High Performance Algorithms Enabling Real-Time Security Assessment of Sustainable Electric Power Systems

Jørgensen, Christina Hildebrandt Lüthje

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Center for Electric Power and Energy
Department of Electrical Engineering

High Performance Algorithms Enabling Real-Time Security Assessment of Sustainable Electric Power Systems

Christina Hildebrandt Lüthje Jørgensen

PhD Thesis, March 2021, Kongens Lyngby, Denmark

DANMARKS TEKNISKE UNIVERSITET
Center for Electric Power and Energy (CEE)
DTU Electrical Engineering

High Performance Algorithms Enabling Real-Time Security Assessment of Sustainable Electric Power Systems

Effektive algoritmer, der muliggør sikkerhedsvurdering af bæredygtige elsystemer i realtid

Dissertation by Christina Hildebrandt Lüthje Jørgensen

Supervisors:
Arne Hejde Nielsen, Technical University of Denmark
Hjörtur Jóhannsson, Technical University of Denmark
Stefan Horst Sommer, University of Copenhagen

DTU - Technical University of Denmark, Kongens Lyngby - March 2021

High Performance Algorithms Enabling Real-Time Security Assessment of Sustainable Electric Power Systems

This thesis was prepared by:
Christina Hildebrandt Lüthje Jørgensen

Supervisors:
Arne Hejde Nielsen, Technical University of Denmark
Hjörtur Jóhannsson, Technical University of Denmark
Stefan Horst Sommer, University of Copenhagen

Dissertation Examination Committee:
Examiner No. 1 (Chairman)
Department of Electrical Engineering, Technical University of Denmark, Denmark

Examiner No. 2
Department of XX, University of XX, Country

Examiner No. 3
Department of XY, Technical University of YY, Country

Center for Electric Power and Energy (CEE)
DTU Electrical Engineering
Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark

Tel: (+45) 4525 3500
Fax: (+45) 4588 6111
E-mail: cee@elektro.dtu.dk

Release date: March 26th, 2021

Edition: Draft V01

Class: Internal

Field: Electrical Engineering

Remarks: The dissertation is presented to the Department of Electrical Engineering of the Technical University of Denmark in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ISBN: 000-00-00000-00-0

Preface

This thesis is prepared at the Department of Electrical Engineering of the Technical University of Denmark in partial fulfillment of the requirements for acquiring the degree of Doctor of Philosophy in Engineering. The Ph.D. project was funded by the project Security Assessment of Renewable Power Systems funded by the Danish FORSKEL funding framework (grant nr. 2016-1-12427).

This dissertation summarizes the work carried out by the author during his Ph.D. project. It started on 15th May 2017, and it was completed on 26th March 2021. During this period, she was hired by the Technical University of Denmark as a Ph.D. student at the Center for Electric Power and Energy (CEE).

The thesis is composed of 6 chapters and 3 attached scientific papers, which have been peer-reviewed and published.

Christina Hildebrandt Lüthje Jørgensen
March 26th, 2021

Acknowledgements

First and foremost, I would like to thank my supervisors Hjörtur Jóhannsson, Stefan Horst Sommer and Arne Hejde Nielsen for providing guidance throughout the years and for believing in my ability to pursue a PhD degree in electric power system, which was all new to me. The knowledge of power systems provided by Hjörtur as well as the help in setting directions for the project has been valuable. Stefan provided insight in to the algorithmic side of things, and I still remember leaving our first meeting at University of Copenhagen happily thinking, that this is a person who speaks the same language as me. Arne always ensured that I felt welcome and appreciated in the department.

I would like to thank my colleagues at CEE, it has truly been a pleasure working there for the past three years and I will surely miss you all. A special thanks goes to my office mates - Jakob, Can, Theis, Jundi and Ha - for being both great colleagues and friends. The last year has unfortunately been a bit strange due to the COVID-19 pandemic and I have surely missed my colleagues, which I haven't met in person, since I returned from maternity leave.

I would furthermore like to thank Jakob for helping my understanding of power systems and helping shape the initial direction of the project, when all of this was new to me. Can has been a valuable partner in the SARP project and I have enjoyed our scientific discussion as well as the help and guidance provided.

I would like to thank my husband, Kasper, which has supported me in all my endeavours - both celebrating with me in my victories and comforting me in my struggles. Thank you for your endless love and support - for believing in me when I did not myself. Last but not least I would like to thank my son Jonathan for being my biggest fan and for bringing overwhelming happiness in to my life.

Christina Hildebrandt Lüthje Jørgensen

Sønderborg, Denmark, 2021

Table of Contents

List of Figures

List of Tables

Abstract

Power systems are transitioning towards low-carbon emission Renewable Energy Sources (RES) such as wind and photovoltaic power. This type of generation is highly dependent weather conditions, which can cause fluctuations in the system operating point. These fluctuations can happen rapidly as more and more generation is based on RES, making the traditionally time-consuming offline approaches for stability and security assessment insufficient. Therefore, there will be a need for real-time stability and security assessment methods.

This PhD is part of the Security Assessment of Renewable Power Systems (SARP) project. The main goal of the project is to push forward the technology needed to ensure secure and stable system operation of future power systems with a high share of RES based production. In the PhD the factorization method for Thévenin equivalent computations has been investigated, which led to the development of a fast and efficient method for computations of Thévenin equivalents. The voltage stability boundary monitoring method acounting for non-linearities in Thévenin voltages have been omtimized using a binary search with polynomial fitting. Lastly an algorithm for fast refactorization using hierarchical matrices have been developed.

Thévenin equivalent computations is used in a long range of methods recently developed for real-time stability and security assessment methods. It is therefore important that these can be computed fast and efficiently to ensure the methods can operate in real-time. Thévenin equivalents can be computed using an LU factorization to optimally invert the bus admittance matrix. Both direct and incomplete factorization methods can be used for the computations. Clark Kent LU factorization (KLU) is almost twice as fast as the standard LU factorization with no cost of accuracy. However, factorization time is seen to be a negligible part of the computations. The most inefficient part is to compute the impedance matrix for the current sources of the system. Incomplemte LU factorization (ILU) reduced the fill-in of the coefficient matrix for computing Thévenin voltages. ILU reduces the runtime for computing Thévenin voltages at the cost of accuracy. As runtimes have a hard time competing with the direct methods for acceptable error levels the applicability of ILU is severely limited.

The impedance matrix for the current sources is used to determine the coefficient matrix for computing Thévenin voltages, however this is inefficient to compute due to the density. Therefore the *factor-solve* method, which avoids these computations, is developed by using KLU. KLU factorization brings a matrix on block triangular form, where each block consists of all nodes, that are connected. For a system of linear equations it uses block back substitution to find the solution. The *factor-solve* method uses block back substitution to determine the Thévenin voltages and thereby avoids computing the coefficient matrix. This makes it possible to determine the Thévenin voltages in linear time compared to the reference implementation, which using the coefficient matrix had close to quadratic complexity.

Computations for Thévenin impedances is still quadratic, however these can easily be parallelized resulting in runtimes lower than 3 seconds for systems up to 30.000 buses, whereas Thévenin

voltages can be determined in less than 6 milliseconds. The Thévenin impedances only depend on system topology and therefore doesn't need to be recomputed as often.

The distance to the boundary of voltage instability can be determined more accurately by taking in to account non-linearity in Thévenin voltages as the load changes. The maximum power transfer to a non-controlled load is used to determine this distance, and repeated changes in load impedance is required to find this.

The maximum power transfer is found between the current load and the Thévenin impedance (which is the boundary normally determined by the impedance match criterion). Doing this naïvely by splitting the interval evenly results in poor runtime and accuracy. Therefore, a binary search is used instead. This determines the load impedances that the power transfer should be computed to find the maximum power transfer. The binary search is further improved by combining the search with a polynomial fitting. Feasible points are fitted to a second order polynomial, and the maximum of the polynomial is found. By utilization of parallelization it is possible to determine the margin for 2.000 non-controlled loads in a 3.000 bus system in less than 6 seconds, while also determining the maximum power transfer accurately.

Even though runtimes are improved the complexity is still quadratic, therefore the scaling needs to be improved. The computations are dominated by a large block in the block triangular form and therefore the computations for this is sought to be optimized. The voltage stability method requires the matrix to be refactorized for each change in load impedance and therefore efficient refactorization would improve runtimes.

The hierarchical matrix structure constructs a hierarchical tree for a chosen level of discretization, where the leaf nodes and edges between these represent the matrix to be factorized. The factorization is then done by eliminating the leaf nodes and compressing fill-in generated to the parent level by approximating using a low-rank approximation such as Singular Value Decomposition. To do fast refactorization the effect a change in the diagonal have on the computations is tracked in the initial factorization. This makes it possible to do refactorization, where only the nodes affected by the change is recomputed. This lowers runtime considerably, however work still remains for the factorization to work for all systems and levels of discretization.

Resumé

Elnetværket ændrer sig i til at være mere og mere afhængig af vedvarende energi såsom vind og solkraft. Denne type energi er meget afhængig af vejret og dette resulterer derfor at systemets drift punk kan fluktuere. The fluktueringer kan komme hutigt eftersom mere og mere energi kommer fra vedvarende energikilder. Dette gør, at traditionelle tidskrævende metoder til stabilitets og sikkerhedsanalyse bliver utilstrækkelige. Derfor vil der være et behov for stabilitets og sikkerhedsmetoder som kan operere i real tid.

Dette PhD er en del af SARP projektet, som har til hovedformål at skubbe teknologien, som skal sikre sikker og stabil system drift af fremtidens elnetværk med store mængder vedvarende energi, fremad. I denne PhD bliver faktoriseringsmetoden brugt i Thévenin ækvivalentberegninger, som har ledt til udviklingen af en hurtig og effektiv metode for disse beregninger. Metoden for at finde afstanden til spændingsustabilitet, som tager højde for ikke-lineære ændringer i Thévenin spænding, er blevet optimeret ved brug af en binær søgning med polynomial regression. Til slut er der blevet udviklet en algoritme til hurtigt at refaktorisere ved brug af hierarkiske matricer.

Thévenin ækvivalentberegninger bliver brugt i en lang række metoder, som er udviklet i de senere år inden for real tids stabilitets og sikkerhedsanalyse. Det er derfor vigtigt at sikre sig at de kan blive beregnet hurtigt og effektivt. Thévenin ækvivalentberegninger benytter sig af en LU faktorisering for optimalt at invertere en bus admittansmatrice. Både direkte og ufuldstændige faktoriseringsmetoder kan benyttes. Clark Kent LU faktorisering (KLU) er næsten dobbelt såhurtig som standard LU faktorisering uden at miste noget pånøjagtigheden af beregningerne. Desværre er faktorisering en ubetydelig del af beregningerne, og det har derfor ikke den store effekt. Den mest ineffektive del af beregningerne er at beregne impedansmatricen for strømkilderne. Den ufuldstændige (incomplete) LU faktorisering (ILU) reducerer mængden af elementer forskellig fra 0 i koefficientmatricen, som bliver brugt i beregningen af Thévenin spændinger. ILU reducerer beregningstiden påbekostning af nøjagtighed. Fordi beregningstiderne for ILU har svært ved at konkurrere med de direkte metoder for acceptable fejlniveauer, begrænser det brugbarheden af ILU.

Impedansmatricen for spændingskilder er brugt i beregningen af koefficientmatricen, som er ineffektiv at beregne fordi den er en meget tæt matrice. Derfor er *factor-solve* metoden udviklet som undgår at beregne disse ved at bruge KLU. KLU faktoriseringer bringer en matrice op blok triangulær form, hvor hver blok består af de punkter, som er forbundne. For et lineært system af ligninger benytter KLU sig af blok substitution for at finde løsningen. *Factor-solve* metoden benytter sig af dette såThévenin spændinger kan beregnes uden koefficientmatricen. Dette gør det muligt at bestemme Thévenin spændinger i lineær tid sammenlignet med referencen, som bruger koefficientmatricen, hvilket resulterer i tæt påkvadratisk kompleksitet.

Beregninger for Thévenin impedanser har stadig kvadratisk kompleksistet, men disse kan nemt paralleliseres, hvilket giver beregningstider lavere end 3 sekunder for systemer op til 30.000

busser. Thévenin spændinger kan derimod bestemmes for hele systemer påunder 6 millisekunder. Thévenin impedanser skal kun bestemmes når topologien for systemet ændrer sig og er derfor ikke nødvendig at beregne såofte.

Afstanden til grænsen for spændingsstabilitet kan bestemmes mere nøjagtigt ved at tage højde for ikke-linearitet i Thévenin spændinger, når belastningen ændrer sig. Den maksimale effektoverførsel til en belastning bliver brugt i udregningen af afstanden til grænsen og gentagne ændringer i belastnings impedansen er nødvendig for at finde denne.

Den maksimale effektoverførsel kan findes mellem impedansen for den nuværende belastning og Thévenin impedansen. Thévenin impedansen bliver normalt brugt som grænsen og bliver kaldt impedans matchning kriteriet. Hvis man helt naivt splitter intervallet mellem belastning og Thévenin impedans for man dårlige beregningstider og et unøjagtigt svar. Derfor bliver der i stedet brugt en binær søgning, som bestemmer hvilket skridt der skal tages for at finde den maksimale effektoverførsel. Dette bliver kombineret med polynomial regression med et anden ordens polynomium og maksimum for dette bliver brugt i søgningen. Ved at benytte sig af parallelisering er det muligt at bestemme afstanden til grænsen for spændingsstabilitet for 2.000 belastninger i et 3.000 bus system påmindre end 6 sekunder. Derudover bliver den maksimale effektoverførsel ogsåfundet mere nøjagtigt.

