computation of voltages, the generation of the data and the evaluation of the results can also be included in the function that is then distributed onto multiple processes. This is especially promising if the evaluation step reduces the raw voltage data significantly in size, e.g. by performing statistical analyses. Section 6.2 shows an example where this

third parallelization option yields a considerable increase in performance.

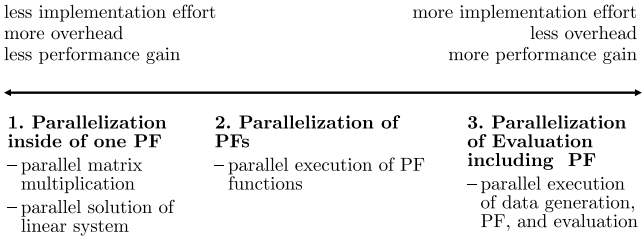


Figure 5.12: Different axes of parallelization for power flow computations

In general, this third option should be chosen wherever possible, but the second approach (parallelization over multiple sets of load scenarios) can be implemented without specific knowledge about the practical problem and can therefore always be an option.
However, not every power flow solution method is equally parallelizable. The speedup factor achievable by a parallel solution is limited by the memory access properties of the algorithm. In modern CPUs, the L1 and L2 caches are usually repeated in every core, but the L3 cache is shared between all cores. Access to the L3 cache is therefore always serial, and algorithms that require frequent access to the L3 usually profit less from parallelization. In any case, the distribution of the functions and the accompanying data onto the different cores as well as the data aggregation and merging at the end are a constant overhead. If the problem is too small, this overhead might not be recoverable by a faster power flow solution.
Also, as mentioned above, certain individual BLAS and LAPACK routines are already parallelized in some implementations. Any additional parallelization on top can also result in a net loss of performance.
All these additional conditions lead to different properties of the algorithms with respect to performance gains from parallelization.
The following performance measurements were made with an Intel 7260U CPU, which has 2 cores and 4 threads, a base clock frequency of 2.2 GHz, and a max. clock speed of 3.4 GHz. All parallel computations were performed with 4 processes, so the theoretically best result is a factor of 0.25 compared to single-threaded execution. In practice, the simulta-

neous multiprocessing capabilities do not usually lead to a significant speedup, if the program is sufficiently optimized.

Y_{BUS} Fixed-Point Methods

As discussed in section 4.5, the Y_{BUS} fixed point methods are slowly converging algorithms from a mathematical point of view, but are very well adapted to modern CPUs. With the hardware used (see 2.4), they are not held back by access to memory or caches, and can leverage the full computational power of the CPU. This indicates that they might also strongly benefit from parallelization.
However, these improvements can not be realized for every size of power flow problem. Smaller grids and a lower number of individual cases to compute do not profit as much and can even take a longer time to compute due to the constant overhead. Figure 5.13 shows a heatmap of the runtime improvements for different combinations of grid size and number of loads in a single feeder low voltage grid for the Y_{BUS} Jacobi method, using the parallelization strategy 2, as defined above. The Y_{BUS} Gauss-Seidel method shows the same characteristics.

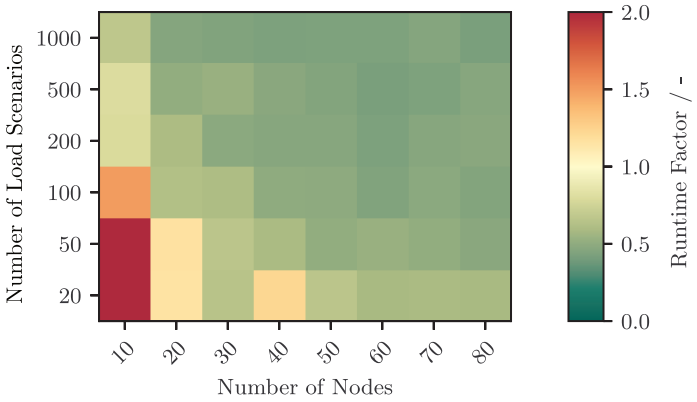


Figure 5.13: Heatmap of performance improvements by parallelizing the Y_{BUS} Jacobi method for different grid sizes and number of loads. Average of 10 runs.

In the heatmap, the orange and red areas represent combinations of grid and problem size where an explicit parallelization is actually harmful. For example, a trivially small problem with a grid of 10 nodes and 20 load scenarios is solved by the unparallelized Y_{BUS} Jacobi method in

0.6 ms, while a Y_{BUS} Jacobi method parallelized to 4 threads solves it in 5 ms, a slow-down by a runtime factor of 8.3. On the other hand, a bigger power flow problem with 80 nodes and 1000 load scenarios is sped up from 4.33 s to 1.76 s, a runtime factor of 0.4.
The results show that problems with more than 200 loads and with grids with more than 50 nodes can profit from parallelization.

Y_{BUS} Newton-Raphson Method

A similar heatmap is plotted for the Y_{BUS} Newton-Raphson method in Figure 5.14. There are two additional problems with a parallelization of the Y_{BUS} Newton-Raphson method: First, the solution of the system of linear equations, which is the main numerical function inside of the Y_{BUS} Newton-Raphson method, is already parallelized in many high-performance libraries like LAPACK and the MKL. Secondly, as shown in section 4.5, the method is not only bounded by the CPU itself, but also by the L3 cache. Because the L3 cache is shared between all the cores, an execution on multiple cores results in a longer runtime for many problems. Only problems with many nodes and a high amount of loads can therefore profit from parallelization.

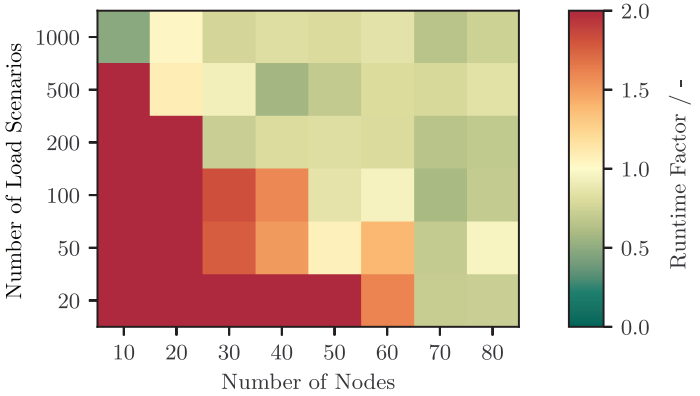


Figure 5.14: Heatmap of performance improvements by parallelizing the Y_{BUS} Newton-Raphson method for different grid sizes and number of loads. Average of 10 runs.

Z_{BUS} Jacobi Method

The Z_{BUS} Jacobi method is even harder to parallelize effectively, as seen in Figure 5.15. Only when a power flow computation task reaches a very big grid size and/or number of load scenarios, it is worthwhile to parallelize it. For smaller problems, the Z_{BUS} Jacobi method is often so fast that any overhead can not be recovered. In that sense, the problems with parallelization of the Z_{BUS} Jacobi are actually a consequence of its strong performance.

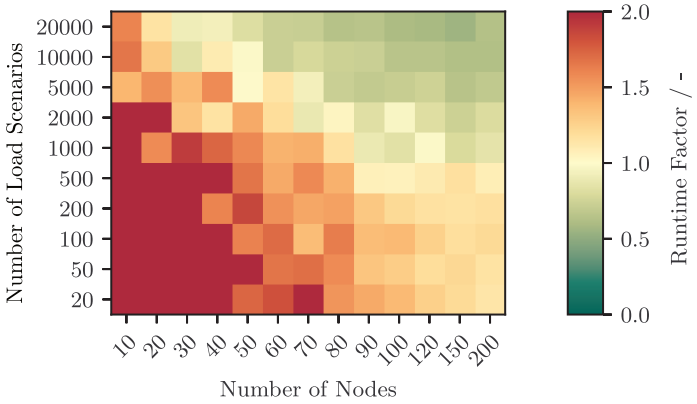


Figure 5.15: Heatmap of performance improvements by parallelizing the Z_{BUS} Jacobi method for different grid sizes and number of loads. Average of 10 runs.

Chapter 6

Power Flow Case Studies

In order to show the usefulness of the presented algorithms and optimization approaches, three benchmark problems are presented in this chapter. They are selected to not just be generic benchmark problems, but practical problems involving a realistic set of input data and an evaluation of the results.
All considered problems contain a large number of individual power flow computations. Those are the problems where a faster algorithm, a better implementation, or an effective optimization can make a significant difference.
In addition to the benchmark results of the selected power flow methods and optimization approaches, the runtime of the open source power flow library PYPOWER is also given. PYPOWER contains an implementation of the Y_{BUS} Newton-Raphson method, but its internal representation of slack nodes and its separated functions are detrimental to its performance compared to an implementation following the algorithm outlined in section 4.2.

6.1 Time-Series Power Flow Computations in the European LV Feeder

The European LV Feeder is a test grid published by the IEEE PES Distribution Systems Analysis Subcommittee [50]. The data stems from a real grid in Manchester, UK and aims to represent an average European residential low-voltage grid [97]. It models 55 households in a 416 V

system. The grid is supplied by an overlaying 11 kV grid via one 800 kVA transformer. A time series with 1440 power values is provided for every load, which represents one day in one-minute resolution. The grid model contains 906 nodes and 905 lines, many of which were introduced to mimic the exact geographic route of the cables.
Figure 6.1 shows the single-line diagram. The red dots are nodes with a load, the gray dots all represent nodes which do not supply any load. Some of these are required to model a cable junction, but the majority has no such purpose and exists solely to model the geographic path of the lines. This grid model contains a lot of redundant information and is almost tailor-made for a grid reduction algorithm.

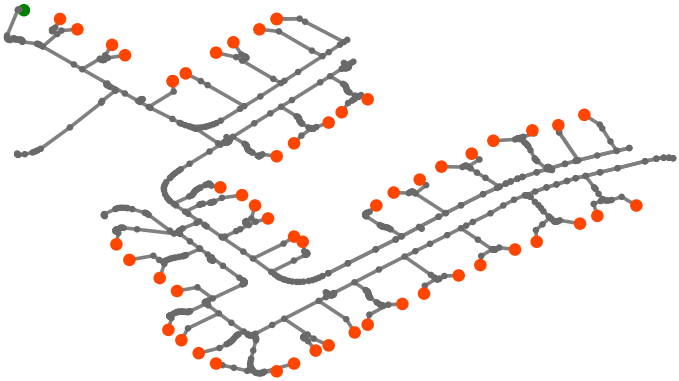


Figure 6.1: Original grid topology of the European LV Feeder test grid, comprised of 906 nodes and 905 lines

The runtimes of the computation of all 1440 load scenarios in sequence for a selection of methods are shown in table 6.1. The relative runtime as compared to the runtime of PYPOWER is also shown. Without any reduction, the size and topology of the grid limits the choice of algorithm to Y_{BUS} Newton-Raphson or Z_{BUS} Jacobi.

Table 6.1: Runtimes and relative runtimes as compared to PYPOWER for European LV Feeder grid, single-phase case, no optimizations

PYPOWER	no customizations	50m 10s	100 %
Y_{BUS} Jacobi	no optimizations	2h 59m	356 %
Y_{BUS} Gauss-Seidel	no optimizations	3h 10m	379 %
Y_{BUS} Newton	no optimizations	12m 1s	24.0 %
Z_{BUS} Jacobi	no optimizations	4.8s	0.16 %

The nature of this distribution grid, namely a comparatively large node count, but not so large that the admittance matrix has trouble fitting into main memory and no PV nodes combined with a time-series computation without any changes in the grid, plays to the strengths of the Z_{BUS} Jacobi method, which is around 150 times faster than the Y_{BUS} Newton-Raphson method in this case.

Sparse Matrix Formulations

The high number of nodes (906) suggests that the sparse-matrix-versions of the methods may be faster, and they are indeed both around 30% faster, as shown in table 6.2.

Table 6.2: Runtimes and relative runtimes as compared to PYPOWER for European LV Feeder grid, single-phase case, sparse-matrix versions

Y_{BUS} Newton	sparse-matrix version (CSR)	8m 48 s	17.6 %
Z_{BUS} Jacobi	sparse-matrix version (CSR)	3.7 s	0.12 %

Lossless Grid Reduction

As mentioned, the node count can be massively reduced with just lossless grid reduction techniques. After removing the loose ends and deleting intermediate nodes using the algorithms in section 5.3.1, 110 nodes and 109 lines remain in the grid, as shown in Figure 6.2.

As expected, power flow computations in this grid are much faster, as recorded in table 6.3.

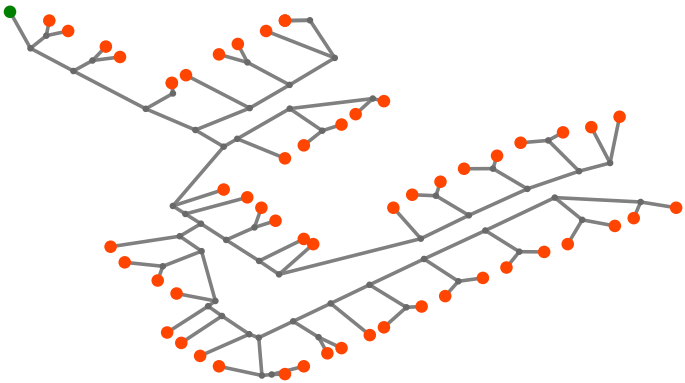


Figure 6.2: Losslessly reduced grid topology of the European LV Feeder test grid, comprising 110 nodes and 109 lines. Two seemingly superflous nodes can be seen in the upper half of the grid, but they actually connect two load nodes with identical coordinates.