Selvom beregningstider er bedre med den binære søgning er kompleksiteten stadig kvadratisk, hvilket gerne skulle forbedres for at kunne skalere bedre med større systemer. Beregninger er domineret af en stor blok i den blok triangulære form, hvilket ogsåer at se i beregninger. Fordi spændingstabilitetsmetoden kræver flere beregninger for at finde den maksimale effektoverførsel bliver der udviklet en metode til at refaktorisere matricen.

Den hierarkiske matrix struktur tager udgangspunkt i et hierarkisk træfor det valgt niveau af diskretisering. Det nederst blade i træet og vejene mellem disse repræsenterer matricen, som skal faktoriseres. Faktoriseringen udføres ved at eliminere bladene og komprimere de ekstra veje der dukker op. Disse bliver komprimeret til det næste niveau ved at brug en lav rangs approksimation såsom SVD. For at refaktorisere effektivt, når et element i diagonalen bliver ændret, bliver elementer påvirket af ændringen markeret under den første faktorisering, og det vil såkun være disse elementer som bliver genberegnet. Det giver lavere beregningstid, men der ligger desværre stadig noget arbejde i at fåfaktoriseringen til at virke for alle systemer og alle niveauer.

Acronyms

AMD	Approximate Minimum Degree
ASSRAS	Aperiodic Small-Signal Rotor Angle Stability
AVR	Automatic Voltage Regulator
BSPF	Binary Search with Polynomial Fitting
BTF	Block Triangular Form
cs node	current source
GPS	Global Positioning System
GTC	Grid-Transformation Coefficient
ILU	Incomplete LU factorization
ILUC	Crout Incomplete LU factorization
HPC	High Performance Computing
KLU	Clark Kent LU factorization
nc	non-controlled
PMU	Phasor Measurement Unit
RES	Renewable Energy Sources
RTDS	Real Time Digital Simulator
SARP	Security Assessment of Renewable Power Systems
SOSPO	Secure Operation of Sustainable Power Systems
SVD	Singular Value Decomposition
SW-platform	Software platform
TESCA	Thévenin Equivalent Based Static Contingency Assessment
TVE	Total Vector Error
vs node	voltage source
vc	voltage controlled
WAM	Wide-Area Measurement

CHAPTER 1

Introduction

1.1 Background

In recent years there has been a big shift in the composition of power systems. Many countries are going from fossil fuel to low-carbon emission Renewable Energy Sources (RES) such as wind and photovoltaic power. With the Paris Agreement many countries agreed to reduce the emission of greenhouse gasses due to their effect on climate changes. The EU agreed to reduced greenhouse gas emission by 40% by 2030 compared to 1990 levels. This goal was recently, in 2019, upgraded and EU now seek to reach a 45 % reduction by 2030 [1]. One of the steps on the road to reach this goal is to base at least 32% of all energy consumption on RES by 2030. The danish goal is even more ambitious with a 70% reduction of emission of greenhouse gasses compared to 1990 by 2030 [2], which includes a goal to have at least 50% RES. In 2018 this number reached 36% and is expected to reach 55% by 2030. By 2050 Denmark aims at being completely independent of fossil fuels [3].

On top of this shift towards a system with a high share of RES, the energy demand is also increasing due to growing populations and development of societies. From 2004 to 2014 the electric power consumption per capita increased by more than 21%[1]. The growth will continue steadily the coming years, and the reason for this can be found in the continued commitment to electrification of the transport sector and district heating as well as the new large data centres [2]. The increased dependency on electricity will put even larger requirements on electric power system's capability to ensure a stable and secure operation. A transition of power systems towards one where a high share of power production is based on RES will introduce new challenges, when stable and secure system operation is to be ensured.

For the past decades the electric power system was composed of few centralized units. Traditionally, approaches to ensure stability planned production hours in advance. This was made possible by a knowledge of the load demand and the ability of large power plants to quickly adjust production. The integration of RES to the existing power system changes the dynamics of the system. Wind power and photovoltaic units are fluctuating and highly dependent on weather. Planning the generation of these units is highly dependent on methods for forecasting. Furtheremore, to be able to match generation and load in the power system a lot of research is carried out in energy storage, load balancing, smart charging and demand response to mention a few.

The traditional methods to enable secure and stable operation was done offline, due to their inefficiency making them quite time-consuming. As the system condition fluctuate due to the large amount of RES in the power system, they will become insufficient. This introduces a need for real-time stability assessment methods to ensure secure and stable operation of such systems [4]. The introduction of Phasor Measurement Units (PMUs) [5, 6] has been a basis for recent research in real-time stability assessment methods. By placing PMUs on nodes of importance in the power

[1] http://data.worldbank.org

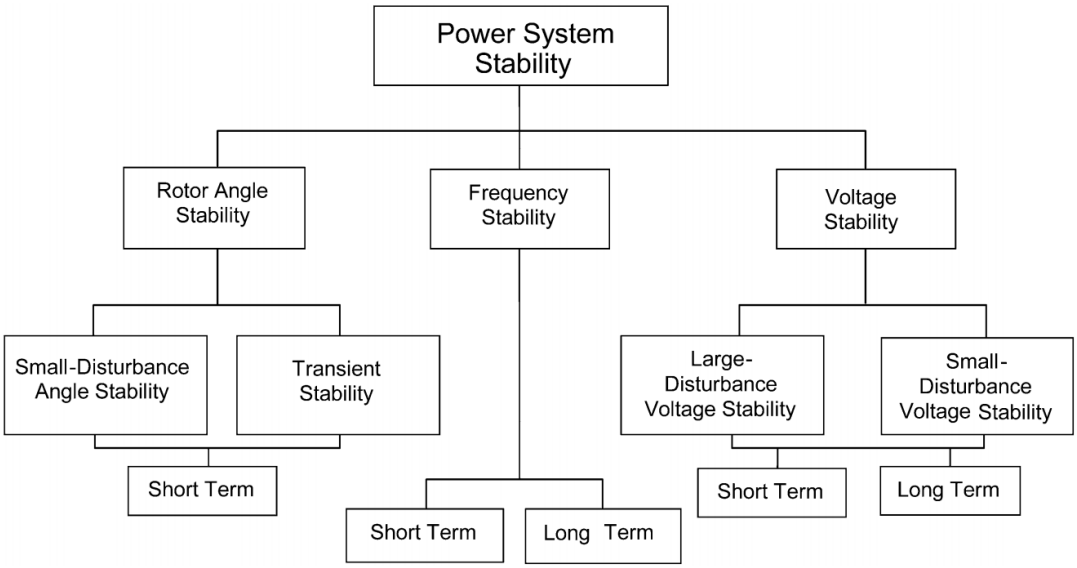


Figure 1.1: Classification of power system stability and subcategories [11].

system and synchronizing these using a Global Positioning System (GPS) makes it possible to monitor system conditions at the rate of the system frequency [7–9].

The availability of Wide-Area Measurements (WAMs) by the use of PMUs enables the development of new methods for real-time stability and control. However, another essential factor is the efficiency of such methods to ensure their capability to operate in real-time. It is both important that the methods have runtimes suitable for real-time operation and that the amount of computational resources (CPU, memory, etc.) are within a feasible range of modern computational capabilities. This introduces a need for High Performance Computing (HPC) techniques as well as other modelling methods to be utilized to overcome the bottleneck in power grid operations, which are mostly run on serial computers today [10].

1.1.1 Power System Stability

Before discussing methods for power system stability assessment the term power system stability needs to be defined. The definition that will be used is [11]:

> *Power system stability is the ability of an electric power system, for a given initial operating condition, to regain a state of operating equilibrium after being subjected to a physical disturbance, with most system variables bounded so that practically the entire system remains intact.*

The subcategories of stability can be seen in Figure 1.1.

- **Rotor Angle Stability** refers to a synchronous machines ability to remain in synchronism after a disturbance. This can be split in to transient (large disturbance) and small signal (small disturbance) stability. Transient instability occurs as aperiodic angular separation due to insufficient synchronizing torque. Small signal instability can be further subdivided in to aperiodic (non-oscillatory) increase of rotor angle due to lack of synchronizing torque or oscillatory with rotor oscillations of increasing amplitude due to lack of sufficient damping torque.

- **Frequency Stability** refers to the ability of a power system to maintain steady frequency following a disturbance resulting in a significant imbalance between generation and load.
- **Voltage Stability** refers to the ability of a power system to maintain steady voltages at all buses in the system after being subjected to a disturbance. The term voltage collapse is used to describe the process by which a sequence of events accompanying voltage instability leads to a blackout or abnormally low voltages in a significant part of the system. Voltage stability can be subdivided in to voltage instability due to a large or a small disturbance.

1.2 State of the art

Stability and security assessment methods suitable for real-time is important for future power system based on RES. The development of these methods was the focus of the Secure Operation of Sustainable Power Systems (SOSPO) project [12]. SOSPO was a four year project and the research conducted in this project serves as a basis for the work in this PhD. The focus was not solely on real-time stability and security assessment but also on methods capable of deploying appropriate control countermeasures for when unsafe operation is approaching.

In the SOSPO project it is assumed that there is full observability by WAMs from the distribution of PMUs. A review of methods for long-term voltage stability concluded that using Thévenin equivalents for calculations was computationally efficient [13]. Thévenin equivalents furthermore give a credible representation of the system in assessment of power system stability.

A Thévenin equivalent method for real-time voltage stability is proposed in [14–16], which also explores a synergy with sensitivity analysis. In [17] sensitivities are derived from a Thévenin equivalent representation of the system to be able to detect instability issues concerning voltage sags after a transient disturbance. The Thévenin Equivalent Based Static Contingency Assessment (TESCA) method performs a contingency analysis for aperiodic small-signal instability [18, 19]. Thévenin equivalents are computed repeatedly for each contingency to find a steady-state post contingency, and the computations make use of a Grid-Transformation Coefficient (GTC) matrix.

The GTC matrix decompose the Thévenin voltage in components induced by each voltage source [20, 21] for a power system split in to voltage controlled and non-controlled nodes. The GTC matrix is used in [14] to improve Thévenin equivalent methods for voltage stability. The expression for the GTC matrix is reevaluated in [22] and defined in terms of the Schur complement [23]. The computation are further extended by modelling loads not only by their impedance but as current sources injecting a current into the power system.

The reduction of the network resulting in the Schur complement is called Kron reduction [24]. The Schur complement is used for voltage stability assessment in [25] to lower the computational burden of Thévenin impedances for load busses, and in [26, 27] the Schur complement is applied to compute Thévenin impedances for generator busses.

The use of a Schur complement in Thévenin equivalent computations has previously been investigated and was determined to have a cubic overhead making it a burden for large systems [28]. The Schur complement is generally considered to be dense [26, 29] making it inefficient to compute, however LU factorization can be used to make computations more efficient. Only the diagonal of the Schur complement is needed to determine Thévenin impedances for voltage controlled nodes. Therefore, the computations can be limited to computing these elements to

avoid the dense fill-in of the entire matrix. Furthermore, parallelization is an important step in optimizing computations. In [27] the method is extended to compute the full Schur complement, which is done efficiently by parallelizing the computations on a GPU. The use of LU factorization to efficiently compute Thévenin impedances was already shown in [30], where the default LU factorization in MATLAB (UMFPACK) [31] was used. The newer methods [26, 27] uses Clark Kent LU factorization (KLU) [32].

As part of the SOSPO project a Software platform (SW-platform) was developed [33, 34]. The platform provides structured access to model parameters and real-time PMU snapshots, which eases the implementation process of new methods. The platform can work with a stream of predefined snapshots or utilize a Real Time Digital Simulator (RTDS).

This PhD is part of the Security Assessment of Renewable Power Systems (SARP) project, which is the direct continuation of the SOSPO project. The main objective of this project is to push forward the technology needed to ensure secure and stable system operation of future power systems with a high share of RES based production. The research in the SARP project is split in to 4 areas:

- *Modelling of RES Under Stressed Conditions and Its Impact on System Stability*
- *Assessment of Voltage Stability in RES Based Power Systems*
- *High Performance Power System Algorithms*
- *SW-platform Improvements and Developments Towards Online Operation*

The PhD falls under the third category.

1.3 Focus of PhD project

The purpose of the PhD project is to develop high performance algorithms capable of real-time security assessment of future power systems. This is done by modifying existing methods to enhance their performance and ensure their feasibility. To achieve this goal 4 objectives were defined:

- Determine computationally critical elements of existing methods for real-time stability and security assessment.
- Develop high performance algorithms through performance enhancing by exploiting graph theoretical properties and avoid repeated calculations
- Develop algorithms utilizing extensive parallel processing.
- Implement and test methods by comparing them to current state-of-the-art methods for real-time stability and security assessment.