Table 6.3: Runtimes and relative runtimes as compared to PYPOWER for European LV Feeder grid, single-phase case, after grid reduction.

Y_{BUS} Newton	lossless grid reduction	750 ms	0.025 %
Z_{BUS} Jacobi	lossless grid reduction	80 ms	0.0027 %

Compared to the initial runtimes in table 6.1, these timings represent a speedup factor of 60 for the Z_{BUS} Jacobi method and 961 for the Y_{BUS} Newton method. In this smaller grid, the sparse-matrix-versions of the methods are actually detrimental to performance, as shown in table 6.4.

Table 6.4: Runtimes and relative runtimes as compared to PYPOWER for European LV Feeder grid, single-phase case, sparse-matrix versions, after grid reduction

Y_{BUS} Newton	lossless grid reduction & sparse-matrix-version (CSR)	1.9 s	0.063 %
Z_{BUS} Jacobi	lossless grid reduction & sparse-matrix-version (CSR)	127 ms	0.0042 %

Lossy Grid Reduction

If an approximate solution is acceptable, a lossy grid reduction can reduce the computation times even further. Figure 6.3 shows the grid model after being reduced to 50 nodes, and Figure 6.4 shows the voltage error introduced by that reduction. The first plot shows the minimal voltage in the non-reduced (orange) and the reduced (blue) grid model for every timestep, and the second plot shows the difference in the minimal voltages, where a positive difference means a higher voltage of the reduced grid. The runtimes of the Y_{BUS} Newton method and the Z_{BUS} Jacobi method are shown in table 6.5.

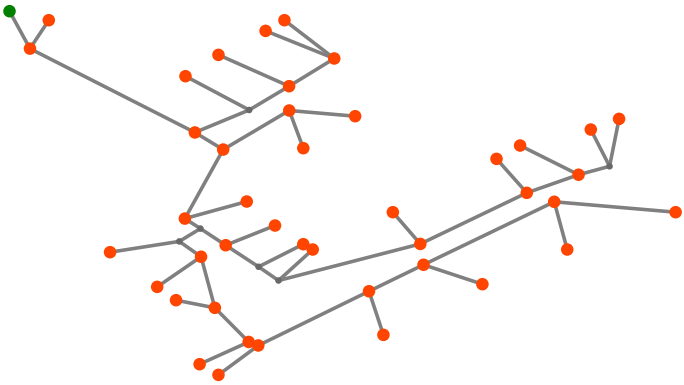


Figure 6.3: Grid model of the European LV Feeder reduced to 50 nodes

Table 6.5: Runtimes and relative runtimes as compared to PYPOWER for 1440 load cases in the European LV Feeder grid, single-phase case, after lossless grid reduction and lossy reduction to 50 nodes

Y_{BUS} Newton	lossless grid reduction + lossy grid reduction to 50 nodes	637 ms	0.021 %
Z_{BUS} Jacobi	lossless grid reduction + lossy grid reduction to 50 nodes	21 ms	0.0007 %

The reduction to 50 nodes leads to a speedup factor of 3 in the case of Y_{BUS} Newton, and 4 in the case of Z_{BUS} Jacobi. The deviations

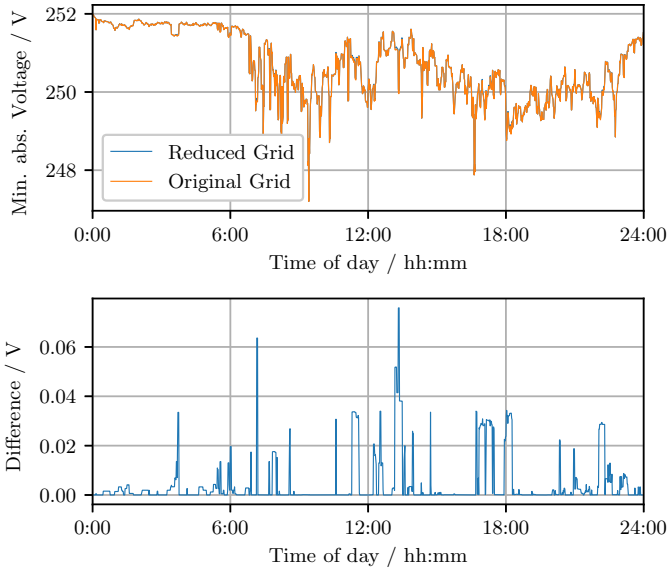


Figure 6.4: Minimal absolute voltages of non-reduced grid model and grid model reduced to 50 nodes and difference between the minimal voltages below

from the non-reduced case (see the lower plot in figure 6.4) max out at around 75 mV ($0.0003\,pu$). This is the maximum magnitude of the overestimation of the voltage drop caused by the lossy grid reduction.

Figure 6.5 shows the grid model of the European LV Feeder after being reduced to just 5 nodes.

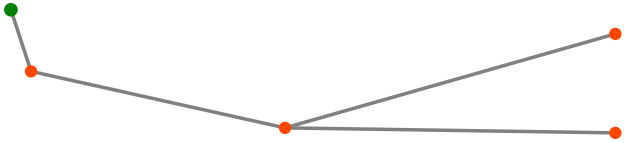


Figure 6.5: Grid model of the European LV Feeder reduced to 5 nodes

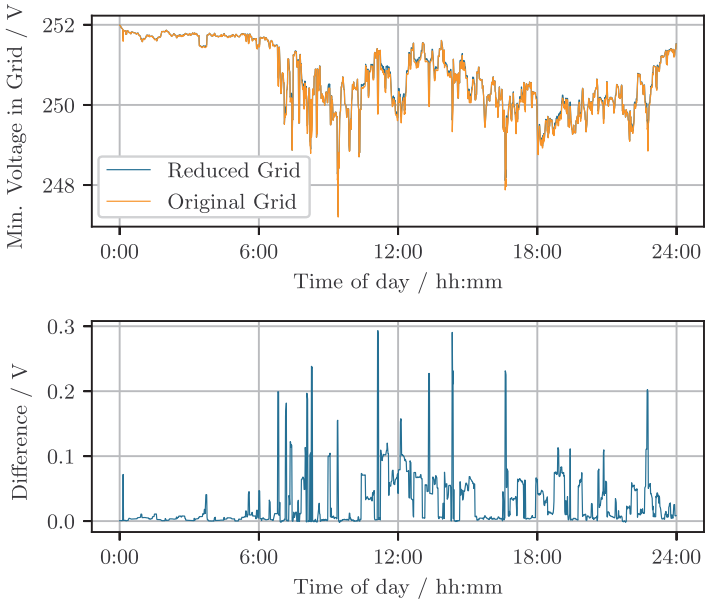


Figure 6.6: Minimal absolute voltages of non-reduced grid model and grid model reduced to 5 nodes

Table 6.6: Runtimes and relative runtimes as compared to PYPOWER for 1440 load cases in the European LV Feeder grid, single-phase case, after lossless grid reduction and lossy reduction to 5 nodes

Y_{BUS} Newton	lossless grid reduction + lossy grid reduction to 5 nodes	18 ms	0.0006 %
Z_{BUS} Jacobi	lossless grid reduction + lossy grid reduction to 5 nodes	2.6 ms	0.000086 %

Although the shape of the grid barely resembles the original grid, the deviations in minimal absolute voltages (Figure 6.6) would be acceptable in many contexts. The difference between the non-reduced and the reduced grid model never exceeds 0.3 V (0.0013 pu). The runtime is improved by factors of 35 and 6 for the Y_{BUS} Newton and the Z_{BUS} Jacobi methods, respectively.
To summarize, the runtime for a power flow computation in this grid could be improved from 12 minutes for the Y_{BUS} Newton-Raphson method without any optimizations to 80 ms by using the Z_{BUS} Jacobi method and a lossless grid reduction, without any loss in precision (a speedup factor of 9000). By allowing a certain precision loss by using a lossy grid reduction, the runtime could even be reduced to 2.6 ms (a speedup factor of more than 250,000).

6.2 Monte Carlo Simulation of EV Penetration in a Low-Voltage Grid

A Monte Carlo simulation is a method which aims to produce statistics about a complex process or function by evaluating that function for a large set of stochastically distributed input data. In power grid research, Monte Carlo simulations are often used to evaluate scenarios regarding the integration of volatile generation [30] and electric vehicles [89] into distribution grids. These kinds of analyses entail a large number of uncertain, stochastic variables like weather conditions and human behavior that need to be evaluated with all their permutations, or at last as many as possible. To reach statistical significance, these simulations routinely contain thousands or millions of individual power flow computations.
As an example problem developed for this thesis, consider the low voltage grid in Figure 6.7. The grid is radial and contains 53 nodes, 40 households, and 21 PV plants. It is modeled after a rural, German low voltage grid with high PV penetration.

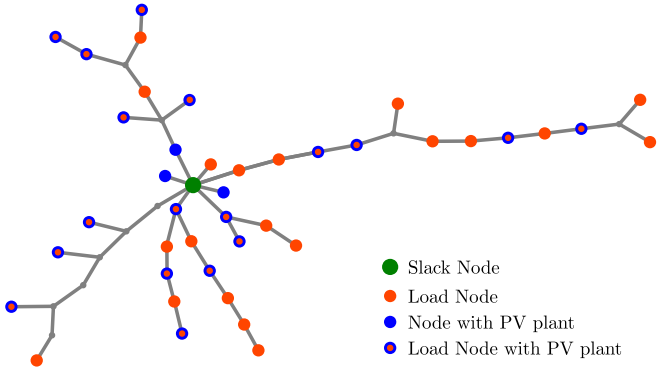


Figure 6.7: Example low voltage grid with high PV penetration

A typical investigation that might lead to a Monte Carlo simulation is the evaluation of the hosting capacity for electric vehicles in this grid. If (or rather, when) the inhabitants start buying electric vehicles and charge them at their homes, the grid operator should investigate what number of electric vehicles and what charging power can be supported by the existing grid infrastructure and what leads to voltage band violations. To illustrate the computational complexity of such a simulation and eval-

uate the runtimes and optimization potential, this section presents a model simulation with simplified data setup and result evaluation, but a full-scale power flow simulation.

The impact of the electric vehicles on the actual load on the grid depends on many parameters, like the charging power, the time at which the charging starts and the distances driven. In order to roughly simulate these influencing factors, the electric vehicle charging processes are modeled as outlined in Figure 6.8. In this model, the charging power is an input variable, the consumption of the EVs is assumed to be uniform with 17 kWh/100 km, and the km driven before the charging process as well as the start time of the charging process are the results of simple stochastic functions. The distance driven is randomly selected according to a Gamma distribution with $k = 18$ and $\theta = 2$ to model the distribution shown in Figure 10 of [100], and the start time of the charging process is selected from a normal distribution with $\mu = 18$ and $\sigma = 1.5$, expressed in hours of the day, as in [24]. μ was shifted to 18 to better represent behaviour of consumers in a rural, German area as opposed to a urban American area. The charging power itself is assumed to be constant over the duration of the charging process, and does not level off towards the end, as it does in reality. The household and PV loads are given as a fixed time series of one year.

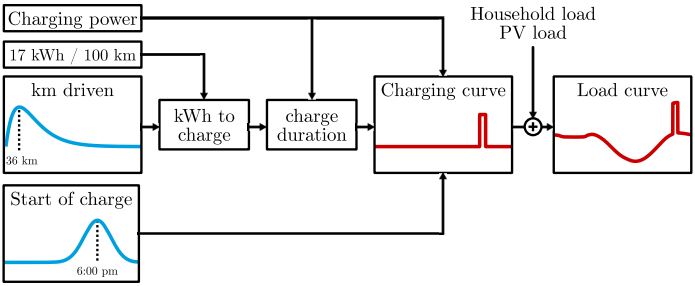


Figure 6.8: Probabilistic process to generate the load curves of households with electric vehicles

An important factor for the validity of the simulation results is the time resolution. Many simulations involving households and PV plants work with one-hour, 15-minute, or 10-minute resolutions. In the case of electric vehicles with high charging powers however, this might not be fine-grained enough, as a charging process might only take a few minutes, and multiple charging processes might overlap in the simulation when

they would not in reality. The combination of fine-grained time resolution (one minute in this case, for a total of 1440 simulations for one day) and a high number of days that must be simulated to achieve a statistically significant result leads to a large number of individual power flow simulations. For this example, the combination and permutation of

1. EV counts of 10, 20, 30, 40, 50, 60, and 70 (7 variants),

2. EV charging powers of 11 kW, 22 kW, 43 kW, 80 kW, and 100 kW (5 variants),

3. a simulation duration of 1 year for each electric vehicle distribution (meaning 365 days),

4. 1440 simulations per day for a one-minute resolution, and

5. repeated 10 times to account for different vehicle distributions

leads to a problem with $181,440,000$ individual power flow simulations. The result of this simulation is an equal number of voltage profiles. The evaluation and interpretation of these profiles is a problem in itself. As a simple first evaluation, Figure 6.9 shows the resulting number of minutes with voltage band violations of more than 10% (i.e. lower than 360 V at at least one node) for each combination of number and charging power of electric vehicles. Notably, this plot does not show how bad a voltage band violation is or how much the lines and the transformer are loaded. This is obviously a very limited evaluation, but the power flow computation itself is equivalent to a more sophisticated problem setup and evaluation. The important metric is the computation time required to compute the voltage profiles as an intermediate result for all other analyses.

The power flow library PYPOWER takes around 9 hours to finish the computation for only one of the 35 combinations of vehicle number and charging power, and therefore would take around 13 days to complete the simulation. The runtimes of the Y_{BUS} Newton-Raphson and Z_{BUS} Jacobi methods are shown in table 6.7. The Z_{BUS} Jacobi method is clearly best suited for this task and is, without any optimizations, 60 times faster than an already quite optimized Y_{BUS} Newton-Raphson method.

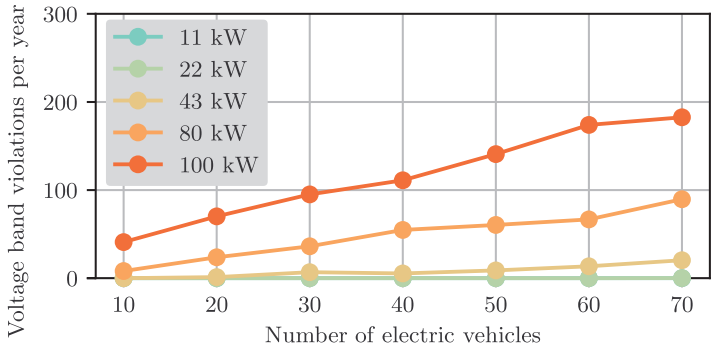


Figure 6.9: Number of minutes with voltage band violations for different numbers and charging powers of electric vehicles

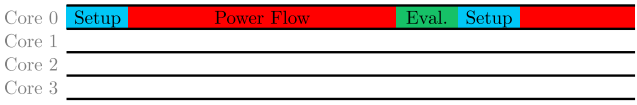
Table 6.7: Runtimes and relative runtime as compared to PYPOWER for 181,440,000 power flow simulations without any optimizations

PYPOWER	no optimizations	≈ 13d	100 %
Y_{BUS} Newton	no optimizations	≈ 33h	10.6 %
Z_{BUS} Jacobi	no optimizations	32m 06s	0.17 %

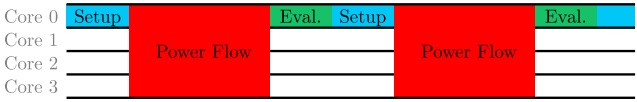
Parallelization

Unlike the comparatively small problem in the previous section 6.1, this problem can benefit from a parallelized execution. As shown in section 5.5, there are multiple ways to set up the parallelization. One possibility is a parallelization analog to section 5.5, where only the power flow computations themselves are distributed to multiple processes. This leaves the setup step (computation of the load profiles through the statistical processes) and result evaluation step (in this case the evaluation of voltage band violations) untouched. In order to parallelize these tasks as well, the entire computation including setup, power flow, and evaluation needs to be encapsulated in a single function which can be executed in a separate process. Figure 6.10 visualizes the three parallelization strategies when four processes can be executed in parallel.

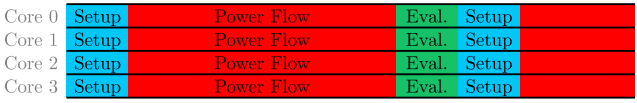
In practice, the latter strategy leads to the better overall execution time and cuts the runtime down to below 20 minutes on the 2-core, 4-thread Intel 7260U CPU, as shown in table 6.8.



(a) Execution in a single process



(b) Parallelization of the power flow function



(c) Parallelization of the entire evaluation

Figure 6.10: 3 different parallelization strategies for the specific task in this section, when four processes can be executed in parallel

Table 6.8: Runtimes and relative runtimes as compared to PYPOWER for 181,440,000 power flow simulations with parallelization

Z_{BUS} Jacobi	Parallelization in the power flow function (strategy b)	25m 48s	0.14 %
Z_{BUS} Jacobi	Parallelization of the entire evaluation (strategy c)	19m 41s	0.11 %

The simpler parallelization leads to a performance gain of around 20%, the more elaborate parallelization of the entire function leads to a gain of around 40 %.
Special care has to be taken to ensure a correct initialization of the processes, especially if they contain a pseudo-random generator. Without explicit seeding, the random state of the processes might be identical, so that all 'random' loads profiles end up the same.

Weak Load Detection

Since the simulation of the loads includes timesteps from all times of day, there are a lot of points in time where the voltage is in very safe bounds

and far from a voltage band violation. This makes this kind of simulation a prime candidate for weak load detection (see section 5.4). The goal of this technique is to find a minimal operation that can determine whether a set of loads will yield voltages that are easily within safe bounds.
In this case, the voltage $\underline{\mathbf{U}}$ can be assumed to be within safe bounds when the absolute power of the most loaded node is under a certain bound S_{cutoff}. Mathematically, this can expressed as

$$\|\underline{\mathbf{S}}\|_{max} < S_{\text{cutoff}} \quad \rightarrow \quad \underline{\mathbf{U}} = \underline{\mathbf{U}}_0, \tag{6.1}$$

where $S_{\text{cutoff}} \in \mathbb{R}$ is a chosen value which ensures that the assumption $\underline{\mathbf{U}} = \underline{\mathbf{U}}_0$ can be applied to the weakly loaded timesteps in a deterministic way which does not influence the statistical relevant parts of the raw results. By performing a trial run with a series of power flow computations, where the grid is progressively, but uniformly loaded with the same load at every node, a safe threshold S_{cutoff} can be found. In this low-voltage grid example, the influence of reactive power on this estimation is ignored, but in high-voltage grids, such an trial run needs to include an evaluation of the maximum reactive power as well. Figure 6.11 shows the results for this trial run for the example grid.

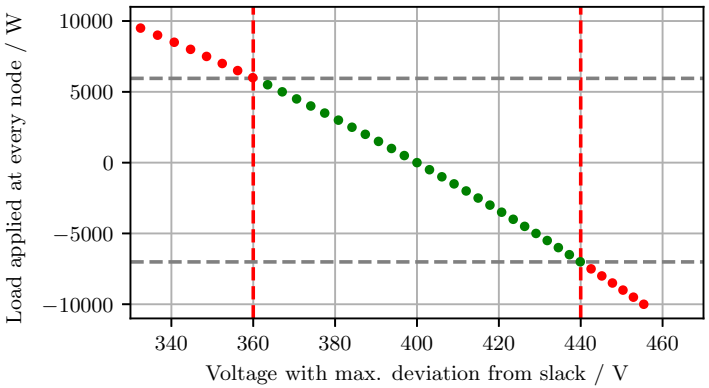


Figure 6.11: Result of a trial run for weak load detection, where the grid is progressively loaded to determine a safe S_{cutoff}

Every marker in Figure 6.11 represents one power flow computation and is located according to the load that applied to every node and the resulting voltage with the greatest deviation from the slack voltage.
The plot shows that an unsafe voltage below 360 V (0.9 pu) is reached

when a load of around 6 kW is applied to all nodes. Therefore, any set of loads where the absolute maximal load does not exceed 6 kW will not lead to a voltage below 360 V.
Critically, this criterion is extremely fast to check during the computation. A more complex criterion might be more effective, but an increased computation time quickly leads to a situation where a complete power flow would actually be faster.
The cirterion is then used to prune the loads before the power flow computations even start. In this case, around 45% of all load profiles can be immediately identified as safe and do not require an explicit power flow computation. The result is identical to Figure 6.9.

Table 6.9: Runtimes and relative runtimes as compared to PYPOWER for 181,440,000 power flow simulations with weak load detection and parallelization

PF method	Optimization	Runtime	
Z_{BUS} Jacobi	Weak Load Detection with $\|\underline{\mathbf{S}}\|_{max} < 6$ kW	27m 24s	0.15 %
Z_{BUS} Jacobi	Parallelization (strategy c) & Weak Load Detection with $\|\underline{\mathbf{S}}\|_{max} < 6$ kW	14m 39s	0.078 %

The application of weak load detection yields a moderate performance increase, as shown in table 6.9. The weak-load detection can easily be incorporated into the parallelized function, and the application of both optimization techniques increases the performance by around 55% compared to the non-optimized Z_{BUS} Jacobi method.
The example shows the optimization potential for a type of a large-scale simulation that is often required for statistical evaluations related to volatile and human behavior. In this example, with the right power flow method, an efficient implementation and some optimization techniques, the runtime can be cut from multiple hours or even days to less than 15 minutes. In practice, this especially helps with development iterations, which might require multiple complete simulations in order to dial in statistical variables.

6.3 n-2 Contingency Analysis in a High Voltage Grid

The concept of n-1 contingency analysis is widely used in electric grid planning and operations. The n-1 criterion states that an electric grid should continue to function within safe operating bounds if any one of the system components fails. Consequently, the n-2 criterion demands a stable operation in the case of two simultaneous failures. The n-1 criterion is routinely applied to transmission grids and high voltage distribution grids. The n-2 criterion is technically evaluated when the operation during a planned outage is analyzed, but it is rarely applied in normal operation for two unplanned outages due to the computational effort.
In this section, a basic n-2 contingency analysis is conducted for the lines of an example high voltage grid with 110 nodes. The grid is modeled after a German high-voltage distribution grid and is pictured in Figure 6.12.

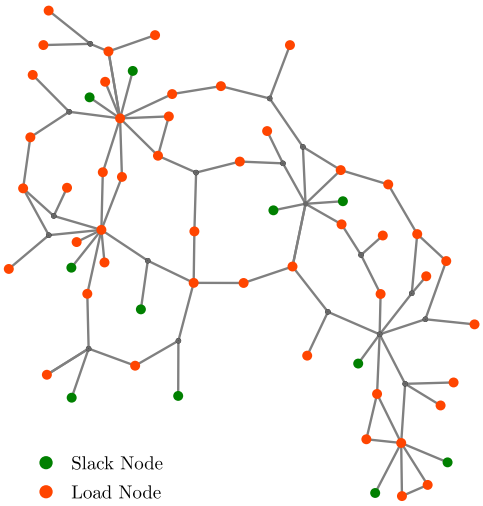


Figure 6.12: Topology of the example high voltage grid for the computation of the n-2 contingency with 110 nodes and 125 lines

The grid contains 11 slack nodes, which model the connections to the overlaying transmission grid. These slack nodes have different voltages

due to different tap changer settings, and the power flow computations therefore require the use of the adapted Y_{BUS} formulations and algorithms presented in sections 3.3 and 4.6.2.
Although some of the high-voltage lines are realized as parallel systems, all lines are modeled as single impedances and as fully impacted by an outage.
The computation considers only one load scenario. There are $l = 125$ lines, so the n-2 contingency analysis leads to $(l-1)^2/2+l = 7813$ different cases due to the (i, j)-(j, i)-symmetry of line outages. Crucially, each of these cases involves a slightly different grid topology. This means that the Y_{BUS} matrix needs to be modified for each case.
As a simple final result, Figure 6.13 shows the maximum current in the grid for each of the line outage combinations. The evaluation shows that there are three critical outage combinations (Line 50+81, 50+82, and 50+30). During a real, comprehensive contingency analysis there would likely be a more in-depth evaluation of the results taking into account the current limits of the individual lines, but that is usually not the performance-critical part.

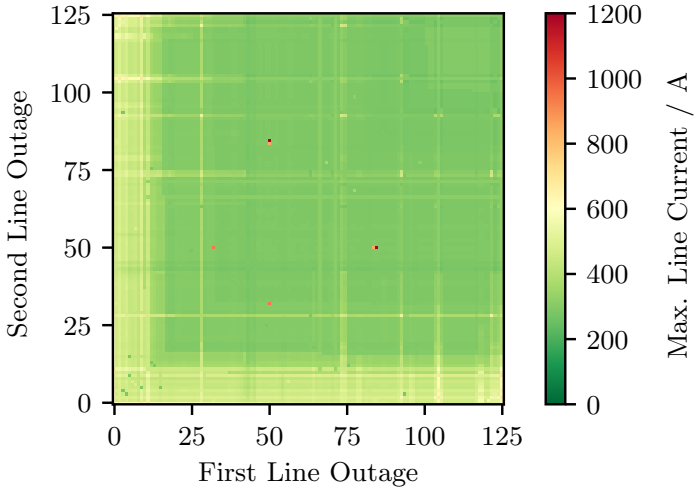


Figure 6.13: Heatmap of the maximal currents in grid for all n-2 line outages. Each outage case is plotted twice due to the symmetry around the diagonal

The runtimes of the power flow methods are presented in table 6.10.

Table 6.10: Runtimes and relative runtimes as compared to PYPOWER for the n-2 contingency analysis (7813 cases)

PYPOWER	no optimization	8m 02s	100 %
Y_{BUS} Jacobi	no optimization	27m 42s	345 %
Y_{BUS} Gauss-Seidel	Acceleration factor 1.2	30m 20s	378 %
Y_{BUS} Newton	no optimization	2m 21s	29.3 %
Z_{BUS} Jacobi	no optimization	5m 16s	65.6 %

The Y_{BUS} Jacobi method is the fastest Y_{BUS} fixed-point method, even if the optimal acceleration factor for the Y_{BUS} Gauss-Seidel method (around $\alpha = 1.2$ in this case) is found and set, but in the end, the Y_{BUS} fixed-point methods are still not competitive. The Y_{BUS} Newton-Raphson method is faster than the Z_{BUS} Jacobi method in this specific scenario. Because the admittance matrix $\underline{\mathbf{Y}}$ changes between every individual power flow calculation, the cost of the explicit matrix inversion $\underline{\mathbf{Z}} = \underline{\mathbf{Y}}^{-1}$ required in the Z_{BUS} Jacobi method incurs in every computation. This is an extreme scenario, and one of the few which does not favor the Z_{BUS} Jacobi method for overall computation time. If the computation includes more load scenarios or even an extensive time series, the Z_{BUS} Jacobi method is back to being the fastest.
PYPOWER does not support multiple slack nodes, so all but one slack node are modeled as PV nodes instead. Its runtime is 8m 02s, which is slightly more competitive compared to the other scenarios.
Among the optimization approaches, the weak load detection is not applicable, because there is only one load scenario. A grid reduction can also not be applicable, since there are no superfluous nodes in this complex topology. A sparse formulation also yields no performance increase. The only applicable optimization technique is parallelization.