1.4 Contributions

The main contributions of the presented work are:

- *An evaluation of the effect factorization methods have on Thévenin equivalent computations* **[Pub. A]**
  Three different factorization methods have been used in Thévenin equivalent computations and was determined to have a low impact on the computations as the factorization methods have negligible runtime compared to the entire computations. The incomplete factorization method is shown to have limited applicability in the computations due to errors along with high runtimes.
- *An improved method for effecient Thévenin equivalent computations* **[Pub. B]**
  The *factor-solve* method efficiently compute the Thévenin equivalents for all buses in the power system. Determining the Thévenin impedances have quadratic complexity, however computations can easily be parallelized. Thévenin voltages can be determined in linear time resulting in runtimes of only a few milliseconds for systems of several thousand of buses. The *factor-solve* method doesn't compute the dense coefficient matrix reducing memory requirements from several gigabytes to a few megabytes for larger systems.
- *A binary search for voltage stability boundary monitoring* **[Pub. C]**
  A binary search is developed to find the maximum power transfer to a non-controlled load. The algorithm is further optimized by fitting feasible points in the search to a second order polynomial. This makes it possible to determine the maximum power transfer faster and more accurately. Parallelization can be used to lower runtimes even further.
- *Algorithm for refactorization using hierarchical matrices*
  A refactorization algorithm is developed, which makes it possible to do fast refactorization of a matrix, when a value in the diagonal changes. This is done by using the factorization for hierarchical matrices, which compresses fill-in to auxiliary variables by using a low rank approximation.

The minor contributions of the presented work are:

- *Block triangular form of a bus admittance matrices have non-zeros in the off-diagonal* **[Pub. A]**
  The block triangular form for a bus admittance matrices with complex admittances is shown to have off-diagonal elements, which are all 0. THis is shown by using definitions for strongly connected components and strong Hall matrices. This ensures, that KLU can be used for the Thévenin equivalent computations.
- *Analysis of relationship between block sizes in block triangular form and the density of coefficients for computation of Thévenin voltage*
  The theoretical density of the impedance matrix for current sources are determined using the block triangular form and shown to be accurate as numerical cancellation is small. The block triangular form is further shown to be dominanted by one large block.
- *Analysis of block-wise calculations for optimization of runtime*
  For systems, where the load impedance is seen outside the Thévenin equivalent the computations has to be done independently for each load. This makes it possible to limit computations to the block in block triangular form, that the node it part of. However, as the block triangular form is dominated by one large block so is the block-wise computations.

1.5 Thesis outline

The thesis is composed of 6 chapters with the following outline:

Chapter 1: Introduces the background and motivation for the research. Power system stability is defined which leads to a presentation of the state of the art literature for stability assessment of electric power systems. From the discussion of the state of the art 4 objectives for the PhD project is defined. The chapter ends with a list of contributions, outline of the thesis and a list of publications.

Chapter 2: Presents Thévenin equivalent computations through the use of a Schur Complement. This leads to an investigation of the factorization method used in the computation to determine the optimal one for fast and efficient computations. The structure of KLU factorization is analyzed and lastly the density of the coefficient matrix is analyzed using the Block Triangular Form (BTF).

Chapter 3: Develops a *factor-solve* method for computing the full Thévenin equivalent for all nodes in a system divided in to current and voltage sources. The method is investigated in term of both runtime and memory requirements. Furthermore, the method is parallelized to optimize runtime.

Chapter 4: Introduces the recently developed method for voltage stability boundary monitoring, which accounts for non-linearity of Thévenin Voltage. Computational improvements is suggested by using a binary search to find the maximum power delivered to a load. Additionally it is investigated how runtimes can be improved by dividing the system in to smaller blocks.

Chapter 5: Introduces the hierarchical matrix structure, which can be used to solve sparse systems in linear time. This is then used to develop an algorithm to do fast recalculation of the factorization, when an element in the diagonal of the matrix changes.

Chapter 6: Concludes the research conducted and gives an outlook on possible directions for future work.

1.6 List of publications

The relevant publications which are the core of this thesis are as follows:

[Pub. A] Christina Hildebrandt, Bahtiyar Can Karatas, Jakob Glarbo Møller, and Hjörtur Jóhannsson. Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment. In *Proceedings of 20th Power Systems Computation Conference (PSCC)*, Dublin, Ireland, 2018

[Pub. B] Christina Hildebrandt Lüthje Jørgensen, Jakob Glarbo Møller, Stefan Sommer, and Hjörtur Jóhannsson. A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment. *IEEE Transactions on Power Systems*, 2019

[Pub. C] Christina Hildebrandt Lüthje Jørgensen, Bahtiyar Can Karatas, Hjörtur Jóhannsson, and Stefan Sommer. Binary Search and Fit Algorithm for Improved Voltage Stability Boundary Monitoring. In *Proceedings of 9th IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Bucharest, Romania, 2019

CHAPTER 2

Thévenin Equivalent Computations

This chapter describes the Thévenin equivalent computations and the block-wise partitioning of the power system for these calculations. Three different factorization methods - UMFPACK, Clark Kent LU factorization (KLU) and Incomplete LU factorization (ILU) are introduced and their influence on the computations are investigated. Before comparison the structure of KLU is analyzed and a tolerance for ILU is determined. Furthermore, the Block Triangular Form (BTF) used in KLU provides a way to theoretically determine the resulting fill-in of the coefficient matrix. The main results of this chapter is based on **[Pub. A]** and **[Pub. B]**.

2.1 Thévenin equivalent circuit

The Thévenin equivalent for a node is an equivalent representation of the remainder of the system seen from that node. The equivalent consists of a Thévenin impedance $\overline{Z}_{th}$ and a Thévenin voltage $\overline{V}_{th}$, such that the equivalent seen from node i satify

$$\overline{V}_{th,i} = \overline{V}_i - \overline{Z}_{th,i}\overline{I}_i \tag{2.1}$$

$\overline{V}_i$ is the node voltage and $\overline{I}_i$ is the current injected at node i.

The power system will in the computations be divided in to two sets of nodes. These two sets generally represent generators and loads. In the first part of this thesis the system will be split in to voltage sources (vs nodes) and current sources (cs nodes). Another way of splitting the system is to divide the nodes in to voltage controlled (vc nodes) node and non-controlled (nc nodes), which will be discussed and used later in the thesis.

Generators with Automatic Voltage Regulator (AVR) or internal voltages of manually excited machines seen behind the synchronous reactance jX_d are represented as voltage sources. Floating nodes can be modelled as a current sources injecting 0 current. Loads may be represented as impedances, dependent- or independent current sources.

The admittance matrix for the system can be block-wise partitioned by the two sets of nodes

$$\begin{bmatrix} I_{cs} \\ I_{vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{cs} & \mathbf{Y}_{v\to c} \\ \mathbf{Y}_{c\to v} & \mathbf{Y}_{vs} \end{bmatrix} \begin{bmatrix} V_{cs} \\ V_{vs} \end{bmatrix} \tag{2.2}$$

$\mathbf{Y}_{v\to c}$ and $\mathbf{Y}_{c\to v}$ contains the edges (transmission lines of the system) from vs nodes to cs nodes and vice versa. The admittance matrix is typically symmetric, but there can be components in the system making it asymmetric. As an example this could be a phase-shifting transformer. However, the non-zero pattern will always be symmetric, since transmission lines are bidirectional.

Using (2.2) expressions for V_{cs} and I_{vs} can be determined

$$V_{cs} = \mathbf{Y}_{cs}^{-1} \left(I_{cs} - \mathbf{Y}_{v\to c} V_{vs} \right) \tag{2.3}$$

$$I_{vs} = \mathbf{Y}_{eq} V_{vs} - \mathbf{Q}_{ac} I_{cs} \tag{2.4}$$

with

$$\mathbf{Y}_{eq} = \mathbf{Y}_{vs} - \mathbf{Y}_{c\to v} \mathbf{Y}_{cs}^{-1} \mathbf{Y}_{v\to c} \tag{2.5}$$

$$\mathbf{Q}_{ac} = -\mathbf{Y}_{c\to v} \mathbf{Y}_{cs}^{-1} \tag{2.6}$$

where $\mathbf{Y}_{eq}$ is the Schur complement and $\mathbf{Q}_{ac}$ is the accompanying matrix.

The Thévenin impedances seen from node i is determined by short circuiting all voltage sources and open-circuiting all current sources and will be the diagonal of the impedance matrix

$$Z_{th,i} = \begin{cases} \mathbf{Z}_{cs}(i,i) & i \in cs \\ \mathbf{Y}_{eq}(i,i)^{-1} & i \in vs \end{cases} \tag{2.7}$$

where $\mathbf{Z}_{cs} = \mathbf{Y}_{cs}^{-1}$.

Substituting (2.3) and (2.4) in to (2.1) the Thévenin voltages for cs nodes and vs nodes are computed as

$$V_{th,cs} = -\mathbf{Z}_{cs} \mathbf{Y}_{v\to c} V_{vs} + (\mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs})) I_{cs} \tag{2.8}$$

$$V_{th,vs} = (\mathcal{I} - \mathcal{D}(Z_{th,vs}) \mathbf{Y}_{eq}) V_{vs} + \mathcal{D}(Z_{th,vs}) \mathbf{Q}_{ac} I_{cs} \tag{2.9}$$

$\mathcal{I}$ is the identity matrix and $\mathcal{D}(Z_{th})$ is the diagonalization of the vector Z_{th} into a diagonal matrix. (2.8)-(2.9) can be written on the form

$$\begin{bmatrix} V_{th,cs} \\ V_{th,vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix} \begin{bmatrix} I_{cs} \\ V_{vs} \end{bmatrix} \tag{2.10}$$

with

$$\mathbf{Z}_c = \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs}) \tag{2.11}$$

$$\mathbf{K}_{v\to c} = -\mathbf{Z}_{cs} \mathbf{Y}_{v\to c} \tag{2.12}$$

$$\mathbf{Z}_{c\to v} = \mathcal{D}(Z_{th,vs}) \mathbf{Q}_{ac} \tag{2.13}$$

$$\mathbf{K}_v = \mathcal{I} - \mathcal{D}(Z_{th,vs}) \mathbf{Y}_{eq} \tag{2.14}$$

The coefficients $\mathbf{K}_v$ is the GTC matrix, which was mentioned in the introduction. The matrix defined in (2.10) will be defined as the coefficient matrix $\mathbf{K}$.

Determining the full Thévenin equivalent requires the Thévenin impedances and the coefficient matrix $\mathbf{K}$. A vital part of the computations is determining the inverse of the admittance matrix for the cs nodes $\mathbf{Z}_{cs} = \mathbf{Y}_{cs}^{-1}$, which is generally considered inefficient to compute. This is therefore done by using a LU-factorization of $\mathbf{Y}_{cs}$, since $\mathbf{L}$ and $\mathbf{U}$ is computationally more efficient to invert than the full matrix [22, 26]. Algorithm 1 shows all the steps required to compute Z_{th} and $\mathbf{K}$.

$\mathcal{D}(\mathbf{X})$ is a vector containing the diagonal of the matrix $\mathbf{X}$, while $\mathcal{D}(X)$ a diagonal matrix with the vector X along the diagonal.

The Thévenin voltages can be determined by (2.10) using the coefficients $\mathbf{K}$, the current injected at cs nodes I_{cs} and the voltage at vs nodes V_{vs}.

Algorithm 1 Thévenin equivalents

1: $\mathbf{L}_{cs}, \mathbf{U}_{cs} \leftarrow$ factorization of $\mathbf{Y}_{cs}$
2: $\mathbf{U}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{L}_{cs}, \mathcal{I}$)
3: $\mathbf{L}^T_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{U}^T_{cs}, \mathcal{I}$)
4: $\mathbf{Z}_{cs} \leftarrow \mathbf{L}_{\mathbf{Z}_{cs}} \mathbf{U}_{\mathbf{Z}_{cs}}$
5: $Z_{th,cs} \leftarrow \mathcal{D}(\mathbf{Z}_{cs})$
6: $\mathbf{Q}_{ac} \leftarrow -\mathbf{Y}_{c\to v}\mathbf{Z}_{cs}$
7: $\mathbf{Y}_{eq} \leftarrow \mathbf{Y}_{vs} + \mathbf{Q}_{ac}\mathbf{Y}_{v\to c}$
8: $Z_{th,vs} \leftarrow \mathcal{D}(\mathbf{Y}_{eq})^{-1}$
9: $\mathbf{Z}_c \leftarrow \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs})$
10: $\mathbf{K}_{v\to c} \leftarrow -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}$
11: $\mathbf{Z}_{c\to v} \leftarrow \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}$
12: $\mathbf{K}_v \leftarrow \mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq}$
13: $Z_{th} \leftarrow \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}$
14: $\mathbf{K} \leftarrow \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix}$
15: **return** Z_{th} and $\mathbf{K}$

2.2 Factorization methods

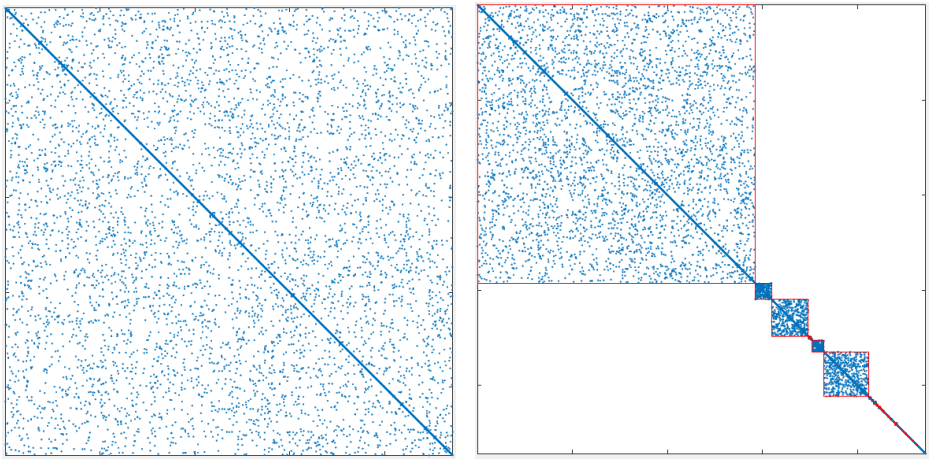
Step 1 of Algorithm 1 computes an LU factorization of the admittance matrix for the cs nodes $\mathbf{Y}_{cs}$. The factorization is the basis of all the computations. It will therefore be investigated how changing the factorization method might affect the computations. The three different factorization methods chosen are:

- *UMFPACK with Approximate Minimum Degree (AMD) ordering*: This is the standard LU factorization in MATLAB. AMD is an ordering scheme used prior to factorization to reduce both runtime and the amount of fill-in [38].
- *KLU*: This factorization method is optimized for sparse systems [32] and is part of the SuiteSparse library [39]. KLU converts the system to BTF. Each block is then reordered using AMD before being factorized, while off-diagonal elements are kept as is. The structure of KLU will be treated later.
- *ILU*: This is an incomplete solver meaning it computes an approximation of the LU factorization. The chosen type will be the Crout Incomplete LU factorization (ILUC) [40]. The method requires a tolerance, which is used to set elements of the matrix to 0. If a value is smaller than the tolerance multiplied by the norm of the row and the tolerance multiplied by the norm of the column, the value is deemed insignificant and set to 0. Furthermore, ILU is set to preserve row sums, since it was empirically seen to significantly increase accuracy without affecting the runtime. The tolerance level will be treated later.

2.2.1 KLU of an admittance matrix

KLU brings the system on BTF. The diagonal of the resulting matrix will contain square matrices with zero-free diagonal and the off-diagonal will contain potentially non-zero blocks. The blocks below the diagonal will be zero.

$$\begin{bmatrix} A_{11} & \cdots & A_{1k} \\ & \ddots & \vdots \\ & & A_{kk} \end{bmatrix} \tag{2.15}$$



(a) (b)

Figure 2.1: (a) Sparsity pattern for $\mathbf{Y}_{cs}$ for the system Pegase2869 from MATPOWER [41] and (b) $\mathbf{Y}_{cs}$ on BTF

Figure 2.1 shows an example of an admittance matrix and it's BTF. The test system in this example is Pegase2869 taken from MATPOWER [41].

As described earlier the diagonal blocks is reordered using AMD ordering. Each block is then factorized to obtain the LU factorization, whereas the potential off-diagonal elements are kept as is. Due to the off-diagonal elements the structure of the KLU factorization for a matrix $\mathbf{A}$ will be

$$\mathbf{PRAQ} = \mathbf{LU} + \mathbf{F} \tag{2.16}$$

$\mathbf{P}, \mathbf{Q}$ are permutation matrices, $\mathbf{R}$ is a diagonal scaling matrix, $\mathbf{L}, \mathbf{U}$ are the factorization of the diagonal elements and $\mathbf{F}$ represents the entire off-diagonal. To solve a linear system of equations KLU use block back substitution.

The structure of KLU in (2.16) does not fit in to Algorithm 1 due to the off-diagonal elements in $\mathbf{F}$. However, this will not be an issue, since for an admittance matrix with complex admittances $\mathbf{F} = \mathbf{0}$, which will be shown below.

BTF of a square matrix $\mathbf{A}$ with zero-free diagonal corresponds to finding the strongly connected components of a directed graph $G(\mathbf{A})$ [39]. An admittance matrix with complex admittances will always have a zero-free diagonal. Therefore, BTF of $\mathbf{Y}_{cs}$ will correspond to finding the strongly connected components in the directed graph $G(\mathbf{Y}_{cs}) = (V, E)$ with the nodes $V = \{1, \ldots, |cs|\}$ and the edges $E = \{(i, j) \mid \mathbf{Y}_{cs}(i, j) \neq 0\}$, where $|cs|$ is the number of cs nodes.

A strongly connected component is defined as the maximal set of nodes such that for any pair of nodes in the set, the paths $i \rightsquigarrow j$ and $j \rightsquigarrow i$, exists. This means, that for nodes to be in the component, there should be a path both from node i to node j and from node j to node i for any two nodes. There need not be a direct edge between the nodes, but it should be possible to follow a path from one node to the other along edges in the component.

The non-zero pattern of an admittance matrix is symmetric. This means, that if

$$\mathbf{Y}_{cs}(i, j) \neq 0 \Leftrightarrow \mathbf{Y}_{cs}(j, i) \neq 0 \tag{2.17}$$

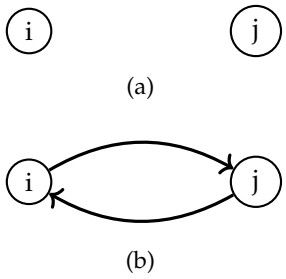


Figure 2.2: The directed graph $G(\mathbf{Y}_{cs})$ will either have (a) no edges between two nodes i and j or (b) 2 edges (i, j) and (j, i)

and in equality

$$\mathbf{Y}_{cs}(i, j) = 0 \Leftrightarrow \mathbf{Y}_{cs}(j, i) = 0. \tag{2.18}$$

Therefore, there are two scenarios for edges between two nodes i and j in the directed graph $G(\mathbf{Y}_{cs})$. Either there will be no edges between the nodes i and j or there will be an edge both from i to j, (i, j), and from j to i, (j, i), see Figure 2.2. This means that two nodes will either be in the same strongly connected component or they will be completely separated, since the graph is bi-directed. Hence BTF of $\mathbf{Y}_{cs}$ will consist of the strongly connected components in the diagonal and the entire off-diagonal will be empty, since there is no connection between the components.

Edges in the off-diagonal block would stem from asymmetries in the non-zero pattern, where $\mathbf{Y}_{cs}(i, j) \neq 0$ and $\mathbf{Y}_{cs}(j, i) = 0$. Therefore, KLU factorization of $\mathbf{Y}_{cs}$ will always satisfy $\mathbf{F} = \mathbf{0}$.

Considering the entire power system there will be a path between any two nodes unless we end up in a scenario with faults resulting in island to form in the power system. This means that for BTF of the full admittance matrix, there would be only one strongly connected component, which is the entire matrix. This means that KLU for the full system would be similar to using UMFPACK with AMD ordering.

Algorithm 1 however only factorize $\mathbf{Y}_{cs}$, and the cs nodes need not all be connected, since they can be connected through vs nodes. Figure 2.3 shows an example of a network divided in to cs nodes and vs nodes. $\mathbf{Y}_{cs}$ for this system will have 2 connected components marked on the figure with dotted lines.

2.3 Implementation and test

Algorithm 1 is implemented in MATLAB and test are done on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.50GHz. Table 2.1 shows the different test systems that will be used in this thesis. The Pegase and Polish-Winter systems can be found in MATPOWER [41], the PTI systems are included in the PSS®E 33.0 examples and Nordic32 can be found in [42]. EECC-PSSE-33-0 is a representation of the American Eastern interconnection.

The two largest systems will be omitted in the test of the different factorization method due to the high runtimes required for these.

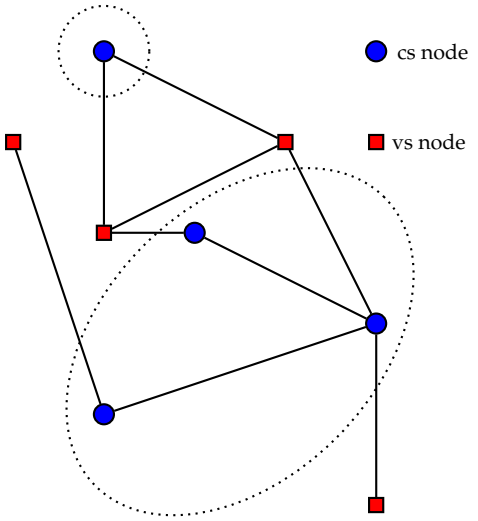


Figure 2.3: Example of a network divided in to cs nodes and vs nodes. The dotted lines represent the strongly connected components of $G(\mathbf{Y}_{cs})$

Table 2.1: Test systems

Case	no. of buses	no. of vs nodes	non-zeros in $\mathbf{Y}$
Nordic32	46	20	160
Pegase89	89	12	501
Pegase1354	1354	260	4774
PTI-WECC-1648	1648	313	6680
Polish-Winter99	2383	327	8155
Polish-Winter03	2746	374	9344
Pegase2869	2869	510	10805
Polish-Winter07	3012	347	10144
PTI-EECC-7991	7917	1325	32211
Pegase9241	9241	1445	37655
Pegase13659	13659	4092	50909
EECC-PSSE-33-0	29827	3780	107527

2.3.1 Tolerance of ILU

Before comparing the different factorization methods it is necessary to determine a suitable tolerance for ILU. This tolerance will effect the sparsity, accuracy and computation time. As a measure of the error the Total Vector Error (TVE) [43] will be used. The error for the Thévenin voltages obtained with ILU can be computed using the results from UMFPACK as reference

$$\text{TVE (\%)} = \sqrt{\frac{(\tilde{X}_r - X_r)^2 + (\tilde{X}_i - X_i)^2}{X_r^2 + X_i^2}} \cdot 100\% \tag{2.19}$$

where $\tilde{X}$ is the estimate from ILU and X is the "true" value from UMFPACK. X_r is the real part and X_i is the imaginary part.

Table 2.2: Performance of ILU for different tolerance levels when computing the Thévenin equivalents for Pegase9241 compared to UMFPACK

Tolerance	Non-zeros **K**	Runtime (s) Algorithm 1	Runtime (ms) V_{th}	Max TVE (%) V_{th}
UMFPACK	42515618	11.95	128.08	-
10^{-3}	9126625	2.14	35.69	4540.43
10^{-4}	29047282	7.13	92.22	14.80
10^{-5}	37618552	17.32	119.92	1.89
10^{-6}	42101927	29.48	126.32	0.27
10^{-7}	42423098	43.80	125.80	0.029
10^{-8}	42498272	47.31	129.70	$2.08{\cdot}10^{-3}$
10^{-9}	42506413	50.67	139.04	$2.28{\cdot}10^{-4}$

Table 2.2 shows the sparsity of ILU for different tolerance levels as well as the resulting runtime of Algorithm 1, runtime for computing Thévenin voltages and the maximum TVE of the Thévenin voltages for the system Pegase9241. As a reference the same values are given for UMFPACK.

Increasing the tolerance increases the sparsity of the coefficient matrix, which reduces the runtime for computing the Thévenin voltages but also the accuracy. For a tolerance larger than 10^{-6} the sparsity of **K** obtained with ILU is approaching that obtained with UMFPACK and the advantage of using ILU disappears.

At a tolerance of 10^{-4} ILU shows runtimes for Algorithm 1, that are comparable to UMFPACK for all systems. For Pegase9241 the runtime is lower, however the errors in Thévenin voltages are up to 15%. An error in Thévenin voltage of a few percent might be accounted for by defining an appropriate trigger-margin for the stability indicators. A suitable tolerance for ILU must therefore satisfy that there is an advantage in terms of runtime when choosing ILU over UMFPACK, and the resulting inaccuracy should be a few percent at most. On this basis a tolerance of 10^{-5} is chosen.

The accuracy of ILU for all systems with the chosen tolerance is shown in Figure 2.4. The mean and maximum TVE of the Thévenin voltages between ILU and UMFPACK is plotted for all systems. The mean value of TVE is in general small, which means that there are few large errors. The largest vector error is smaller than 2% for all systems.

2.3.2 Comparison of factorization methods

After choosing a tolerance of ILU it is possible to compare the different factorization methods. They are compared in terms of runtime, accuracy and fill-in of the coefficient matrix **K**.

Figure 2.5 shows the runtime of Algorithm 1 for each factorization method. The total runtime of KLU is a faster than UMFPACK, however in general the runtimes are quite close. ILU for the chosen tolerance is slower than both UMFPACK and KLU except for the smallest system. The runtime seems to be very dependent on the structure of the system. As an example PTI-EECC-7991 takes longer than Pegase9241 even though the system is smaller

Figure 2.6 shows the runtime for the factorization step (Step 1) of Algorithm 1. KLU consistently performs the factorization almost twice as fast as UMFPACK and is the fastest factorization method

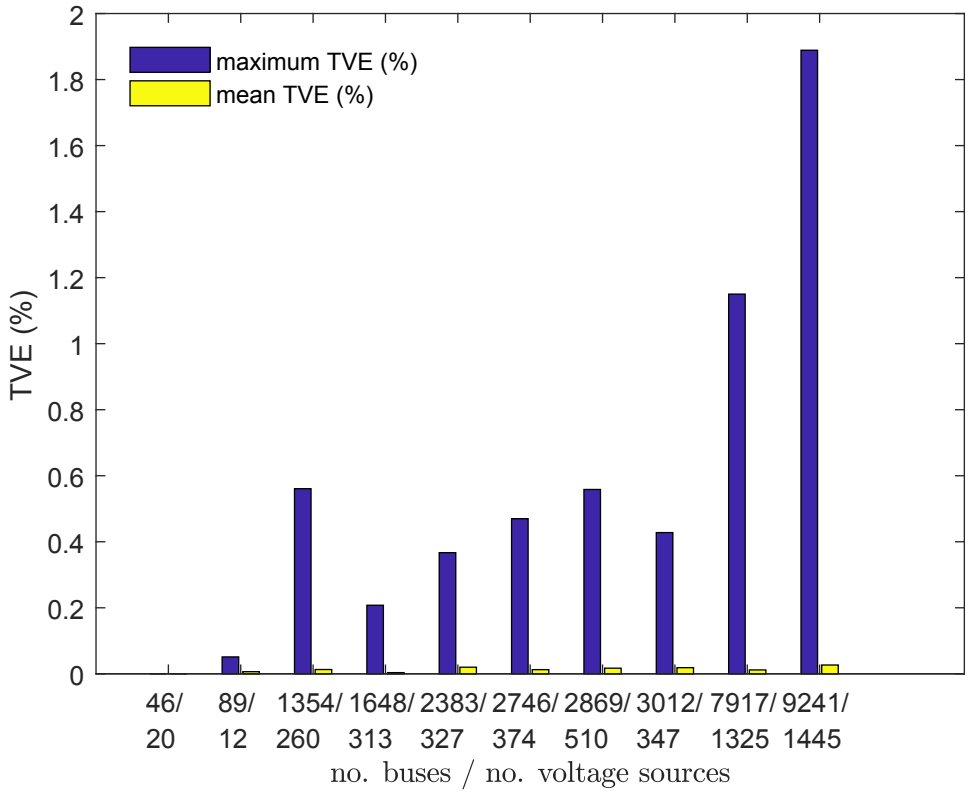


Figure 2.4: Maximum and mean TVE (%), when comparing Thévenin voltages from ILU to UMFPACK. The tolerance of ILU is set to 10^{-5} [35].