Parallelization

In this contingency analysis, each outage simulation includes only one set of loads, and so only one power flow simulation is done for every unique grid. In order to gain any performance advantage from parallelization, the entire function of one outage simulation (generate grid with two lines removed, execute the power flow, evaluate the currents) has to be parallelized. This yields a 25 % speedup, as shown in table 6.11. The improvement is limited by the significant overhead of the parallelization,

including the separation of the data and copy operations to and from the different cores.

Table 6.11: Runtime and relative runtime as compared to PYPOWER for the n-2 contingency analysis (7813 cases) using parallelization

Y_{BUS} Newton	Parallelization to 2 cores	1m 45s	21.8 %

To summarize, in simulations where the grid topology or the impedances and therefore the admittance matrix $\underline{\mathbf{Y}}$ frequently changes, the Y_{BUS} Newton-Raphson method can be faster than the Z_{BUS} Jacobi method, because of the matrix inversion overhead for the latter. This might however be different if each outage case involves a time-series simulation. In that case, this task resembles the case in the previous section 6.2.

Chapter 7

Summary

This thesis has given an overview over the mathematical, numerical, and computational nature of the power flow problem, including the reasoning behind the physical and mathematical model, available solution methods, and their optimal formulation from a computing point of view. The examined solution methods were the widely applied Y_{BUS} Newton-Raphson, Y_{BUS} Gauss-Seidel and Backward-Forward Sweep methods along with the less popular Y_{BUS} Jacobi, Y_{BUS} Relaxation, and Z_{BUS} Jacobi methods. All methods were derived algorithmically, and the most computationally performant implementations were given. After a generic benchmark, the methods were applied to 3 example problems modeled after modern distribution grid planning approaches.
The key findings were as follows:

1. **An implementation of the solution methods based on the computational properties of modern CPUs can unlock significant performance gains.** Modern CPUs employ a range of increasingly complicated techniques to bridge the growing gap in speed between memory and execution units, like caches, pipelining, and branch-prediction. Using these techniques to its advantage, the Y_{BUS} Newton-Raphson method devised in this thesis is 3 - 10 times faster than the open-source solver PYPOWER, while being algorithmically identical.

2. **Most power flow methods belong to three distinct classes**, Y_{BUS} fixed-point methods, Y_{BUS} Newton-Raphson methods, and

the Z_{BUS} Jacobi method. Among the Y_{BUS} fixed-point methods, the Y_{BUS} Jacobi method is more performant than the more popular Y_{BUS} Gauss-Seidel method. The Y_{BUS} Newton-Raphson method is currently the most popular method and can be effectively combined with model simplifications to form 'decoupled' methods, which are effective for transmission grids. The Z_{BUS} Jacobi method is an old and often neglected method in its original form, but Backward-Forward-Sweep methods, which are a special case of the Y_{BUS} Jacobi method, are popular for use with single-feeder distribution grids. Many other seemingly methods can be traced back to one of those classes.

3. **Popular acceleration approaches like sparse matrices and parallelization can actually be detrimental to performance.** This again speaks to the raw performance of memory-contiguous numerical computations. Sparse matrices can e.g. severely impede cache performance, and parallelization can lead to an unrecoverable overhead. The smaller the grid and the computation problem, the more probable it is that an unmodified power flow method is actually fastest. In sufficiently large-scale problems however, these and other acceleration approaches like weak-load detection and grid reduction can have a great impact. In the presented examples, these techniques could speed up the computation by up to a factor of 1000 without any loss in precision.

4. **For most practical distribution grid planning problems, the Z_{BUS} Jacobi method is by far the fastest power flow method.** Algorithmically, it is on par with the currently dominating Y_{BUS} Newton-Raphson method, but computationally is much more efficient, as it employs fewer variables and simpler operations, leading to better memory and cache utilization as well as pipeline behaviour, which are critical to performance on modern CPUs. In the presented large-scale example problems, the computationally optimized Z_{BUS} Jacobi method is 60 to 150 times faster. The aforementioned Y_{BUS} fixed-point methods are not competitive at all. The only instances where a Y_{BUS} Newton-Raphson method might be faster are scenarios with changes in the grid impedances after every computation, as shown with the purpose-built example in section 6.3. One obstacle for the widespread adoption of the Z_{BUS} Jacobi method is the lacking experience with more complex

grid elements and component models. To that end, the thesis lists an existing, but seemingly unused method to incorporate generator nodes and contributes an extension to the method for grids with multiple slack nodes.

In summary, the results show that there is a lot of potential in computationally optimized power flow algorithms and customizable optimization approaches especially for large-scale power flow problems involving many load scenarios, which are typical for modern distribution grid planning approaches. With the options laid out above, the runtime can often be cut down by orders of magnitude, without resorting to probabilistic methods or major simplifications.

Bibliography

[1] Ahmad Abdel-Majeed. “Three-Phase State Estimation for Low-Voltage Grid”. PhD thesis. Universität Stuttgart, 2016.

[2] V. Ajjarapu and C. Christy. “The continuation power flow: A tool for steady state voltage stability analysis”. In: *IEEE Transactions on Power Systems* 7.1 (1992), pp. 416–423. DOI: 10.1109/59.141737. URL: https://ieeexplore.ieee.org/abstract/document/141737.

[3] R. N. Allan, B. Borkowska, and C. H. Grigg. “Probabilistic analysis of power flows”. In: *Proceedings of the Institution of Electrical Engineers* 121.12 (1974), pp. 1551–1556. ISSN: 00203270. DOI: 10.1049/piee.1974.0320. URL: http://digital-library.theiet.org/content/journals/10.1049/piee.1974.0320.

[4] R.N. Allan et al. “Probabilistic power-flow techniques extended and applied to operational decision making”. In: *Proceedings of the Institution of Electrical Engineers* 123.12 (1976), pp. 1317–1324. ISSN: 00203270. DOI: 10.1049/piee.1976.0264. URL: http://digital-library.theiet.org/content/journals/10.1049/piee.1976.0264.

[5] AMD. *AMD Optimizing CPU Libraries*. Oct. 22, 2021. URL: https://developer.amd.com/amd-aocl/.

[6] Richard V. Andre. *Programming the IBM 650 Magnetic Drum Computer and Data-Processing Machine*. 1958. URL: http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/650/Andree_Programming_the_IBM_650_Magnetic_Drum_Computer_and_Data-Processing_Machine_1958.pdf.

[7] K. Balamurugan and Dipti Srinivasan. “Review of power flow studies on distribution network with distributed generation”. In: *2011 IEEE Ninth International Conference on Power Electronics and Drive Systems*. 2011, pp. 411–417. DOI: 10.1109/PEDS.2011.6147281.

[8] Stefan G. Berg. *Cache Prefetching*. Tech. rep. 2002. URL: http://embedded.cs.uni-sb.de/literature/CachePrefetching.pdf.

[9] Susan Blackford and Jack Dongorra. *LAPACK Working Note 41: Installation Guide for LAPACK*. Tech. rep. Department of Computer Science, University of Tennessee, Knoxville, Tennessee, 1999. URL: http://www.netlib.org/lapack/lawnspdf/lawn41.pdf (visited on 09/21/2021).

[10] A. Brameller and J. K. Denmead. “Some improved methods for digital network analysis”. In: *Proceedings of the IEE - Part A: Power Engineering* 109.43 (1962), pp. 109–116. ISSN: 03698882. DOI: 10.1049/pi-a.1962.0078. URL: http://digital-library.theiet.org/content/journals/10.1049/pi-a.1962.0078.

[11] L.M.C. Braz, C.A. Castro, and C.A.F. Murati. “A critical evaluation of step size optimization based load flow methods”. In: *IEEE Transactions on Power Systems* 15.1 (2000), pp. 202–207. DOI: 10.1109/59.852122.

[12] H. E. Brown et al. “Power Flow Solution by Impedance Matrix Iterative Method”. In: *IEEE Transactions on Power Apparatus and Systems* 82.65 (1963), pp. 1–10. ISSN: 00189510. DOI: 10.1109/TPAS.1963.291392.

[13] Homer E. Brown et al. “Z-Matrix Algorithms in Load-Flow Programs”. In: *IEEE Transactions on Power Apparatus and Systems* PAS-87.3 (1967), pp. 807–814. ISSN: 00189510. DOI: 10.1109/TPAS.1968.292196.

[14] Rodney J. Brown and William F. Tinney. “Digital Solutions for Large Power Networks”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 76.June (Apr. 1957), pp. 347–351. ISSN: 00972460. DOI: 10.1109/AIEEPAS.1957.4499563. URL: http://ieeexplore.ieee.org/document/4499563/.

[15] Charles Byrne. *Applied Iterative Methods*. 2007. DOI: 10.1201/b10651.

[16] D. P. Chassin, K. Schneider, and C. Gerkensmeyer. "GridLAB-D: An open-source power systems modeling and simulation environment". In: *2008 IEEE/PES Transmission and Distribution Conference and Exposition*. 2008, pp. 1–5. DOI: 10.1109/TDC.2008.4517260.

[17] T.-H. Chen et al. "Distribution system short circuit analysis-A rigid approach". In: *IEEE Transactions on Power Systems* 7.1 (1992), pp. 444–450. DOI: 10.1109/59.141741.

[18] Cigré Working Group C4.605. *Modelling and Aggregation of Loads in Flexible Power Networks*. Tech. rep. February. CIGRE, 2014, p. 190. URL: https://e-cigre.org/publication/566-modelling-and-aggregation-of-loads-in-flexible-power-networks.

[19] T. Cui et al. "Accelerated AC contingency calculation on commodity multi-core SIMD CPUs". In: *2014 IEEE PES General Meeting, Conference Exposition*. 2014, pp. 1–5. DOI: 10.1109/PESGM.2014.6939078.

[20] J.C. Das. *Power System Analysis. Short-Circuit Load Flow and Harmonics*. CRC Press, 2012. ISBN: 978-1-4398-2080-3.

[21] Erhan Demirok et al. "Three-Phase Unbalanced Load Flow Tool for Distribution Networks". In: *Proceedings of the 2nd International Workshop on Integration of Solar Power Systems*. 2012. ISBN: 978-3-9813870-6-3. URL: http://vbn.aau.dk/files/76360540/SIW12_69.pdf.

[22] Hermann Dommel and William Tinney. "Optimal Power Flow Solutions". In: *IEEE Transactions on Power Apparatus and Systems* PAS-87.10 (Oct. 1968), pp. 1866–1876. ISSN: 00189510. DOI: 10.1109/TPAS.1968.292150. URL: http://ieeexplore.ieee.org/document/4073461/.

[23] Ulrich Drepper. *What every programmer should know about memory*. Tech. rep. Red Hat Inc., 2007, pp. 1–114. URL: https://people.freebsd.org/~lstewart/articles/cpumemory.pdf.

[24] Anamika Dubey et al. "Determining Time-of-Use Schedules for Electric Vehicle Loads: A Practical Perspective". In: *IEEE Power and Energy Technology Systems Journal* 2.1 (2015), pp. 12–20. ISSN: 23327707. DOI: 10.1109/jpets.2015.2405069.

[25] Roger C. Dugan and Thomas E. McDermott. "An open source platform for collaborating on smart grid research". In: *2011 IEEE Power and Energy Society General Meeting*. 2011, pp. 1–7. DOI: 10.1109/PES.2011.6039829.

[26] Roger C. Dugan and Davis Montenegro. *Reference Guide - The Open Distribution System Simulator*. Tech. rep. EPRI, 2020. URL: https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Distrib/Doc/OpenDSSManual.pdf (visited on 07/25/2020).

[27] L. A. Dunstan. "Digital Load Flow Studies". In: *Transactions of the American Institute of Electrical Engineers* 73.1 (Jan. 1954), pp. 825–832. ISSN: 00972460. DOI: 10.1109/AIEEPAS.1954.4498891. URL: http://ieeexplore.ieee.org/document/4498891/.

[28] L. A. Dunstan. "Machine Computing of Networks". In: *Electrical Engineering* 66 (1947), pp. 901–906. DOI: 10.1109/EE.1947.6443719. URL: https://ieeexplore.ieee.org/document/6443719/.

[29] L. A. Dunstan. "The General Solution Method of Power Network Analysis". In: *Transactions of the American Institute of Electrical Engineers* 67 (1948), pp. 631–639. ISSN: 00963860. DOI: 10.1109/T-AIEE.1948.5059722.

[30] Simon Eilenberger et al. "Probabilistic simulation for LV-grid optimization with new network components". In: *22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013)*. 2013, pp. 1–5. DOI: 10.1049/cp.2013.0813.

[31] U. Eminoglu and M. H. Hocaoglu. "Distribution systems forward/backward sweep-based power flow algorithms: A review and comparison study". In: *Electric Power Components and Systems* 37.1 (2009), pp. 91–110. ISSN: 15325008. DOI: 10.1080/15325000802322046.

[32] Entso-e. *Phase Shift Transformers Modelling*. Tech. rep. Brussels: ENTSO-E, 2014, pp. 1–27. URL: https://eepublicdownloads.entsoe.eu/clean-documents/CIM_documents/Grid_Model_CIM/ENTSOE_CGMES_v2.4_28May2014_PSTmodelling.pdf.

[33] Albert Fazio. “Future directions of non-volatile memory in compute applications”. In: *2009 IEEE International Electron Devices Meeting (IEDM)*. 2009, pp. 1–4. DOI: 10.1109/IEDM.2009.5424257. URL: https://ieeexplore.ieee.org/document/5424257.

[34] A. J. Flueck and H.-D. Chiang. “Solving the nonlinear power flow equations with an inexact Newton method using GMRES”. In: *IEEE Transactions on Power Systems* 13.2 (1998), pp. 267–273. DOI: 10.1109/59.667330. URL: https://ieeexplore.ieee.org/document/667330/authors%7B%5C#%7Dauthors.

[35] Agner Fog. *4. Instruction tables*. Tech. rep. Technical University of Denmark, 2021. URL: https://www.agner.org/optimize/instruction_tables.pdf (visited on 10/21/2021).

[36] Agner Fog. *The microarchitecture of Intel and AMD CPU's*. Tech. rep. Technical University of Denmark, 2020. URL: https://www.agner.org/optimize/microarchitecture.pdf.

[37] S. Ghosh and D. Das. “Method for load-flow solution of radial distribution networks”. In: *IEE Proc.-Gener. Transm. Distrib.* Vol. 146. 6. 1999. DOI: 10.1049/ip-gtd:19990464.

[38] Smarajit Ghosh and Karma Sonam Sherpa. “An Efficient Method for Load–Flow Solution of Radial Distribution Networks”. In: *International Journal of Electrical and Computer Engineering* 2 (2008), pp. 2094–2101.

[39] A. F. Glimn and G. W. Stagg. “Automatic Calculation of Load Flows”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 76.3 (Apr. 1957), pp. 817–825. ISSN: 0097-2460. DOI: 10.1109/AIEEPAS.1957.4499665. URL: http://ieeexplore.ieee.org/document/4499665/.

[40] Masoud Aliakbar Golkar. “A Novel Method for Load Flow Analysis of Unbalanced Three-Phase Radial Distribution Networks”. In: *Turk J Elec Engin* 15.3 (2007). URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.9476&rep=rep1&type=pdf.

[41] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore: The John Hopkins University Press, 1996. ISBN: 0-8018-5414-8.

[42] Abdellatif Hamouda and Khaled Zehar. "Improved algorithm for radial distribution networks load flow solution". In: *International Journal of Electrical Power & Energy Systems* 33.3 (Mar. 2011), pp. 508–514. ISSN: 01420615. DOI: 10.1016/J.IJEPES.2010.11.004. URL: https://www.sciencedirect.com/science/article/abs/pii/S0142061510002036.

[43] R.J. Hanson, F.T. Krogh, and C.L. Lawson. *A Proposal for Standard Linear Algebra Subprograms*. Tech. rep. Jet Propulsion Library, California Institue of Technology, Pasadena, California, 1973. URL: https://ntrs.nasa.gov/citations/19740005175 (visited on 10/21/2021).

[44] H. L. Hazen, O. R. Schurig, and M. F. Gardner. "The M. I. T. Network Analyzer: Design and Application to Power System Problems". In: *Transactions of the American Institute of Electrical Engineers* 49.3 (1930), pp. 1102–1113.

[45] J. M. Henderson. "Automatic Digital Computer Solution of Load Flow Studies". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 73.2 (1955), pp. 1696–1702. ISSN: 0097-2460. DOI: 10.1109/AIEEPAS.1954.4499023.

[46] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach, Fifth Edition*. Morgan Kaufmann, 2012. ISBN: 978-0-12811-905-1.

[47] Raphael Hunger. *Floating Point Operations in Matrix-Vector Calculus*. Tech. rep. Technische Universität München, 2007. URL: https://mediatum.ub.tum.de/doc/625604/625604 (visited on 09/21/2021).

[48] IBM. *IBM Archives: 650 Feeds and speeds*. 2000. URL: http://www-03.ibm.com/ibm/history/exhibits/650/650_fs1.html.

[49] *Graphical Symbols for Diagrams*. Standard. Geneva, CH: International Electrotechnical Commission, 2012. URL: https://webstore.iec.ch/publication/2723.

[50] *IEEE PES Distribution System Analysis Subcommitee Radial Test Feeder*. 2020. URL: https://site.ieee.org/pes-testfeeders/ (visited on 12/23/2020).

[51] IEEE Task Force on Load Representation for Dynamic Performance. "Load Representation For Dynamic Performance Analysis". In: *IEEE Transactions on Power Systems* 8.2 (1993), pp. 472–482. DOI: 10.1109/59.260837. URL: https://ieeexplore.ieee.org/document/260837.

[52] Intel. "Intel 64 and IA-32 Architectures Optimization Reference Manual". In: *Intel Technology Journal* (2016). ISSN: 15222594. DOI: 10.1535/itj.0903.05. URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf.

[53] Intel. *Math Kernel Library*. Dec. 24, 2020. URL: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html.

[54] *Intel oneAPI Math Kernel Library*. Intel. 2021. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/onemkl-developerreference-c.pdf.

[55] S. Iwamoto and Y. Tamura. "A Load Flow Calculation Method for Ill-Conditioned Power Systems". In: *IEEE Transactions on Power Apparatus and Systems* PAS-100.4 (1981), pp. 1736–1743. DOI: 10.1109/TPAS.1981.316511.

[56] Philip D. Jennings and George E. Quinan. "The Use of Business Machines in Determining the Distribution of in Power Line Networks Load and Reactive Components". In: *Electrical Engineering* 65.12 (1946), pp. 1045–1046. DOI: 10.1109/EE.1946.6440026.

[57] R. H. Jordan. "Rapidly Converging Digital Load Flow". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 76.3 (Apr. 1957), pp. 1433–1438. ISSN: 0097-2460. DOI: 10.1109/AIEEPAS.1957.4499813. URL: http://ieeexplore.ieee.org/document/4499813/.

[58] W. Kahan. *A Logarithm Too Clever by Half*. 2004. URL: https://people.eecs.berkeley.edu/~wkahan/LOG10HAF.TXT (visited on 10/20/2020).

[59] M.Z. Kamh and R. Iravani. "A Sequence Frame-Based Distributed Slack Bus Model for Energy Management of Active Distribution Networks". In: *IEEE Transactions on Smart Grid* 3.2 (2012), pp. 828–836. DOI: 10.1109/TSG.2012.2188915. URL: https://ieeexplore.ieee.org/abstract/document/6193195.

[60] Edward W. Kimbark, James H. Starr, and James E. van Ness. "A Compact, Inexpensive A-C Network Analyzer". In: *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 71.1 (1952), pp. 122–128. DOI: 10.1109/TCE.1952.6371897. URL: https://ieeexplore.ieee.org/document/6371897/.

[61] *LAPACK Website*. 2020. URL: http://www.netlib.org/lapack/ (visited on 01/13/2020).

[62] Robert Larson, William Tinney, and John Peschon. "State Estimation in Power Systems Part I: Theory and Feasibility". In: *IEEE Transactions on Power Apparatus and Systems* PAS-89.3 (1970), pp. 345–352. ISSN: 0018-9510. DOI: 10.1109/TPAS.1970.292711. URL: http://ieeexplore.ieee.org/document/4074060/.

[63] M.A. Laughton and M.W. Humphrey Davies. "Numerical techniques in solution of power-system load-flow problems". In: *Proceedings of the Institution of Electrical Engineers* 111.9 (1964), pp. 1575–1588. ISSN: 00203270. DOI: 10.1049/piee.1964.0259. URL: https://ieeexplore.ieee.org/document/5248026.

[64] M.A. Laughton et al. "Discussion on Investigation of the load-flow problem and Bootstrap Gauss-Seidel load flow". In: *Proceedings of the Institution of Electrical Engineers* 117.2 (1970), pp. 397–400. DOI: 10.1049/piee.1970.0078. URL: https://ieeexplore.ieee.org/document/5249041.

[65] C. L. Lawson et al. "Basic Linear Algebra Subprograms for Fortran Usage". In: *ACM Trans. Math. Softw.* 5.3 (Sept. 1979), pp. 308–323. ISSN: 0098-3500. DOI: 10.1145/355841.355847. URL: https://doi.org/10.1145/355841.355847.

[66] H. Le Nguyen. "Newton-Raphson method in complex form [power system load flow analysis]". In: *IEEE Transactions on Power Systems* 12.3 (1997), pp. 1355–1359. DOI: 10.1109/59.630481.

[67] F. de Leon and A. Semlyen. "Iterative solvers in the Newton power flow problem: preconditioners, inexact solutions and partial Jacobian updates". In: *IEE Proceedings - Generation, Transmission and Distribution* 149.4 (2002), pp. 479–484. DOI: 10.1049/ip-gtd:20020172. URL: https://ieeexplore.ieee.org/document/1024195.

[68] David Levinthal. *Performance Analysis Guide for Intel® Core ™ i7 Processor and Intel® Xeon ™ 5500 processors*. Tech. rep. Intel Corporation, 2009, pp. 1–72. URL: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf.

[69] G.X. Luo and A. Semlyen. "Efficient load flow for large weakly meshed networks". In: *IEEE Transactions on Power Systems* 5.4 (1990), pp. 1309–1316. DOI: 10.1109/59.99382.

[70] Manuel Marin. "GPU-Enhanced Power Flow Analysis". PhD thesis. Université de Perpignan via Domitia, 2016. URL: https://hal.archives-ouvertes.fr/tel-01299182.

[71] Federico Milano. *Power System Modelling and Scripting*. Springer, 2010. ISBN: 9783642136689. DOI: 10.1007/978-3-642-13669-6. URL: https://link.springer.com/book/10.1007/978-3-642-13669-6.

[72] Jovica V. Milanović et al. "International industry practice on power system load modeling". In: *IEEE Transactions on Power Systems* 28.3 (2013), pp. 3038–3046. ISSN: 08858950. DOI: 10.1109/TPWRS.2012.2231969.

[73] Sivkumar Mishra and Debapriya Das. "Distribution System Load Flow Methods: A Review". In: *Icfai University Press (IUP) Journal of Electrical and Electronics Engineering* 1 (Apr. 2008), pp. 7–25.

[74] Vijay Laxmi Mishra, Manish Madhav, and R Bajpai. "Comparison of Different Techniques for Distribution System Load Flow Analysis-A Review". In: *International Journal for Scientific Research and Development* 3 (11 Jan. 2016), pp. 731–735. ISSN: 2321-0613.

[75] R. K. Moore. "A General Course in Traveling Waves". In: *IRE Transactions on Education* 3.1 (1960), pp. 15–19. ISSN: 0893-7141. DOI: 10.1109/TE.1960.4322115. URL: http://ieeexplore.ieee.org/document/4322115/.

[76] Ulrich Münz and Diego Romeres. "Region of Attraction of Power Systems". In: *IFAC Proceedings Volumes* 46.27 (2013). 4th IFAC Workshop on Distributed Estimation and Control in Networked Systems (2013), pp. 49–54. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20130925-2-DE-4044.00018. URL: https://www.sciencedirect.com/science/article/pii/S1474667015402071.

[77] Walter Murray, Tomas Tinoco De Rubira, and Adam Wigington. *Improving the robustness of Newton-based power flow methods to cope with poor initial points*. Tech. rep. Stanford University, CA, 2013. URL: https://web.stanford.edu/class/cme334/docs/2013-10-29-murray_newton_power_flow.pdf.

[78] Makwana Nirbhaykumar. "Methods for Load Flow Analysis of Weakly Meshed Distribution System". In: *International Journal of Scientific and Research Publications* 2.3 (2012). URL: http://www.ijsrp.org/research_paper_mar2012/ijsrp-Mar-2012-95.pdf.

[79] Dietrich Oeding and Bernd Rüdiger Oswald. *Elektrische Kraftwerke und Netze, 7. Auflage*. Springer-Verlag Berlin Heidelberg, 2011.

[80] A. Pandey et al. "Robust Power Flow and Three-Phase Power Flow Analyses". In: *IEEE Transactions on Power Systems* 34.1 (2019), pp. 616–626. DOI: 10.1109/TPWRS.2018.2863042. URL: https://ieeexplore.ieee.org/document/8424869.

[81] Konstantin Papaillou, ed. *Handbook of Power Systems*. Springer-Verlag Berlin Heidelberg, 2021.

[82] Ruud van der Pas. "Memory Hierarchy in Cache Based Systems". In: (2002). URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=81413C949E54E995E59A04B37F3D8A58?doi=10.1.1.137.7841%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[83] J. Peschon, D.W. Bree, and L.P. Hajdu. "Optimal power-flow solutions for power system planning". In: *Proceedings of the IEEE* 60.1 (1972), pp. 64–70. ISSN: 0018-9219. DOI: 10.1109/PROC.1972.8558. URL: http://ieeexplore.ieee.org/document/1450488/.