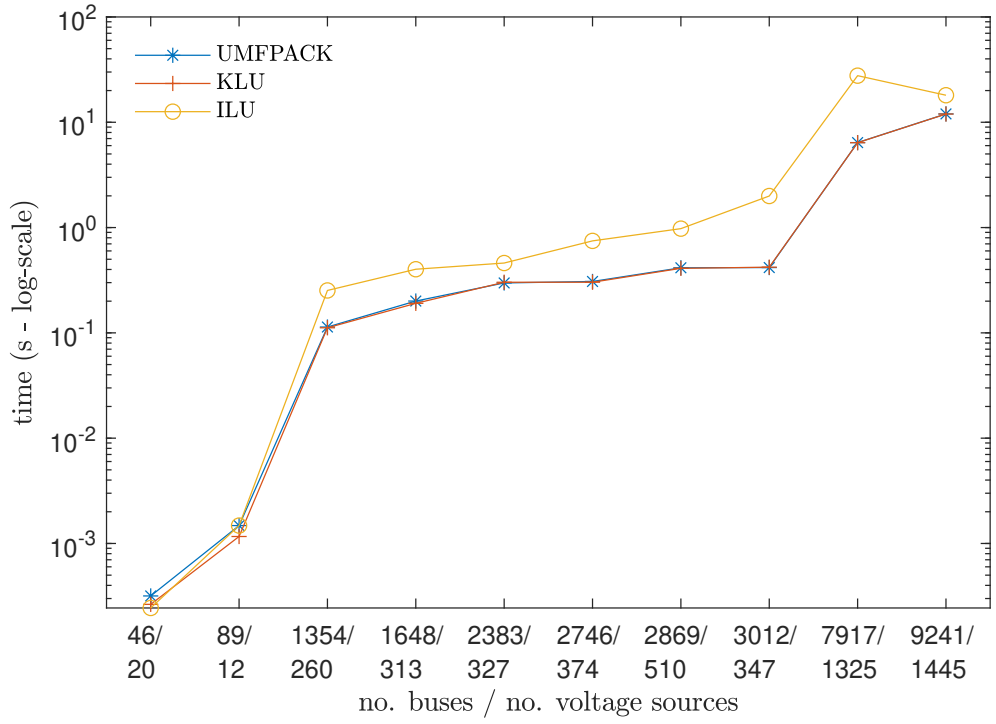


Figure 2.5: Runtime of Algorithm 1 for each factorization method.

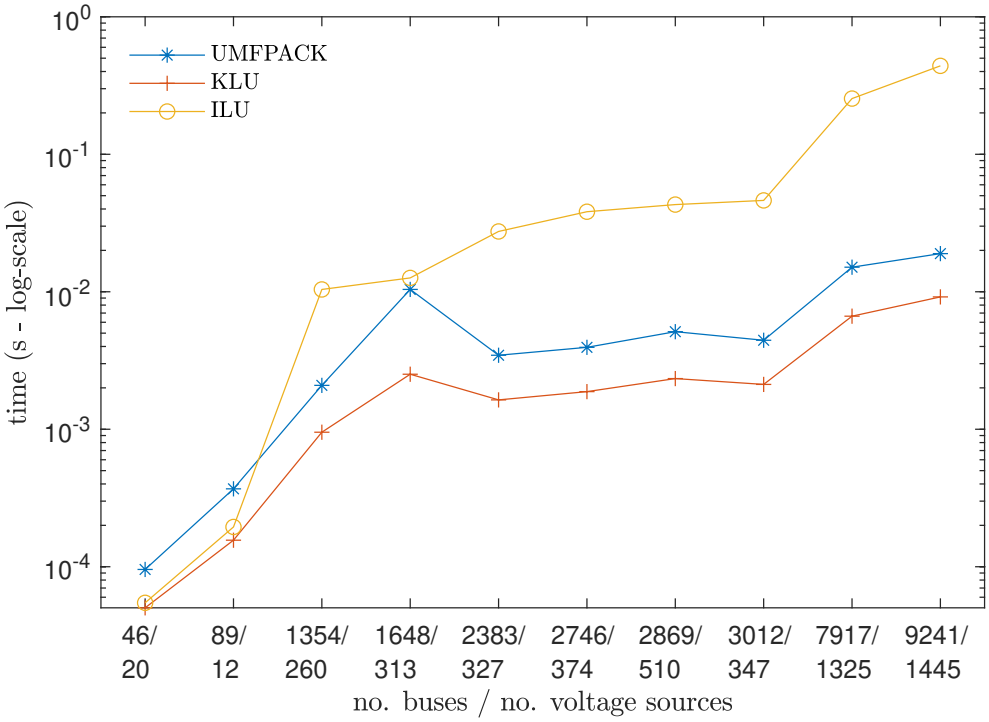


Figure 2.6: Runtime of factorizing $\mathbf{Y}_{cs}$ for each factorization method.

in all cases. When comparing the runtimes in Figure 2.5 to the runtime for the factorization in Figure 2.6 it is evident that the time spent on factorization is negligible compared to the total runtime of Algorithm 1. Further investigation showed that a majority of time was spend on Step 4 i.e. determining $\mathbf{Z}_{cs}$.

For all cases KLU and UMFPACK are almost identical in both runtime and accuracy. The largest difference between the computed Thévenin voltages for the two was 10^{-13}, which could be due to numerical cancellation, as it is close to machine precision. The larger errors seen for ILU is explained by the inaccuracies in Thévenin voltages seen in Figure 2.4.

One advantage of ILU is the increased sparsity of the coefficient matrices. Table 2.3 shows the number of non-zeros of the coefficient matrix $\mathbf{K}$ for each factorization method. For small systems ILU does not provide additional sparsity however as the systems grows larger it significantly reduces fill-in. The difference with respect to sparsity between KLU and UMFPACK is insignificant and can be due to numerical cancellation.

Reduced fill-in will lower the runtime of the Thévenin voltage calculations. Figure 2.7 shows the runtime for computing the Thévenin voltage using the coefficient matrix computed in Algorithm 1. For all systems the runtime of computing the Thévenin voltages was smaller for ILU than for UMFPACK and KLU, however the difference is quite small. If coefficients were to be used for contingency analysis as in [19] the computation time for the contingency assessments could be reduced using ILU. However, the runtime of Algorithm 1 is also increased significantly using ILU.

For Pegase9241 the runtime for computing Thévenin voltages is reduced by 8ms, while the runtime of Algorithm 1 is 5.45s slower Table 2.2. This means that the extra runtime for Algorithm 1 in the case of ILU requires the Thévenin voltage to be computed at least 675 times for ILU to be faster.

Table 2.3: Non-zeros in coefficient matrix, **K**, for each factorization method

Case	Non-zeros UMFPACK	Non-zeros KLU	Non-zeros ILU
Nordic32	548	547	549
Pegase89	7668	7663	7664
Pegase1354	1120074	1120078	1025267
PTI-WECC-1648	1706590	1706580	1694777
Polish-Winter99	2825724	2825717	1835089
Polish-Winter03	3027546	3027555	2577363
Pegase2869	2961791	2961793	2790832
Polish-Winter07	4206611	4206582	4095535
PTI-EECC-7991	44852964	44852962	31146092
Pegase9241	42515624	42515659	37618552

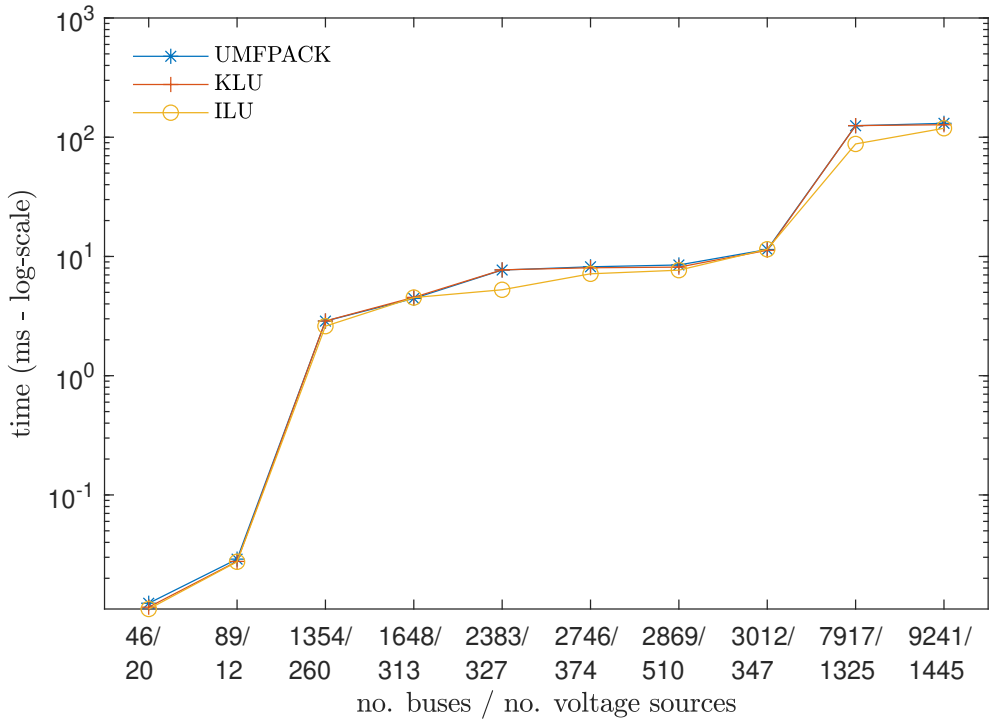


Figure 2.7: Runtime of calculating the Thévenin voltage V_{th} using the coefficient matrix **K** for each factorization method.

This combined with the fact that there will be an error when using an approximation for the LU factorization severly limits the applicability of ILU.

2.3.3 Accuracy of stability margins

Another measure of accuracy is to compare the ability to predict instability issues. To do this two measures of instability are introduced:

- L-index

- Aperiodic Small-Signal Rotor Angle Stability (ASSRAS) margin

The L-index is a local voltage stability margin for loads [44]. For a node i the local indicator L_i is defined by the node voltage $\overline{V}_i$ and the Thévenin voltage $\overline{V}_{th,i}$ seen from node i

$$L_i = \left| 1 - \frac{\overline{V}_{th,i}}{\overline{V}_i} \right|. \tag{2.20}$$

For stable situations $L_i \leq 1$ must not be violated for any node i. Hence the global indicator for voltage stability of the entire system is given as the maximum of all L_i for the cs nodes that are loads

$$L = \max_{i \in csload} \{L_i\}, \tag{2.21}$$

Voltage instability may be inferred in the case where $L > 1$.

In [45] a power margin of the injected power P_{inj} to the maximum power injection $P_{inj,max}$ is defined. This margin describes the distance from a generator's operating point to the stability boundary of ASSRAS. In [46] this margin is reformulated in terms of voltages instead of impedances. A percentage margin to the maximum injectable power is defined as

$$\%\Delta P_{inj} = \frac{P_{inj,max} - P_{inj}}{P_{inj,max}} \cdot 100\% \tag{2.22}$$

$$= \frac{\cos(\delta + \phi_{th}) + 1}{1 + \frac{V}{V_{th}} \cos \phi_{th}} \cdot 100\%, \tag{2.23}$$

where the generator is represented as a voltage source $V\angle\delta$ and the remaining grid by its Thévenin equivalent with a Thévenin equivalent voltage of magnitude (V_{th} and Thévenin equivalent impedance $\theta_{th}, Z_{th}\angle\phi_{th}$. $\overline{V}_{th}$ is the phase angle reference.

If, for any generator, $\%\Delta P_{inj} < 0$, that generator will begin to lose synchronism, which may destabilize the entire system. Therefore, the overall systems ASSRAS margin is defined as the minimum $\%\Delta P_{inj}$.

Table 2.4 and 2.5 show the computed L-index and ASSRAS margin for UMFPACK and the difference between this and the values computed by KLU and ILU. ILU with the chosen tolerance is close to the results from UMFPACK, while KLU has an accuracy, that can be considered to be the same as for UMFPACK. In all cases ILU predicted the same node to be the critical node i.e. the node with the smallest margin to instability as both UMFPACK and KLU.

Therefore the resulting margin is not that different using ILU however there is still a small error using this.