[84] Norris M. Peterson and W. Scott Meyer. "Automatic Adjustment of Transformer and Phase-Shifter Taps in the Newton Power Flow". In: *IEEE Transactions on Power Apparatus and Systems* PAS-90.1 (1971), pp. 103–108. ISSN: 00189510. DOI: 10.1109/TPAS.1971.292904.

[85] F. Pilo et al. *Planning and Optimization Methods for Active Distribution Systems Working Group C6.19*. Tech. rep. August. Cigre, 2014. URL: https://e-cigre.org/publication/ELT_276_7-planning-and-optimization-methods-for-active-distribution-systems.

[86] Sergio Pissanetzky. *Sparse Matrix Technology*. Academic Press, London, 1984. ISBN: 978-0-9762775-3-8.

[87] R. Podmore and J. Undrill. "Modified Nodal Iterative Load Flow Algorithm to Handle Series Capacitive Branches". In: *IEEE Transactions on Power Apparatus and Systems* PAS-92.4 (July 1973), pp. 1379–1387. ISSN: 00189510. DOI: 10.1109/TPAS.1973.293545. URL: http://ieeexplore.ieee.org/document/4075219/.

[88] R. Pritchard and C. Pottle. "High-Speed Power Flows Using Attached Scientific ("Array") Processors". In: *IEEE Transactions on Power Apparatus and Systems* PAS-101.1 (1982), pp. 249–253. DOI: 10.1109/TPAS.1982.317345.

[89] Alexander Probst. "Auswirkungen von Elektromobilität auf Energieversorgungsnetze analysiert auf Basis probabilistischer Netzplanung". PhD thesis. Universität Stuttgart, 2014. URL: https://www.ieh.uni-stuttgart.de/dokumente/dissertationen/Diss_Probst.pdf.

[90] D. Rajicic, R. Ackovski, and R. Taleski. "Voltage correction power flow". In: *IEEE Transactions on Power Delivery* 9.2 (1994), pp. 1056–1062. DOI: 10.1109/61.296308.

[91] Shruti Rao et al. "The Holomorphic Embedding Method Applied to the Power-Flow Problem". In: *IEEE Transactions on Power Systems* 31.5 (Sept. 2016), pp. 3816–3828. ISSN: 08858950. DOI: 10.1109/TPWRS.2015.2503423. URL: http://ieeexplore.ieee.org/document/7352383/.

[92] L. Rese, A.S. Costa, and A.S. Silva. "A modified load flow algorithm for microgrids operating in islanded mode". In: *IEEE PES Conference on Innovative Smart Grid Technologies (ISGT Latin America)*. 2013. DOI: 10.1109/ISGT-LA.2013.6554384. URL: https://ieeexplore.ieee.org/abstract/document/6554384.

[93] Walter Robertson. *How to measure FLOPS of a MatLab function*. 2017. URL: https://se.mathworks.com/matlabcentral/answers/172095-how-to-measure-flops-of-a-matlab-function (visited on 10/02/2021).

[94] Tomas Tinoco De Rubira. *Alternative methods for solving power flow problems*. Tech. rep. Stanford University, CA, 2012. URL: http://web.stanford.edu/class/cme334/docs/2013-10-29-rubira_power_flow.pdf.

[95] Nobuo Sato and W. F. Tinney. "Techniques for Exploiting the Sparsity of the Network Admittance Matrix". In: 82.69 (Dec. 1963), pp. 944–950. ISSN: 00189510. DOI: 10.1109/TPAS.1963.291477. URL: http://ieeexplore.ieee.org/document/4072891/.

[96] Patrick S. Sauter, Christian A. Braun, and Mathias Kluwe. "Comparison of the Holomorphic Embedding Load Flow Method with Established Power Flow Algorithms and a New Hybrid Approach". In: *2017 Ninth Annual IEEE Green Technologies Conference*. 2017. DOI: 10.1109/GreenTech.2017.36.

[97] K. P. Schneider et al. "Analytic Considerations and Design Basis for the IEEE Distribution Test Feeders". In: *IEEE Transactions on Power Systems* 33.3 (2018), pp. 3181–3188. ISSN: 08858950. DOI: 10.1109/TPWRS.2017.2760011. URL: https://www.osti.gov/pages/servlets/purl/1432185.

[98] D. Shirmohammadi et al. "A compensation-based power flow method for weakly meshed distribution and transmission networks". In: *IEEE Transactions on Power Systems* 3.2 (1988), pp. 753–762. DOI: 10.1109/59.192932.

[99] Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2008. ISBN: 978-1-84800-069-8. DOI: 10.1007/978-1-84800-070-4.

[100] John Smart, Warren Powell, and Stephen Schey. "Extended range electric vehicle driving and charging behavior observed early in the EV project". In: *SAE Technical Papers* 2 (2013). ISSN: 26883627. DOI: 10.4271/2013-01-1441.

[101] James E. Smith. "A Study of Branch Prediction Strategies". In: *Proceedings of the 8th Annual Symposium on Computer Architecture*. ISCA '81. Minneapolis, Minnesota, USA: IEEE Computer Society Press, 1981, pp. 135–148.

[102] Glenn W. Stagg and Ahmed H. El-Abiad. *Computer Methods In Power System Analysis*. McGraw-Hill Book Company, 1968, p. 427. ISBN: 0-07-085764-4.

[103] B. Stott and O. Alsac. "Fast Decoupled Load Flow". In: *IEEE Transactions on Power Apparatus and Systems* PAS-93.3 (1974), pp. 859–869. DOI: 10.1109/TPAS.1974.293985. URL: https://ieeexplore.ieee.org/abstract/document/4075431.

[104] Brian Stott. "Review of Load-Flow Calculation Methods". In: *Proceedings of the IEEE* 62.7 (1974), pp. 916–929. ISSN: 15582256. DOI: 10.1109/PROC.1974.9544. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1451474.

[105] Brian Stott, Jorge Jardim, and Ongun Alsac. "DC Power Flow Revisited". In: *IEEE Transactions on Power Systems* 24.3 (2009), pp. 1290–1300. DOI: 10.1109/TPWRS.2009.2021235.

[106] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems, Fourth Edition*. Pearson, 2015. ISBN: 978-0-13359-162-0.

[107] Jen-Hao Teng. "A direct approach for distribution system load flow solutions". In: *IEEE Transactions on Power Delivery* 18.3 (2003), pp. 882–887. DOI: 10.1109/TPWRD.2003.813818.

[108] L. Thurner et al. "pandapower — An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems". In: *IEEE Transactions on Power Systems* 33.6 (Nov. 2018), pp. 6510–6521. ISSN: 0885-8950. DOI: 10.1109/TPWRS.2018.2829021.

[109] William F. Tinney and J. W. Walker. "Direct solutions of sparse network equations by optimally ordered triangular factorization". In: *Proceedings of the IEEE* 55.11 (1967), pp. 1801–1809. ISSN: 0018-9219. DOI: 10.1109/PROC.1967.6011. URL: http://ieeexplore.ieee.org/document/1447941/.

[110] J. A. Treece. "Bootstrap Gauss-Seidel load flow". In: *Proceedings of the Institution of Electrical Engineers* 116.5 (1969), pp. 866–870. DOI: 10.1049/piee.1969.0161.

[111] A. Trias. "The Holomorphic Embedding Load Flow method". In: *2012 IEEE Power and Energy Society General Meeting*. IEEE, July 2012, pp. 1–8. ISBN: 978-1-4673-2729-9. DOI: 10.1109/PESGM.2012.6344759. URL: http://ieeexplore.ieee.org/document/6344759/.

[112] Antonio Trias. *Fundamentals of the Holomorphic Embedding Load-Flow Method*. Tech. rep. Sant Cugat del Valles: Aplicaciones en Informatica Avanzada, 2015. arXiv: 1509.02421v1. URL: https://arxiv.org/pdf/1509.02421.pdf.

[113] S.C. Tripathy et al. "Load-Flow Solutions for Ill-Conditioned Power Systems by a Newton-Like Method". In: *IEEE Transactions on Power Apparatus and Systems* PAS-101.10 (1982), pp. 3648–3657. DOI: 10.1109/TPAS.1982.317050.

[114] Andrew Urquhart and Murray Thomson. "Resolving Inconsistencies In Three-Phase Current Measurements". In: *Cired 2017*. June. 2017, pp. 12–15.

[115] James E. Van Ness. "Iteration Methods for Digital Load Flow Studies". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 78.3 (Apr. 1959), pp. 583–586. ISSN: 00972460. DOI: 10.1109/AIEEPAS.1959.4500383. URL: http://ieeexplore.ieee.org/document/4500383/.

[116] Stephen Vavasis et al. *Exchanges in NA Digest regarding flops*. 2000. URL: http://www.stat.uchicago.edu/~lekheng/courses/309f14/flops/vmdd.html (visited on 10/02/2021).

[117] Lucian N. Vintan. "Neural Branch Prediction: From the First Ideas, to Implementations in Advanced Microprocessors and Medical Applications". In: *Proceedings of the Romanian Academy, Series A*. Vol. 20. 2/2019. 2019, pp. 205–212. URL: https://acad.ro/sectii2002/proceedings/doc2019-2/12-Vintan.pdf.

[118] Gerhard Walker. "Impact and Chances of Electric Mobility for the German Low Voltage Distribution Grids". PhD thesis. Universität Stuttgart, 2018.

[119] Y. Wang et al. "Analysis of ill-conditioned power-flow problems using voltage stability method". In: *IEE Proc.-Gener. Transm. Distrib.* 148.5 (2001), pp. 384–390. DOI: 10.1149/ip-gtd:20010424.

[120] J. B. Ward and H. W. Hale. "Digital Computer Solution of Power-Flow Problems". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 75.3 (1956), pp. 398–404. ISSN: 00972460. DOI: 10.1109/AIEEPAS.1956.4499318.

[121] Pascal Wiest. "Probabilistische Verteilnetzplanung zur optimierten Integration flexibler dezentraler Erzeuger und Verbraucher". PhD thesis. Universität Stuttgart, 2018. URL: https://www.ieh.uni-stuttgart.de/dokumente/dissertationen/2018_Dissertation_Wiest.pdf.

[122] Virginia Vassilevska Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Proceedings of the 44th symposium on Theory of Computing - STOC '12*. New York, New York, USA: ACM Press, 2012, pp. 887–898. ISBN: 9781450312455. DOI: 10.1145/2213977.2214056. URL: https://dl.acm.org/doi/10.1145/2213977.2214056.

[123] Chao Zhai and Hung D. Nguyen. "Region of Attraction for Power Systems using Gaussian Process and Converse Lyapunov Function - Part 1: Theoretical Framework and Off-line Study". In: *CoRR* abs/1906.03590 (2019). arXiv: 1906.03590. URL: http://arxiv.org/abs/1906.03590.

[124] Fan Zhang and C.S. Cheng. "A modified Newton method for radial distribution system power flow analysis". In: *IEEE Transactions on Power Systems* 12.1 (1997), pp. 389–397. DOI: 10.1109/59.575728.

[125] Ray D. Zimmerman. *AC Power Flows, Generalized OPF Costs and their Derivatives using Complex Matrix Notation, Rev. 4*. Tech. rep. Power Systems Engineering Research Center (Pserc), 2011, p. 25. DOI: 10.13140/2.1.2731.2327. URL: http://www.pserc.cornell.edu/matpower/TN2-OPF-Derivatives.pdf.

[126] Ray Daniel Zimmerman. "Comprehensive distribution power flow modeling, formulation, solution algorithm and analysis". PhD thesis. Cornell University, 1995, p. 200. URL: https://dl.acm.org/doi/10.5555/269740.

Appendix A

Operation Counts of BLAS, LAPACK, and MKL routines

This appendix chapter outlines the operation counts for the routines used in the algorithms 1, 2, 3, 5, and 6 and the tables 4.1, 4.2, 4.3, 4.4, and 4.5 in a central place.
Matrix and vector algebra operations usually require multiplications and additions, which count as individual floating point operations (FLOPs). On a hardware level, floating-point subtractions are a special case of additions. The real-world throughput of all those operations are however not similar. For instance, on the Intel Skylake architecture, a floating point addition with operands sitting in a register takes 3 cycles, a multiplication takes 5 cycles, and a division takes $14 - 16$ cycles [35].
Since most of the computations are conducted with complex numbers, the inner complexity of basic operations with complex numbers must be taken into account. Table A.1 shows the OPs and resulting theoretical FLOP count required. In order to avoid confusion, the complex numbers are shown as ordered pairs.

Table A.1: FLOP counts of basic complex operations

Name	Math. Operation	Transformed to real	OPs	FLOPs
Complex Addition	$(a, jb) + (c, jd)$	$(a + c, j(b + d))$	2 Add.	2
Complex Subtraction	$(a, jb) - (c, jd)$	$(a - c, j(b - d))$	2 Add.	2
Complex Multiplication	$(a, jb) \cdot (c, jd)$	$(ac - bd, j(bc + ad))$	4 Mult., 2 Add.	6
Complex Division	$(a, jb) \,/\, (c, jd)$	$\left(\frac{ac-bd}{c^2+d^2}, j\frac{(bc+ad)}{c^2+d^2}\right)$	4 Mult., 2 Div., 2 Squares, 3 Add.	11

Table A.2 shows the BLAS, LAPACK, and MKL VM routines (OPs) used in this thesis together with their theoretical floating point operation count (FLOPs). The FLOPs data is derived from [9], [47], and [54].

Table A.2: Operation Counts of BLAS, LAPACK, and MKL routines used throughout the thesis

Interface	OP	Description	FLOPs
BLAS Lvl. 2	`zgemv`	Complex matrix-vector multiplication	$12n^2$
BLAS Lvl. 3	`zgemm`	Complex matrix-matrix multiplication	$12n^3$
MKL VM	`vzAdd`	Complex elementwise vector-vector addition	$2n$
	`vzMul`	Complex elementwise vector-vector multiplication	$6n$
	`vzDiv`	Complex elementwise vector-vector division	$11n$
	`vzExp`	Complex elementwise vector exponent	-
	`vzAbs`	Absolute value of complex vector	$3n$
	`zomatadd`	Complex elementwise matrix-matrix addition	$2n^2$
	`zimatcopy`	Complex conjugate of matrix	n^2
	`izamax`	Find index of max. abs. element of vector	$2n$
LAPACK	`zlacgv`	Complex conjugate of a vector	n
	`zscal`	Scale complex vector with constant	$2n$
	`dgesv`	Solution of linear system $\mathbf{Ax} = \mathbf{b}$	$8/3n^3 + 7n^2 + 7/3n$
	`zgetrf`	Complex LU Factorization	$8/3n^3 - n^2 + 13/3n$
	`zgetri`	Complex Matrix Inverse from LU	$16/3n^3 - n^2 + 20/3n$

On modern systems (roughly Intel Pentium I and later), floating point operation counts are no substitute for benchmarks. Real-world computational performance is heavily influenced by memory and cache usage, branch prediction, and other CPU functionalities. However, the disparity between the theoretical algorithmic complexity and real-world performance allows for conclusions about the utilization of those functionalities.

Appendix B

Benchmarks of BLAS and LAPACK Operations

The dominating numerical operations in the power flow methods are complex matrix-vector multiplications (`zgemm()`) for Y_{BUS} Jacobi and Z_{BUS} Jacobi methods and the solution of a numerical system (`dgesv()`) for the Y_{BUS} Newton-Raphson method, respectively. The Z_{BUS} Jacobi method requires an additional matrix inversion (`zgetrf(); zgetri()`). Figure B.1 shows the timings for these raw operations with the system and software stack outlined in section 2.4.
The three operations are executed for matrix sizes from $n = 5$ to $n = 5000$, representing similar node counts. The matrices were derived from randomly generated, meshed high voltage grids. All operations were repeated at least 100 times for the benchmark, after an initial warmup run. The solution of the linear system is benchmarked with the resulting real-valued Jacobi matrix of order $2n$. The runtimes of the operations ranges from around 1 µs for a complex matrix-vector multiplication with $n = 5$ to 14 seconds for a complex matrix inversion with $n = 5000$. Over the entire range of grid sizes, the complex matrix multiplication is roughly $10 - 20$ times faster than the solution of the linear system. In turn, the solution of the linear system is as fast as the matrix inversion for matrix sizes up to around $n = 100$, after which the inversion takes about twice as long up to $n = 5000$.
The results indicate lower bounds for the runtime of the three power flow methods, specifically the Y_{BUS} Newton-Raphson and Z_{BUS} Jacobi methods. After the complex matrix inversion, the Z_{BUS} Jacobi method executes

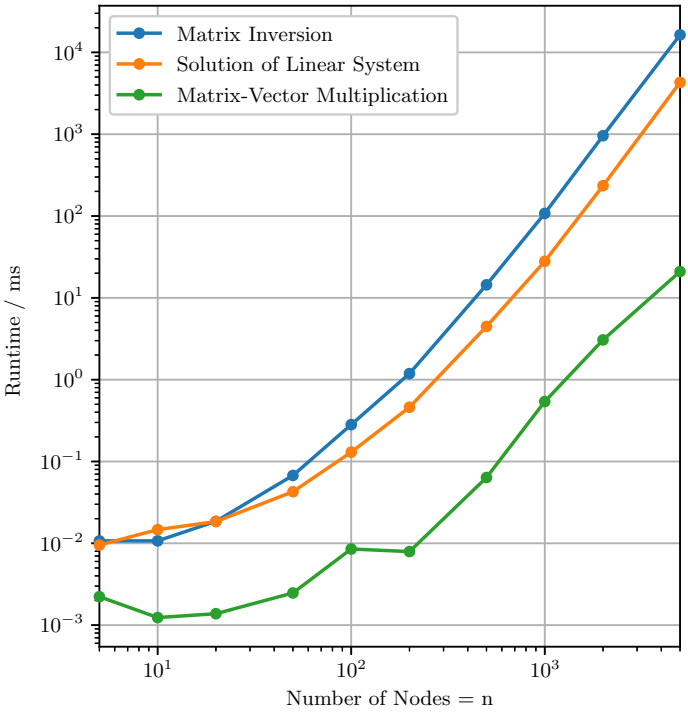


Figure B.1: Raw Individual Timings of Solution of Linear System, Matrix Inversion, and Matrix-Vector Multiplication with varying matrix sizes

mostly matrix-vector multiplications, whereas the Y_{BUS} Newton-Raphson method requires the solution of a linear system in every iteration (unless an inexact Newton method is chosen). The more iterations are performed without a change in the grid, the greater the advantage of the Z_{BUS} Jacobi method is. According to these results, the Z_{BUS} Jacobi method can be up to 20 times faster when a high number of computations are performed in the same grid, as is the case in the example in section 6.2.
Of course, these benchmarks do not take into account sparse matrix formulations (see section 5.2) and the additional operation that occur during the solutions, especially during one iteration of the Y_{BUS} Newton-Raphson method (see 4.2).

Appendix C

Grid Data

C.1 Low-voltage grid from section 4.5

The generic low-voltage grid used for the performance comparison in section 4.5 was generated as a single-feeder grid with identical impedances between the nodes. The goal was to create a totally generic grid with realistic proportions and values, but no special features whatsoever. The nodes are connected by cables with cross-section area $3 \times 150\,\mathrm{mm}^2$ which are all 50 meters long, a common distance for low-voltage grids in rural villages. The electrical properties outlined in table C.1. The cable data is taken from `https://bruggcables.com/fileadmin/site/documents/Mittelspannung/Produktkatalog_MS_NS_DE.pdf#page=14`. The line model is shown in figure 3.3.

Table C.1: Electrical properties of the low-voltage cable

		per km	per 50m
Resistance	R	$0.240\,\Omega$	$0.012\,\Omega$
Reactance	X	$0.071\,\Omega$	$0.00355\,\Omega$
Capacitance	C	$0.174\,\mu\mathrm{F}$	$0.0087\,\mu\mathrm{F}$
Susceptance	B	$18.3\,\mathrm{k}\Omega$ @ $50\,\mathrm{Hz}$	$366\,\mathrm{k}\Omega$ @ $50\,\mathrm{Hz}$

The assumed power at the nodes is randomly generated with a Weibull distribution taken from [118], with parameters $\lambda = 1, \mathrm{k} = 1.7$ and a scaling factor of 4000 W. The pseudorandom number generator from Numpy is

an implementation of MT19927 32-bit with seed 4268376. All loads are assumed inductive with a cos ϕ of 0.95.

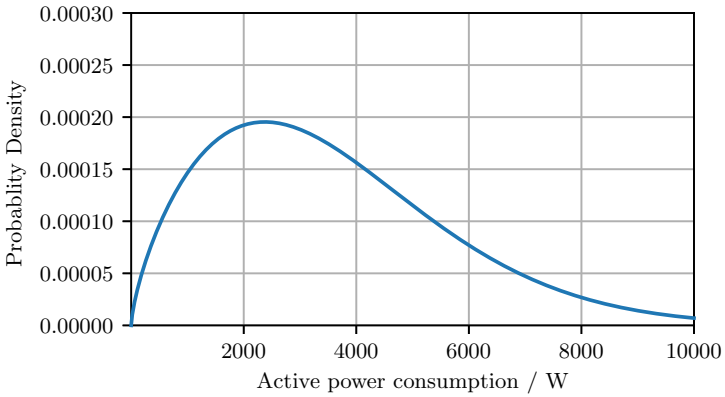


Figure C.1: Probability Density Function (PDF) of the Weibull distribution used for the performance measurements in section 4.5. Parameters are $\lambda = 1, \mathrm{k} = 1.7$, moments are `mean` $= 3569\,\mathrm{W}$, `variance` $= 4669677\,\mathrm{W}^2$, `skew` $= 0.865$, `kurtosis` $= 0.77$

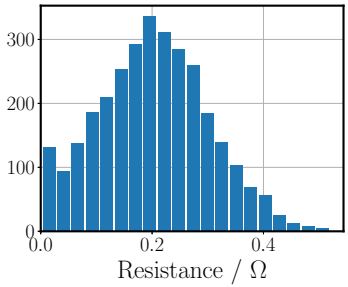
C.2 Medium voltage test grid from section 5.1.2

Table C.2 shows the properties of the medium voltage grid used to show the (lacking) potential of the Y_{BUS} Newton-Raphson method with line search in section 5.1.2. The assumed powers randomly generated following the same statistical function as shown in section C.1 and in figure C.1.

Table C.2: Overview of properties of the 3000-node medium voltage grid used in section 5.1.2

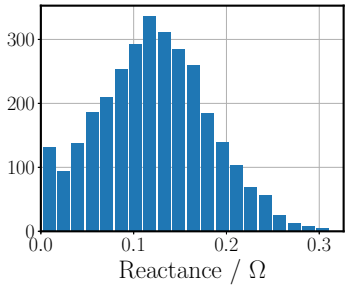
Name	Generic MV grid
Source	randomly generated
Number of Nodes	3000
Number of Lines	3099
Number of Slack Nodes	1
Types of Lines	100% Cables
Degree of Intermeshing	2.06
Nominal Voltage Level	10 kV
Slack Voltage	10 kV (Line-Line voltage)

Distribution of R



Min	0.002 Ω
Max	0.56 Ω
Mean	0.21 Ω
Var	0.009 Ω^2
Skew	0.14
Kurtosis	−0.2

Distribution of X



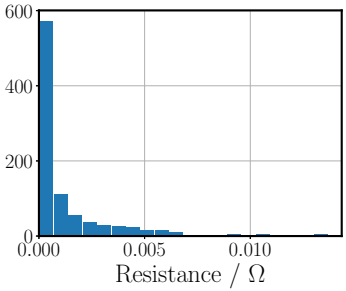
Min	0.0012 Ω
Max	0.34 Ω
Mean	0.12 Ω
Var	0.003 Ω^2
Skew	0.14
Kurtosis	−0.2

C.3 European Low Voltage Test Feeder

Table C.3: Overview of properties of the European Low-Voltage Test Feeder used in section 6.1

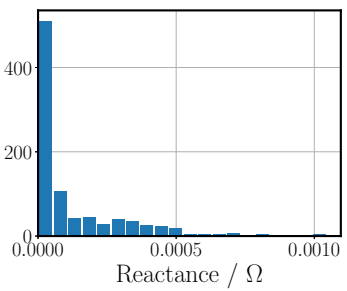
Name	European Low Voltage Test Feeder
Source	[50]
Location	Manchester, UK
Number of Nodes	906
Number of Lines	905
Number of Slack Nodes	1
Types of Lines	100% Cables
Nominal Voltage Level	416 V
Slack Voltage	252.2 V (Line-Earth voltage with tap changer set to 1.05 p.u.)

Distribution of R



Min	$6.18 \cdot 10^{-6}$ Ω
Max	0.014 Ω
Mean	0.0012 Ω
Variance	$4.06 \cdot 10^{-6}$ Ω^2
Skew	2.97
Kurtosis	10.9

Distribution of X



Min	$2.53 \cdot 10^{-6}$ Ω
Max	0.001 Ω
Mean	0.0001 Ω
Variance	$3.22 \cdot 10^{-8}$ Ω^2
Skew	2.12
Kurtosis	4.97

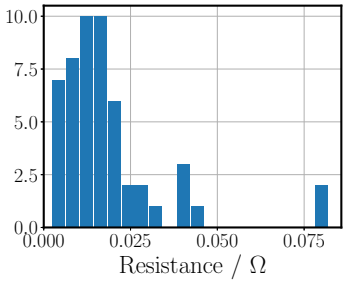
C.4 Low voltage grid from section 6.2

The low-voltage grid used in section 6.2 was derived from a real low-voltage grid in the south of Germany. It connects 40 private households with unknown load characteristics and 21 photovoltaic plants with a total installed capacity is 461.8 kW, making the entire grid a net producer of energy on days with high insolation. The grid topology is radial, the standard and desired topology for low-voltage grids in Germany. Most of the connections are cables, but parts of the grid still consist of overhead lines. Because all of the lines are short, the parallel susceptance B is negligible and was not part of the original dataset.