2.3.4 Density of coefficient matrix

The impedance matrix $\mathbf{Z}_{cs}$ is part of the computations for each of the elements in the coefficient matrix $\mathbf{K}$ and hence an essential part of Algorithm 1. The density of the impedance matrix will therefore be a determining factor for the density of the coefficient matrix since the remaining coefficients are quite sparse.

The density of $\mathbf{Z}_{cs}$ can be determined theoretically by using BTF of $\mathbf{Y}_{cs}$, where the only elements will be the diagonal blocks as determined in Section 2.2.1. These blocks have the strong Hall property meaning they have full structural rank [39]. Inverting $\mathbf{Y}_{cs}$ will be the same as inverting

Table 2.4: Voltage stability index for loads (L-index)

Case	L-index UMFPACK	Difference KLU	Difference ILU
Nordic32	0.183	0	0
Pegase89	0.316	0	$1.25{\cdot}10^{-4}$
Pegase1354	0.212	0	0
PTI-WECC-1648	0.283	$-1.7{\cdot}10^{-15}$	$-1.9{\cdot}10^{-4}$
Polish-Winter99	0.066	$-1.2{\cdot}10^{-14}$	$7.39{\cdot}10^{-5}$
Polish-Winter03	0.105	$5.4{\cdot}10^{-15}$	$1.01{\cdot}10^{-4}$
Pegase2869	0.163	0	$6.04{\cdot}10^{-4}$
Polish-Winter07	0.075	$2.9{\cdot}10^{-15}$	$-6.3{\cdot}10^{-4}$
PTI-EECC-7991	0.311	0	$-1.1{\cdot}10^{-4}$
Pegase9241	0.176	0	$-2.17{\cdot}10^{-3}$

Table 2.5: ASSRAS margin for generators ($\min \%\Delta P_{inj}$)

Case	$\min \%\Delta P_{inj}$ UMFPACK	Difference KLU	Difference ILU
Nordic32	38.01	0	0
Pegase89	94.33	0	0.004
Pegase1354	81.00	0	0.018
PTI-WECC-1648	32.82	0	0
Polish-Winter99	81.42	$5.0{\cdot}10^{-13}$	0.026
Polish-Winter03	88.79	$4.0{\cdot}10^{-13}$	$9.05{\cdot}10^{-4}$
Pegase2869	63.41	0	0
Polish-Winter07	79.93	$-7.7{\cdot}10^{-13}$	0.009
PTI-EECC-7991	44.54	0	$1.39{\cdot}10^{-4}$
Pegase9241	62.84	0	0

each of these strong Hall blocks and the inverse of a strong Hall matrix has no zero entries, when ignoring numerical cancellation. Therefore, the size of the blocks determines the density of $\mathbf{Z}_{cs}$.

$$\text{density} = \sum_i n_i^2, \tag{2.24}$$

where n_i is the size of the quadratic blocks.

The number of non-zeros is predetermined by the structure of the matrix and therefore the number of non-zeros should be roughly the same, when using any direct method i.e. UMFPACK and KLU, whereas an incomplete method such as ILU can have fewer entries.

For each of the systems in Table 2.1 the theoretical number of non-zeros in $\mathbf{Z}_{cs}$, determined by the size of the blocks in BTF, is computed and shown in Table 2.6. Next to this the actual density of $\mathbf{Z}_{cs}$, the density of $\mathbf{K}$ using KLU and the size of the largest block in BTF for $\mathbf{Y}_{cs}$ is shown.

The theoretical density is the same as the actual density of $\mathbf{Z}_{cs}$ for all systems. By including more digits it is possible to see a small difference. The density of $\mathbf{K}$ is not far from the density of $\mathbf{Z}_{cs}$,

Table 2.6: Size of largest block compared to entire matrix, theoretical and actual density of $\mathbf{Z}_{cs}$ and actual density of $\mathbf{K}$ for test systems.

Case	Largest block (%) $\mathbf{Y}_{cs}$	Theoretical density (%) $\mathbf{Z}_{cs}$	Density (%) $\mathbf{Z}_{cs}$	Density (%) $\mathbf{K}$
Nordic32	25	27.2	27.2	27.1
Pegase89	97.4	97.4	97.4	96.7
Pegase1354	65.9	66.0	66.0	61.2
PTI-WECC-1648	63.3	64.1	64.1	62.9
Polish-Winter99	48.0	48.4	48.4	49.8
Polish-Winter03	38.7	39.3	39.3	40.2
Pegase2869	38.4	40.3	40.3	36.0
Polish-Winter07	45.6	46.0	46.0	46.4
PTI-EECC-7991	72.7	73.1	73.1	71.6
Pegase9241	54.7	55.3	55.3	49.8
Pegase13659	99.7	99.7	99.7	99.0
EECC-PSSE-33-0	97.0	97.0	97.0	96.1

which makes sense since this is used in the computations of all blocks in $\mathbf{K}$. The size of the largest block in $\mathbf{Y}_{cs}$ is very close to the density of $\mathbf{Z}_{cs}$, which shows that there is one large block dominating the computations. For three of the systems the size of the largest block is more than 90%. For these systems the advantage of using KLU over UMFPACK is small since the large block is almost the same size as the entire system and the resulting matrices have a large amount of fill-in making them in-effective to compute.

2.4 Conclusion

The influence of using different factorization methods on Thévenin equivalent computations was investigated. The results show that the chosen factorization method has little impact, since the factorization step is a negligible part of Algorithm 1. Investigation show that the computational heavy part is determining $\mathbf{Z}_{cs}$. One approach to remedy this would be to take advantage of the backwards solve of KLU and just use the factorization directly instead of computing $\mathbf{Z}_{cs}$. This is similar to the approach in [26], where good performance is achieved.

The incomplete factorization method ILU was able to compute stability indicators with an accuracy close to that of the direct methods UMFPACK and KLU. However using ILU involves a trade-off between accuracy and sparsity, where sparsity also means lower runtimes. For the systems investigated the runtimes for computing the coefficients were generally higher than that of the direct methods, while there was also an error. Therefore ILU has limited applicability in this setting.

Furthermore, it was shown how the density of $\mathbf{Z}_{cs}$ and in consequence the coefficient matrix $\mathbf{K}$ could be predetermined by the BTF of $\mathbf{Y}_{cs}$ and that one large block in $\mathbf{Y}_{cs}$ dominates the computations.

CHAPTER 3

Factor-Solve Method for Thévenin Computations

This chapter introduces the *factor-solve* method for determining Thévenin equivalents for an entire system split in to cs nodes and vs nodes. The approach in the method is developed using the conclusions in Chapter 2. It was concluded that the computationally heavy part of the method is determining the inverse of the admittance matrix of the cs nodes and the *factor-solve* method therefore avoids computing this. The method is tested and parallelized and will be compared to the reference method given in Chapter 2. The complexity of both of these methods is investigated as well as their memory requirements. The main results of this chapter is based on **[Pub. B]**.

3.1 Reference method

The method for computing Thévenin equivalents introduced in Chapter 2 (Algorithm 1) will be considered as the reference method for the work presented in this Chapter. KLU will be used as the factorization method in Algorithm 1, since it was shown to have a little better performance compared to UMFPACK and since it will be used in the *factor-solve* method introduced later. To do a proper comparison of the reference method to the *factor-solve* method the complexity will be determined.

3.1.1 Complexity

All computations of Algorithm 1 is done with sparse matrices and therefore all computations are dependent on the number of non-zeros i.e. the density of the matrices involved. $|cs|$ is the number of cs nodes and $|\mathbf{X}|$ is the number of non-zeros in the matrix $\mathbf{X}$.

- The factorization step and the amount of fill-in generated scales close to linear with system size in this context [26].
- Inverting the sparse LU factors will at maximum have one computation per non-zero element in the matrix for each column of the identity matrix. This means that solve($\mathbf{L}_{cs}, \mathcal{I}$) and solve($\mathbf{U}_{cs}^T, \mathcal{I}$) has a complexity of $O(|\mathbf{L}_{cs}||cs|) = O(|\mathbf{U}_{cs}||cs|)$.
- Computing $\mathbf{Z}_{cs}$ will have the complexity $O(|\mathbf{L}_{\mathbf{Z}_{cs}}||cs|)$, since the non-zeros in $\mathbf{L}_{\mathbf{Z}_{cs}}$ is multiplied by the $|cs|$ columns in $\mathbf{U}_{\mathbf{Z}_{cs}}$.

Empirically $|\mathbf{U}_{cs}| \leq |\mathbf{L}_{\mathbf{Z}_{cs}}|$, and therefore inversion will be computationally cheaper than computing $\mathbf{Z}_{cs}$. The remaining computations of Algorithm 1 will be of lower complexity, since they involve at least one sparse matrix. Hence the complexity of the algorithm will be $O(|\mathbf{L}_{\mathbf{Z}_{cs}}||cs|)$. Assuming

that $|\mathbf{L}_{\mathbf{Z}_{cs}}|$ ($\simeq |\mathbf{U}_{\mathbf{Z}_{cs}}|$) scale close to linear with system size like it's the case (in practise) with the number of non-zeros and fill-in generated in the factorization the complexity will be $O(|cs|^2)$.

Determining Thévenin voltages (2.10) is $O(|\mathbf{K}|)$, since it is sparse matrix-vector multiplication. Therefore, the computation of Thévenin voltages will depend on the density of the coefficient matrix. As concluded in Section 2.3.4 the density of the coefficient matrix depend on the density of $\mathbf{Z}_{cs}$, which is determined by the size of the blocks in the BTF for $\mathbf{Y}_{cs}$. Therefore, the structure of the system will determine the computational time for Thévenin voltages.

3.2 Factor-solve method

Performance of Algorithm 1 and the computation of Thévenin voltages is dissatisfying and doesn't scale well with system size, but using some of the ideas from the conclusion in Chapter 2 there is potential for improvements. KLU solves a system by using block back substitution and this solver makes it possible to find the Thévenin voltages without computing the coefficient matrix.

The equations from (2.10) is revisited and written as

$$V_{th,cs} = \left(\mathbf{Z}_{cs} - \mathcal{D}\left(Z_{th,cs}\right)\right) I_{cs} - \mathbf{Z}_{cs}\mathbf{Y}_{v\to c}V_{vs} \tag{3.1}$$

$$V_{th,vs} = \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}I_{cs} + \left(\mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq}\right)V_{vs} \tag{3.2}$$

By using the expression for $\mathbf{Y}_{eq}$ and $\mathbf{Q}_{ac}$ given in (2.5)-(2.6) and moving some terms around the following expression emerges

$$V_{th,cs} = \mathbf{Z}_{cs}\left(I_{cs} - \mathbf{Y}_{v\to c}V_{vs}\right) - Z_{th,cs}I_{cs} \tag{3.3}$$

$$V_{th,vs} = V_{vs} - Z_{th,vs}\left(\mathbf{Y}_{vs}V_{vs} + \mathbf{Y}_{c\to v}\mathbf{Z}_{cs}\left(I_{cs} - \mathbf{Y}_{v\to c}V_{vs}\right)\right) \tag{3.4}$$

Defining $\tilde{V}$ as

$$\tilde{V} = \mathbf{Z}_{cs}\left(I_{cs} - \mathbf{Y}_{v\to c}V_{vs}\right) \tag{3.5}$$

and inserting this in to (3.3) and (3.4) gives

$$V_{th,cs} = \tilde{V} - Z_{th,cs}I_{cs} \tag{3.6}$$

$$V_{th,vs} = V_{vs} - Z_{th,vs}\left(\mathbf{Y}_{vs}V_{vs} + \mathbf{Y}_{c\to v}\tilde{V}\right) \tag{3.7}$$

The expensive part of the computations is contained in $\tilde{V}$. The remaining part of (3.6)-(3.7) is either subtraction of vectors, multiplication of vectors or sparse matrix-vector multiplication, which are all linear in complexity.

(3.5) can be written as

$$\mathbf{Y}_{cs}\tilde{V} = I_{cs} - \mathbf{Y}_{v\to c}V_{vs} \tag{3.8}$$

$\tilde{V}$ can therefore be determined by solving the above linear equation. Solving $\mathbf{Y}_{cs}x = b$ for x can be determined by block back substitution using the KLU factorization. This will be defined as klu(LU, b). $\tilde{V}$ can then be calculated by klu($LU, I_{cs} - \mathbf{Y}_{v\to c}V_{vs}$). This makes it possible to determine the Thévenin voltages using only the factorization of $\mathbf{Y}_{cs}$ and the Thévenin impedances. This means, that the coefficient matrix $\mathbf{K}$ no longer needs to be computed.

The Thévenin impedances for cs nodes and vs nodes are defined in (2.7). The diagonal of $\mathbf{Z}_{cs}$ and the diagonal of the Schur complement $\mathbf{Y}_{eq}$ is needed in these computations.

The diagonal of $\mathbf{Z}_{cs}$ determines the Thévenin impedances for cs nodes. This is computed by multiplying the rows and columns of $\mathbf{U}_{\mathbf{Z}_{cs}} = \mathbf{L}_{cs}^{-1}$ and $\mathbf{L}_{\mathbf{Z}_{cs}} = \mathbf{U}_{cs}^{-1}$ i.e.

$$Z_{th,cs,k} = \mathbf{L}_{\mathbf{Z}_{cs}}(k,:)\mathbf{U}_{\mathbf{Z}_{cs}}(:,k) \quad \forall k \in cs, \tag{3.9}$$

where $\mathbf{L}_{\mathbf{Z}_{cs}}(k,:)$ is the k'th row of $\mathbf{L}_{\mathbf{Z}_{cs}}$ and $\mathbf{U}_{\mathbf{Z}_{cs}}(:,k)$ is the k'th column of $\mathbf{U}_{\mathbf{Z}_{cs}}$.

The Thévenin impedances for the vs nodes are given as the inverse of the diagonal elements in the Schur complment $\mathbf{Y}_{eq}$.

$$Z_{th,vs,k} = \mathbf{Y}_{eq}(k,k)^{-1} \quad \forall k \in vs, \tag{3.10}$$

which is scalar inversion.

Using (2.5) each element of the diagonal is determined as

$$\mathbf{Y}_{eq}(k,k) = \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}(:,k) \tag{3.11}$$

As with the Thévenin voltages block back substitution can be used to determine part of the equation. The Thévenin impedances for the vs nodes are then found as

$$\hat{\mathbf{U}}(:,k) = \mathbf{Z}_{cs}\mathbf{Y}_{v\to c}(:,k) \leftarrow \text{klu}\left(LU, \mathbf{Y}_{v\to c}(:,k)\right) \tag{3.12}$$

$$\mathbf{Y}_{eq}(k,k) = \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\hat{\mathbf{U}}(:,k) \tag{3.13}$$

$$Z_{th,vs,k} = \mathbf{Y}_{eq}(k,k)^{-1} \tag{3.14}$$

The computations for the vs nodes is similar to the computations given in [26]. This method was called *reduce-factor-solve* and following this notation the new method will be called a *factor-solve* method. The method factorize part of the system and then solves using the factorization to compute the solution. This *factor-solve* approach is used to find the Thévenin impedances for the vs nodes and to find the Thévenin voltages for the entire system. The method in [26] has an initial reduce step to eliminate part of the nodes. However, this does not apply here, since the Thévenin equivalents for the entire system is determined, and therefore there isn't any nodes that can be eliminated prior to the computations.

Algorithm 2 factorize $\mathbf{Y}_{cs}$ and compute the Thévenin impedances. The two loops that determines the Thévenin impedances can be run in parallel.

The *factor-solve* method ensures that no computation time is spent on computing the dense coefficient matrix. Furthermore, the computation of the Thévenin voltages is altered from matrix-vector multiplication with a sparse matrix, that is quite dense, to a block back substitution and matrix-vector multiplications with very sparse matrices.

3.2.1 Complexity

The complexity of Algorithm 2 is split in to a sequential and a parallel part.

The first 3 steps are sequential. The inversions are $O(|\mathbf{L}_{cs}||cs|)$ and $O(|\mathbf{U}_{cs}||cs|)$, respectively, and the fill-in scales linearly with system size meaning the complexity is $O(|cs|^2)$. The factorization has negligible runtime compared to this as it is close to linear, and therefore the serial part will have a complexity of $O(|cs|^2)$.

The two loops can be run sequential or in parallel. The first loop will be $|cs|$ multiplications of vectors of length $|cs|$ giving a complexity of $O(|cs|^2)$. The second loop has $|vs|$ computations of a

Algorithm 2 Thévenin equivalents

$\mathbf{L}_{cs}, \mathbf{U}_{cs} \leftarrow$ factorization of $\mathbf{Y}_{cs}$ {Output: LU}
$\mathbf{U}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{L}_{cs}, \mathcal{I}$)
$\mathbf{L}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{U}_{cs}^T, \mathcal{I}$)^T
for $k = 1..|cs|$ {In parallel} **do**
  $Z_{th,cs,k} \leftarrow \mathbf{L}_{\mathbf{Z}_{cs}}(k,:)\mathbf{U}_{\mathbf{Z}_{cs}}(:,k)$
end for
for $k = 1..|vs|$ {In parallel} **do**
  $\hat{\mathbf{U}}(:,k) \leftarrow$ klu($LU, \mathbf{Y}_{v\to c}(:,k)$)
  $\mathbf{Y}_{eq}(k,k) \leftarrow \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\hat{\mathbf{U}}(:,k)$
  $Z_{th,vs,k} \leftarrow \mathbf{Y}_{eq}(k,k)^{-1}$
end for
$Z_{th} \leftarrow \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}$
return Z_{th} and LU

solve step with linear complexity $O(|cs|)$, multiplication of vectors of length $|cs|$ and inversion of $Y_{eq}(k,k)^{-1}$, which is $O(1)$, as it is inversion of a single number. This gives a complexity of $O(|cs||vs|)$ for the second loop.

When running an algorithm in parallel the runtime is theoretically determined by the number of cores. By Amdahl's law the total computational time will only be limited by the sequential part of the algorithm, since an unlimited number of cores can be added to completely parallelize the loops making the computational time for these negligble. In practise however this will not be possible, since there will be an overhead due to communication between the cores.

The computation of the Thévenin voltages now consists of a solve step using KLU and matrix-vector multiplication with sparse matrices. The block back substitution of KLU is close to linear and so is the matrix-vector multiplications due to the sparsity scaling linear with system size. This gives a complexity of $O(|cs|)$. As comparison the reference method's complexity is $O(|\mathbf{K}|)$, which was shown earlier to be very dependent on the system but in general the matrix was quite dense.

3.3 Implementation and test

The reference method from Algorithm 1 and equation (2.10) and the *factor-solve* method from Algorithm 2 and equations (3.5)-(3.7) is analysed in MATLAB on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. The test systems are the ones given earlier in Table 2.1. The reference method is rerun here such that the runtimes for both methods are computed at the same time on the same machine. Therefore, the runtime might be a little different compared to the values given in Chapter 2.

Table 3.1 shows the runtimes for Algorithm 1 and 2 and Table 3.2 shows the runtimes for computing the Thévenin voltages with the reference and the *factor-solve* method.

The speed-up is calculated as a quantity for the performance of the *factor-solve* method compared to the reference method. This is defined as

$$\text{speed-up} = \frac{t_1}{t_2}, \tag{3.15}$$

where t_1 is the runtime of the reference method and t_2 is the runtime of the *factor-solve* method. The speed-up is shown alongside the runtimes in Table 3.1 and 3.2.

Table 3.1: Runtime of Algorithm 1 and Algorithm 2 and the speed-up.

Case	Runtime (s) Algorithm 1	Runtime (s) Algorithm 2	Speed-up
Nordic32	$2.98{\cdot}10^{-4}$	0.045	0.01
Pegase89	$1.34{\cdot}10^{-3}$	0.050	0.03
Pegase1354	0.11	0.114	0.97
PTI-WECC-1648	0.20	0.544	0.37
Polish-Winter99	0.30	0.241	1.23
Polish-Winter03	0.31	0.319	0.98
Pegase2869	0.41	0.355	1.15
Polish-Winter07	0.42	0.300	1.39
PTI-EECC-7991	6.31	1.745	3.62
Pegase9241	11.58	2.983	3.88
Pegase13659	36.70	6.276	5.85
EECC-PSSE-33-0	253.08	20.592	12.29

Table 3.2: Runtime of computing Thévenin voltages with reference and *factor-solve* method and the resulting speed-up.

Case	Runtime (ms) V_{th} reference	Runtime (ms) V_{th} factor-solve	Speed-up
Nordic32	$1.27{\cdot}10^{-2}$	0.084	0.15
Pegase89	$3.13{\cdot}10^{-2}$	0.101	0.31
Pegase1354	2.74	0.295	9.28
PTI-WECC-1648	4.62	0.731	6.32
Polish-Winter99	7.08	0.457	15.50
Polish-Winter03	8.50	0.540	15.72
Pegase2869	7.65	0.604	12.66
Polish-Winter07	10.51	0.565	18.60
PTI-EECC-7991	118.13	1.473	80.18
Pegase9241	118.95	1.708	69.63
Pegase13659	530.13	2.328	227.75
EECC-PSSE-33-0	2519.75	5.522	456.28

The runtimes for Algorithm 1 and 2 is plotted in Figure 3.1. The runtimes for Algorithm 2 is slower for the smaller systems and faster for the largest systems. As analysed the complexity of Algorithm 2 is seen to be close to quadratic if the smallest systems are ignored. These systems are quite small and therefore doing normal matrix computations is faster than using loops in MATLAB, which is computationally heavy when the system is so small. The complexity for Algorithm 1 was also analysed to be quadratic under the assumption that the fill-in of $\mathbf{L}_{\mathbf{Z}_{cs}}$ scales close to linear with system size. This type of complexity is what is seen in Figure 3.1. Hence as expected the complexity is the same for Algorithm 1 and 2.

Figure 3.2 shows the runtimes for computing the Thévenin voltages with the reference and

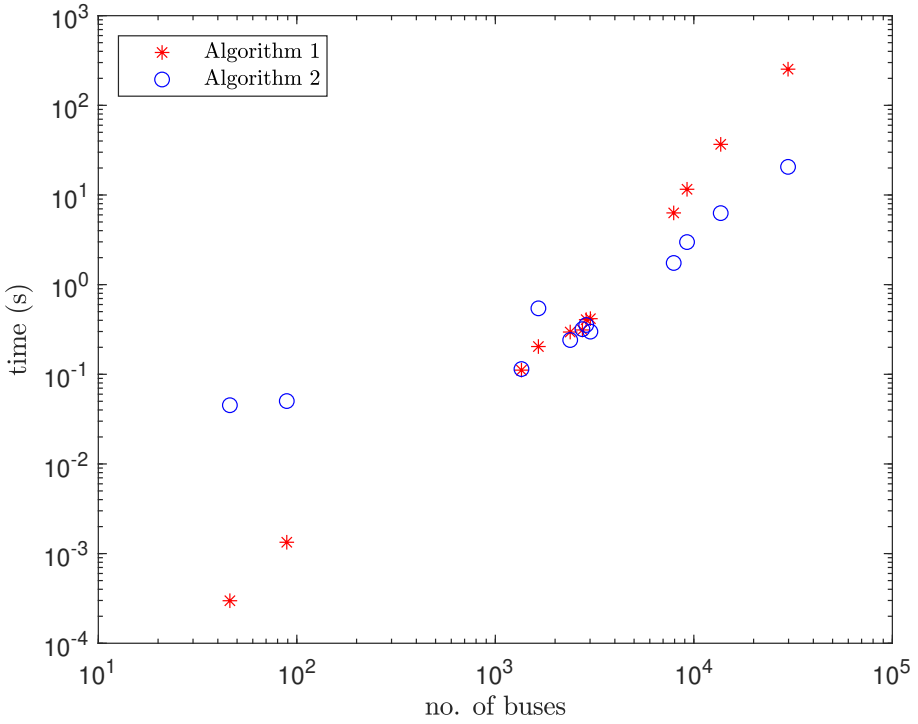


Figure 3.1: Runtime for the Algorithm 1 and 2 depending on the number of buses. The plot is logarithmic.

factor-solve method. The Thévenin voltages computations for the reference method was earlier analyzed to have complexity of $O(|\mathbf{K}|)$. The runtimes in Figure 3.2 show that the complexity is close to quadratic. $|\mathbf{K}|$ was seen earlier to be quite dense and therefore quadratic scaling seems reasonable. Increased system size will result in larger matrices and therefore also a larger number of non-zeros. Computing the Thévenin voltages with the *factor-solve* method is seen to have close to linear complexity as analysed earlier. Figure 3.2 shows few points lying outside the linearity. The complexity is only close to linear, since it actually depends on the fill-in in the factorization, which can differ depending on the system structure.

For the smaller systems neither the algorithm nor the calculations of Thévenin voltage for *factor-solve* method receives a speed-up. Systems larger than 1000 buses is sped up in Thévenin voltage computations but only some benefit from Algorithm 2. Algorithm 2 requires large power systems to be faster than Algorithm 1. The complexity is the same, which limits the benefit, however parallelization can be used in Algorithm 2, which will be tested later.

Some systems have a larger speed-up when computing Thévenin voltages with the *factor-solve* method instead of the reference method compared to systems of similar size. The systems Pegase2869 and Pegase9241 both have a lower speed-up compared to similar sized systems. The reason is found in Table 2.6. The coefficient matrix for both Pegase2869 and Pegase9241 is less dense than for the systems of similar size, and therefore weren't as slow with the reference method. The change in complexity from dependence on the density of the coefficient matrix to be close to linear to system size, now gives runtimes, that scale better with system size.

Errors in Thévenin voltages obtained with the two methods is determined in terms of the TVE defined in (2.19). Here the reference method using UMFPACK will be considered the true value and the reference and *factor-solve* method using KLU will be the estimate. The maximum TVE can

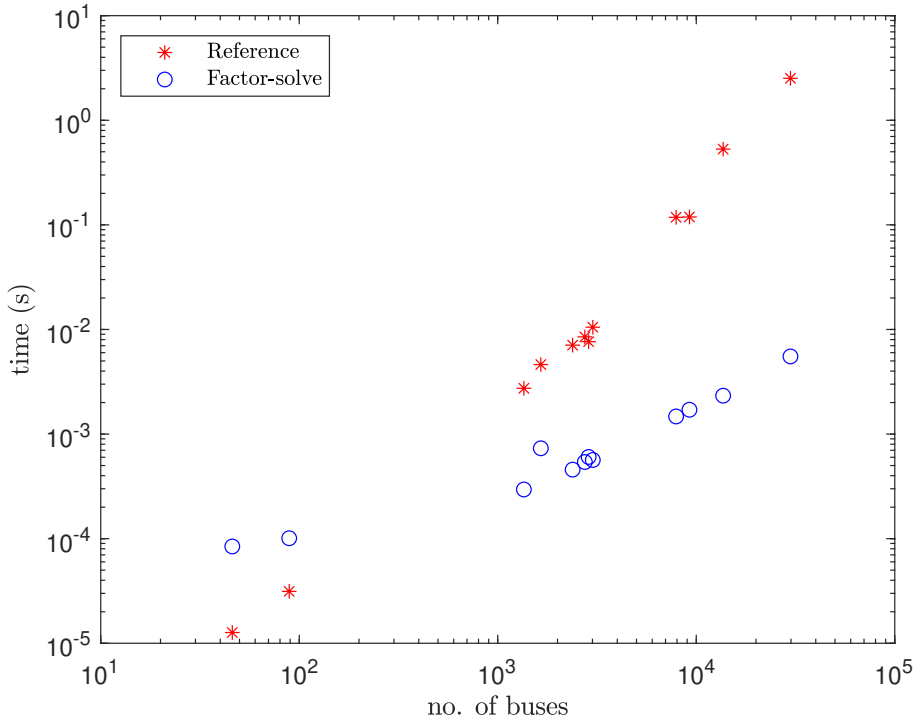


Figure 3.2: Runtime for computing the Thévenin voltages with the reference and *factor-solve* method depending on the number of buses. The plot is logarithmic.