Table C.4: Overview of properties of the Low Voltage Test Grid used in section 6.1

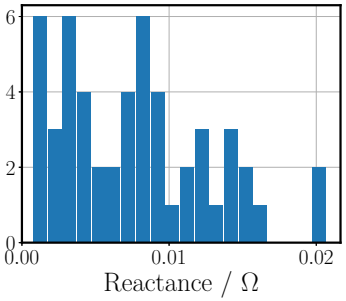
Name	Low Voltage Test Grid
Source	not public / shared by DGO
Location	South of Germany
Number of Nodes	53
Number of Lines	52
Number of Slack Nodes	1
Types of Lines	63.5 % overhead lines, 36.5% cables
Nominal Voltage Level	400 V line-line
Slack Voltage	230 V line-earth

Distribution of R



Min	0.0024 Ω
Max	0.082 Ω
Mean	0.019 Ω
Variance	0.00025 Ω^2
Skew	2.41
Kurtosis	6.48

Distribution of X



Min	0.00072 Ω
Max	0.02 Ω
Mean	0.0077 Ω
Variance	$2.55 \cdot 10^{-5}$ Ω^2
Skew	0.63
Kurtosis	−0.27

Table B.1: Node Data for low voltage grid in section 6.2

Node	Type	Inst. PV / kW
0	PQ	
1	PQ	
2	PQ	
3	PQ	
4	PQ	
5	PQ	
6	PQ	
7	PQ	
8	PQ	
9	PQ	
10	PQ	22.80
11	PQ	
12	PQ	6.97
13	PQ	
14	PQ	2.00
15	PQ	
16	PQ	58.00
17	PQ	
18	PQ	
19	PQ	12.40
20	PQ	12.40
21	PQ	8.40
22	PQ	
23	PQ	
24	PQ	
25	PQ	
26	PQ	
27	PQ	
28	PQ	
29	PQ	3.00
30	PQ	
31	PQ	
32	PQ	
33	PQ	
34	PQ	84.40
35	PQ	
36	PQ	29.00
37	PQ	
38	PQ	27.30
39	PQ	5.10
40	PQ	
41	PQ	4.40
42	PQ	29.60
43	PQ	46.60
44	PQ	27.40
45	PQ	9.90
46	PQ	15.87
47	Slack	
48	PQ	
49	PQ	28.06

Table B.2: Line Data for the low voltage grid in section 6.2

From	To	R/mΩ	X/mΩ
0	1	21.1529	2.7390
1	2	10.8999	7.7249
1	48	29.2119	20.7029
3	7	8.7199	6.1799
3	16	18.3119	12.9779
4	8	8.7599	0.7200
5	7	17.8759	12.6689
5	36	19.6199	13.9049
6	35	10.8999	7.7249
6	39	6.1039	4.3260
7	10	10.0980	5.4230
7	39	14.3879	10.1969
8	38	4.3600	3.0900
8	39	13.0799	9.2699
9	16	23.5999	6.8199
10	11	17.8199	9.5699
12	13	4.1580	2.2330
12	40	14.2559	7.6560
14	25	46.0719	8.5280
14	42	16.8339	3.1160
15	47	30.9000	11.9999
16	41	23.5999	6.8199
16	45	39.8699	7.3800
16	47	81.9549	15.1700
17	47	17.3069	2.2410
18	21	9.6149	1.2450
19	22	9.6149	1.2450
20	23	12.8199	1.6600
21	22	4.9440	1.9200
21	47	10.7120	4.1600
22	23	8.2399	3.2000
24	40	17.2259	9.2510
24	44	17.8199	9.5699
25	47	15.9479	2.9520
26	27	7.7219	4.1470
27	46	11.8799	6.3800
28	46	19.2719	1.5840
29	43	7.8479	5.5619
29	44	10.8999	7.7249
30	31	17.4399	12.3599
30	43	21.7999	15.4499
31	32	13.1399	1.0800
33	47	41.1999	16.0000
34	35	11.7719	8.3429
34	48	6.1039	4.3260
36	37	4.3600	3.0900
38	42	15.2599	10.8149
42	47	78.8539	14.5959
44	47	19.6199	13.9049
45	47	42.0849	7.7899
46	47	28.3399	20.0849
49	47	2.4000	10.9398

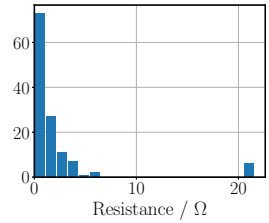
C.5 High voltage grid from section 6.3

The high-voltage grid used in section 6.3 was derived from a real high-voltage grid in the south of Germany. The grid is connected to the overlaying transmission grid by 11 transformers, which are modeled as slack nodes with 5 different voltages, but identical voltage angle in the original dataset. The 98 PQ nodes represent the underlying medium- and low-voltage grids. The active and reactive power was sourced from a measurement at all of the transformers with no further inside into the load characteristics. Many of the PQ nodes have negative active power consumption due to the photovoltaics plants installed in lower grid levels. The lines are all overhead lines except for short connections to transformers, which are not modeled in this dataset.

Table C.5: Overview of properties of the high-voltage grid used in section 6.3

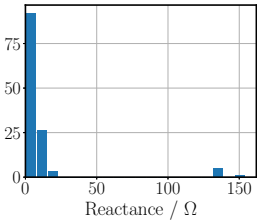
Name	HV Test Grid
Source	not public / shared by grid operator
Location	South of Germany
Number of Nodes	110
Number of Lines	125
Number of Slack Nodes	11
Types of Lines	100% Overhead Lines
Nominal Voltage Level	110 kV
Slack Voltages	117.95 kV - 120.35 kV

Distribution of R



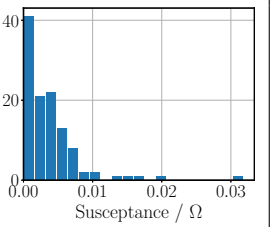
Min	0.0016 Ω
Max	21.6 Ω
Mean	2.15 Ω
Variance	20.3 Ω^2
Skew	3.8
Kurtosis	13.6

Distribution of X



Min	0.001 Ω
Max	154.7 Ω
Mean	11.0 Ω
Variance	853.5 Ω^2
Skew	4.14
Kurtosis	15.6

Distribution of B



Min	0 Ω
Max	0.03 Ω
Mean	0.004 Ω
Variance	1.91 Ω^2
Skew	3.26
Kurtosis	15.4

Table B.3: Node Data for high voltage grid in section 6.3

Node	Type	$\lvert U_{slack}\rvert/V$	Node	Type	$\lvert U_{slack}\rvert/V$
0	PQ	-	34	PQ	-
1	PQ	-	35	PQ	-
2	PQ	-	36	PQ	-
3	PQ	-	37	PQ	-
4	PQ	-	38	PQ	-
5	PQ	-	39	PQ	-
6	PQ	-	40	PQ	-
7	PQ	-	41	PQ	-
8	PQ	-	42	PQ	-
9	PQ	-	43	PQ	-
10	PQ	-	44	PQ	-
11	PQ	-	45	Slack	119321.063757
12	PQ	-	46	PQ	-
13	PQ	-	47	Slack	119321.063757
14	PQ	-	48	PQ	-
15	PQ	-	49	PQ	-
16	PQ	-	50	PQ	-
17	PQ	-	51	PQ	-
18	PQ	-	52	PQ	-
19	PQ	-	53	PQ	-
20	PQ	-	54	PQ	-
21	PQ	-	55	PQ	-
22	PQ	-	56	PQ	-
23	PQ	-	57	PQ	-
24	PQ	-	58	Slack	119900
25	PQ	-	59	PQ	-
26	PQ	-	60	PQ	-
27	PQ	-	61	PQ	-
28	PQ	-	62	Slack	119900
29	PQ	-	63	PQ	-
30	Slack	119786.787748	64	Slack	119900
31	PQ	-	65	PQ	-
32	PQ	-	66	PQ	-
33	PQ	-	67	PQ	-

Node	Type	$\lvert U_{slack}\rvert/V$
68	PQ	-
69	PQ	-
70	PQ	-
71	Slack	120350
72	PQ	-
73	Slack	120350
74	PQ	-
75	PQ	-
76	PQ	-
77	PQ	-
78	PQ	-
79	PQ	-
80	PQ	-
81	PQ	-
82	PQ	-
83	Slack	117950
84	Slack	117950
85	PQ	-
86	PQ	-
87	PQ	-
88	PQ	-
89	PQ	-
90	PQ	-
91	PQ	-
92	PQ	-
93	PQ	-
94	PQ	-
95	PQ	-
96	PQ	-
97	PQ	-
98	PQ	-
99	PQ	-
100	PQ	-
101	PQ	-
102	PQ	-
103	PQ	-
104	PQ	-
105	PQ	-
106	PQ	-
107	PQ	-
108	PQ	-
109	Slack	118800.004721

Table B.4: Line Data for the high voltage grid in section 6.3

From	To	R/Ω	X/Ω	B/mΩ
10	11	0.0147	0.0368	4.0
10	76	0.0011	0.0039	1.3
10	97	0.0000	0.0002	0.0
11	54	0.0000	0.0002	1.4
11	103	0.0138	0.0345	3.8
12	47	0.0129	0.0323	3.5
12	55	0.0000	0.0001	1.1
12	82	0.0343	0.0860	9.4
13	15	0.0198	0.0498	5.4
13	61	0.0032	0.0098	1.1
13	76	0.0030	0.0079	0.9
14	76	0.0030	0.0079	0.9
14	86	0.0243	0.0608	6.6
14	62	0.0032	0.0098	1.1
15	86	0.0045	0.0112	1.2
16	82	0.0256	0.0678	7.5
17	80	0.0004	0.0012	0.1
17	82	0.0078	0.0241	2.7
17	108	0.0253	0.0789	8.8
18	67	0.0033	0.0094	3.5
18	76	0.0015	0.0042	2.7
19	67	0.0057	0.0154	1.5
19	83	0.0066	0.0179	1.8
20	85	0.0029	0.0102	6.9
20	77	0.0000	0.0002	0.0
20	83	0.0270	0.0722	7.2
21	82	0.0053	0.0256	4.2
21	107	0.0002	0.0001	0.6
21	22	0.0018	0.0087	1.8
22	25	0.0143	0.0686	14.0
22	57	0.0000	0.0001	0.5
22	102	0.0000	0.0001	0.0
23	53	0.0163	0.0561	5.7
23	82	0.0102	0.0285	3.8
23	101	0.0000	0.0001	0.0
24	26	0.0030	0.0102	1.0
24	53	0.0134	0.0428	4.4
25	66	0.0079	0.0451	5.4
25	81	0.0000	0.0001	0.0
26	66	0.0074	0.0386	4.1
26	58	0.0076	0.0191	2.1
27	93	0.0066	0.0176	1.9
27	91	0.0094	0.0295	3.2
27	78	0.0000	0.0001	0.0
28	29	0.0091	0.0314	2.8
28	90	0.0074	0.0231	2.6
28	93	0.0085	0.0239	2.5
29	95	0.0000	0.0002	0.0
30	32	0.0027	0.0080	1.0
30	48	0.0039	0.0116	1.4
30	106	0.0000	0.0001	0.0
31	99	0.0012	0.0039	0.4
31	46	0.0107	0.0320	3.9
31	110	0.0108	0.0345	4.0
32	100	0.0012	0.0039	0.4
32	59	0.0074	0.0222	2.7
33	75	0.0084	0.0262	2.9
33	50	0.0015	0.0046	0.5
33	66	0.0186	0.0580	6.4
34	35	0.0214	0.0679	7.2
34	75	0.0049	0.0152	1.7
34	109	0.0130	0.0402	4.5
35	63	0.0000	0.0001	0.0
36	69	0.0000	0.0000	0.0
36	92	0.0036	0.0129	1.4
36	110	0.0023	0.0081	0.9
37	38	0.0116	0.0407	4.6
37	105	0.0000	0.0001	0.0
37	110	0.0108	0.0382	4.3
38	94	0.0055	0.0193	2.2
38	73	0.0143	0.0476	5.4
39	74	0.0015	0.0037	0.4
39	89	0.0020	0.0051	0.6
39	96	0.0138	0.0437	4.8

From	To	R/Ω	X/Ω	B/mΩ
40	56	0.0002	0.0004	0.0
40	84	0.0160	0.0284	2.7
40	98	0.0021	0.0039	0.4
41	76	0.0036	0.0176	6.8
42	76	0.0003	0.0008	4.5
43	76	0.0001	0.0007	5.5
44	70	0.0079	0.0143	1.3
44	52	0.0001	0.0001	0.0
44	60	0.0071	0.0131	1.2
46	66	0.0094	0.0306	3.4
46	66	0.0078	0.0236	2.8
46	65	0.0265	0.0840	15.0
47	97	0.0021	0.0055	1.9
48	110	0.0043	0.0154	1.7
49	67	0.0035	0.0093	1.0
49	67	0.0035	0.0093	1.0
51	58	0.0181	0.0632	7.2
51	94	0.0110	0.0382	4.4
52	96	0.0099	0.0344	3.9
56	96	0.0157	0.0449	5.0
58	66	0.0088	0.0279	4.0
59	66	0.0092	0.0277	3.3
64	74	0.0080	0.0197	2.2
64	96	0.0140	0.0442	6.0
65	85	0.0506	0.1719	31.9
68	76	0.0048	0.0145	5.0
68	85	0.0394	0.1066	16.4
70	104	0.0115	0.0205	1.9
71	84	0.0142	0.0255	2.4
71	104	0.0172	0.0310	2.9
72	82	0.0152	0.0468	5.3
74	96	0.0152	0.0472	5.3
76	97	0.0011	0.0041	1.4
79	82	0.0082	0.0254	3.9
82	93	0.0281	0.0910	10.9
82	93	0.0281	0.0910	10.5
82	103	0.0204	0.0514	5.6
85	109	0.0308	0.1097	19.4
86	108	0.0131	0.0410	4.6
89	110	0.0148	0.0464	5.5