Table 3.3: Maximum TVE (%) for the reference and *factor-solve* method with UMFPACK as the true value.

Case	Max TVE (%) (reference)	Max TVE (%) (*factor-solve*)
Nordic32	$2.43{\cdot}10^{-13}$	$2.23{\cdot}10^{-13}$
Pegase89	$1.71{\cdot}10^{-12}$	$1.71{\cdot}10^{-12}$
Pegase1354	$4.55{\cdot}10^{-12}$	$4.56{\cdot}10^{-12}$
PTI-WECC-1648	$6.98{\cdot}10^{-12}$	$7.02{\cdot}10^{-12}$
Polish-Winter99	$1.95{\cdot}10^{-11}$	$2.79{\cdot}10^{-11}$
Polish-Winter03	$2.98{\cdot}10^{-11}$	$2.98{\cdot}10^{-11}$
Pegase2869	$5.50{\cdot}10^{-12}$	$5.54{\cdot}10^{-12}$
Polish-Winter07	$1.52{\cdot}10^{-11}$	$1.52{\cdot}10^{-11}$
PTI-EECC-7991	$9.26{\cdot}10^{-12}$	$9.17{\cdot}10^{-12}$
Pegase9241	$2.13{\cdot}10^{-11}$	$2.13{\cdot}10^{-11}$
Pegase13659	$2.20{\cdot}10^{-11}$	$2.22{\cdot}10^{-11}$
EECC-PSSE-33-0	$1.30{\cdot}10^{-09}$	$8.01{\cdot}10^{-10}$

be seen in Table 3.3. The two methods differ in values close to machine precision and the accuracy can therefore be concluded to be the same.

Table 3.4: Runtime in seconds for Algorithm 2 run in parallel for different number of cores. Lowest runtime for each system is marked with green.

Case \ Cores	1	2	4	6	8	10	12	14	16	18	20	22	24
Nordic32	0.045	0.054	0.064	0.063	0.072	0.085	0.095	0.107	0.128	0.140	0.153	0.164	0.162
Pegase89	0.050	0.061	0.063	0.065	0.082	0.093	0.118	0.122	0.133	0.151	0.161	0.170	0.177
Pegase1354	0.114	0.101	0.107	0.102	0.117	0.134	0.167	0.192	0.212	0.238	0.256	0.283	0.302
PTI-WECC-1648	0.544	0.302	0.281	0.296	0.369	0.292	0.333	0.428	0.350	0.389	0.579	0.446	0.512
Polish-Winter99	0.241	0.162	0.124	0.121	0.121	0.141	0.177	0.192	0.218	0.252	0.279	0.289	0.350
Polish-Winter03	0.319	0.206	0.165	0.161	0.157	0.155	0.171	0.195	0.217	0.236	0.260	0.277	0.310
Pegase2869	0.355	0.242	0.181	0.164	0.174	0.176	0.190	0.218	0.234	0.256	0.277	0.307	0.321
Polish-Winter07	0.299	0.188	0.152	0.136	0.136	0.156	0.186	0.207	0.226	0.252	0.271	0.307	0.314
PTI-EECC-7991	1.745	0.882	0.550	0.438	0.388	0.358	0.350	0.353	0.344	0.370	0.390	0.412	0.455
Pegase9241	2.983	1.603	0.909	0.707	0.602	0.540	0.513	0.497	0.500	0.518	0.540	0.564	0.600
Pegase13659	6.276	3.334	1.814	1.388	1.173	1.039	0.950	0.907	0.901	0.918	0.917	0.967	0.976
EECC-PSSE-33-0	20.592	9.936	5.439	4.209	3.604	3.216	2.994	2.871	2.759	2.786	2.803	2.848	2.891

3.3.1 Parallelization

The runtime for Algorithm 2 is only a little faster than Algorithm 1 for some of the larger systems and their complexity is both quadratic. However, a clear benefit from the method is that it can easily be run in parallel. The runtime is therefore tested on a machine with 2 CPUs of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz with 12 cores each. The algorithm will be tested on the following number of cores $1, 2, 4, 6, \ldots, 22, 24$.

The runtimes for the systems for the different number of cores is shown in Table 3.4. The number of cores giving the lowest runtime varies for the systems. The smallest systems are fastest using only 1 core and comparing to the runtimes given in Table 3.1 the runtimes seems to be dominated by the overhead from the parallelization. This is expected, since to running a parallel process with `parfor` instead of just using `for` in MATLAB will cause this.

As the system size increases so does the optimal number of cores for the computation. The lowest runtime is for each systems marked with green in the Table 3.4. For Polish-Winter99 and Polish-Winter07 there is 2 runtime that are the lowest in the table. The difference was found using more decimals and therefore only one of them is marked in the table.

The speed-up for Algorithm 2 compared to Algorithm 1 is shown in Figure 3.3. Runtime of Algorithm 2 is decreased considerably. Speed-up increases with system size. For the larger system the speed-up increases up to a factor of over 90 compared to a factor of 12 without parallelization. The 4 smallest systems still does not receive a speed-up due to their size being too small to make up for the overhead. The remaining systems i.e. the middle systems that did not or barely have a speed-up when running Algorithm 2 in serial is now speed-up using parallelization.

The runtime for the optimal number of cores as well as the speed-up compared to Algorithm 1 is seen in Table 3.5. It shows explicitly how the optimal number of cores as well as the speed-up grows with system size. For the largest system EECC-PSSE-33-0 the runtime using the parallelized *factor-solve* method gets as low as 2.8s compared to a runtime of 253s with the reference method. This makes it possible to respond faster to changes in the system topology and makes it viable to be run in real-time.

Figure 3.4 shows the runtime for each part of Algorithm 2 for the optimal number of cores in Table 3.5. Factorization is Step 1 of Algorithm 2, $Z_{th,cs}$ is Step 2-6 and $Z_{th,vs}$ is Step 7-11. Factorization time is the smallest part of the runtime and remains negligible, which fits with it's complexity being close to linear to system size comparing to the remaining being quadratic.

The majority of time is spend on computing the Thévenin impedances for the cs nodes. However, there is also a considerable larger amount of cs nodes, thus when dividing the runtimes with the number of cs nodes and vs nodes the individual runtime for each $Z_{th,cs}$ is lower than for $Z_{th,vs}$. This is shown in Figure 3.5.

Each loop of Algorithm 2 is independent and the optimal number of cores for each loop might not be the same. Therefore, running both loops using the same number of cores might result in the loop with the fewest elements being affected by overhead. This might explain some of the difference between the two, since there is a lot fewer vs nodes than cs nodes. There is also a difference in the computations for the different nodes, which will effect individual runtimes. Computations for cs nodes are also divided in to a sequential part (inversion of $\mathbf{L}$ and $\mathbf{U}$) and a parallel part (the loop multiplying vectors). The sequential part of the computations will not benefit from additional cores.

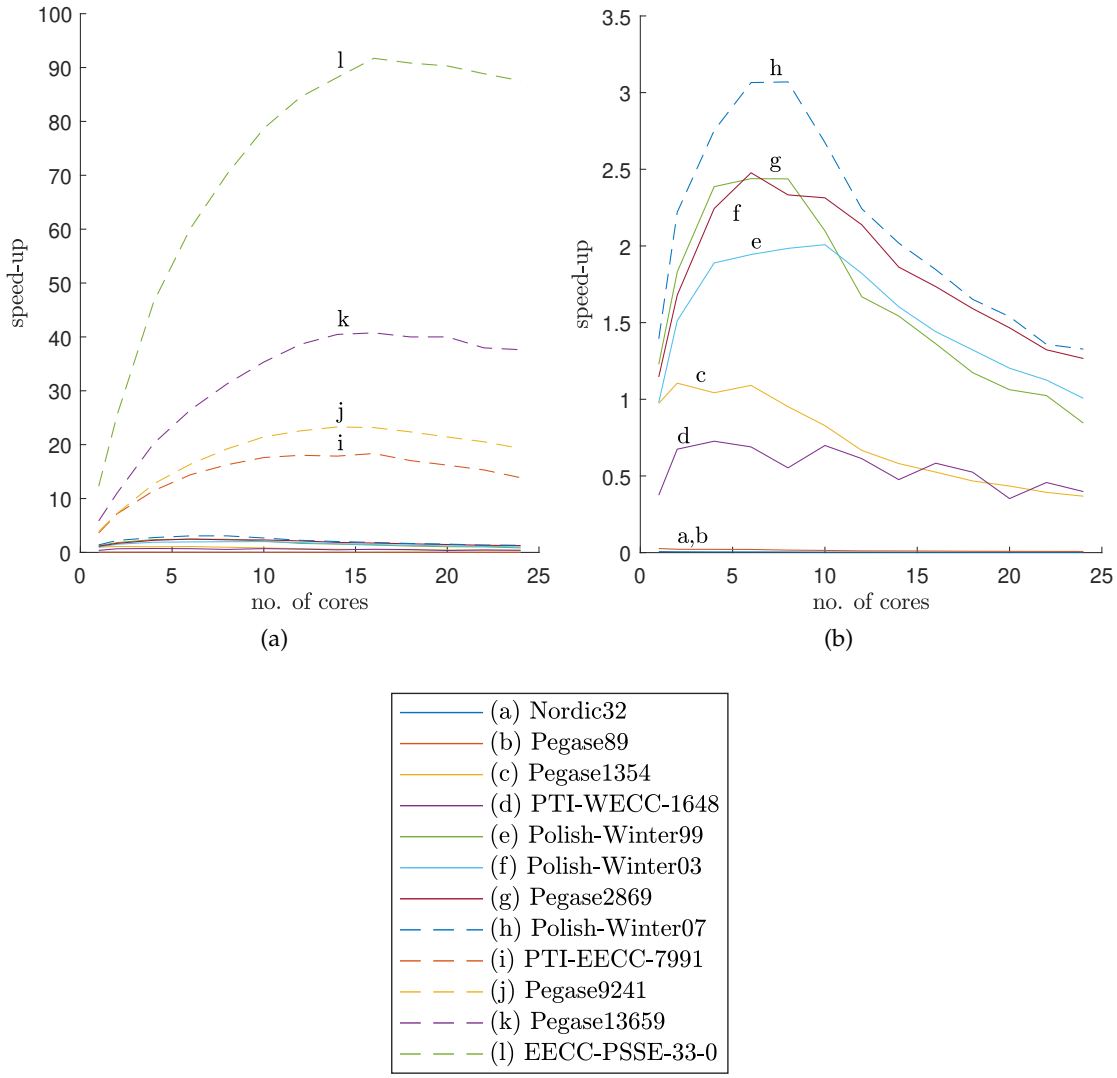


Figure 3.3: Speed-up of Algorithm 2 compared to Algorithm 1 - (a) shows all systems and (b) is a zoom of the smaller systems.

3.3.2 Memory requirements

A benefit from the *factor-solve* method compared to the reference method is that the coefficient matrix is no longer required. Not only is it beneficial to not compute it but furthermore it is beneficial not having to store it, as it requires more memory compared to only using the factorization. The coefficient matrix can be stored either as a sparse matrix or a dense matrix.

Storing a sparse matrix requires storing the non-zero entries and the location of the entry, which is stored in two vectors for the row and column indexes. This gives a formula for the memory required for a matrix of size mxn

$$mem_{\text{sparse}} = 2 \cdot nnz \cdot b_{\text{double}} + nnz \cdot b_{\text{integer}} + (n+1) \cdot b_{\text{integer}}, \tag{3.16}$$

where nnz is the number of non-zeros and b_{double} and $b_{integer}$ is the number of bits used to store doubles and integers respectively. The above formula is for a compressed column storage (CCS) of a sparse matrix for compressed row storage (CRS) replace n with m, since matrices will then be

Table 3.5: Results for the optimal number of cores for each system

Case	Optimal no. of cores	Runtime (s) Algorithm 2	Speed-up Algorithm 2
Nordic32	1	0.045	0.007
Pegase89	1	0.050	0.027
Pegase1354	2	0.101	1.105
PTI-WECC-1648	4	0.281	0.727
Polish-Winter99	6	0.121	2.439
Polish-Winter03	10	0.155	2.008
Pegase2869	6	0.164	2.477
Polish-Winter07	8	0.136	3.070
PTI-EECC-7991	16	0.344	18.354
Pegase9241	14	0.497	23.289
Pegase13659	16	0.901	40.750
EECC-PSSE-33-0	16	2.759	91.715

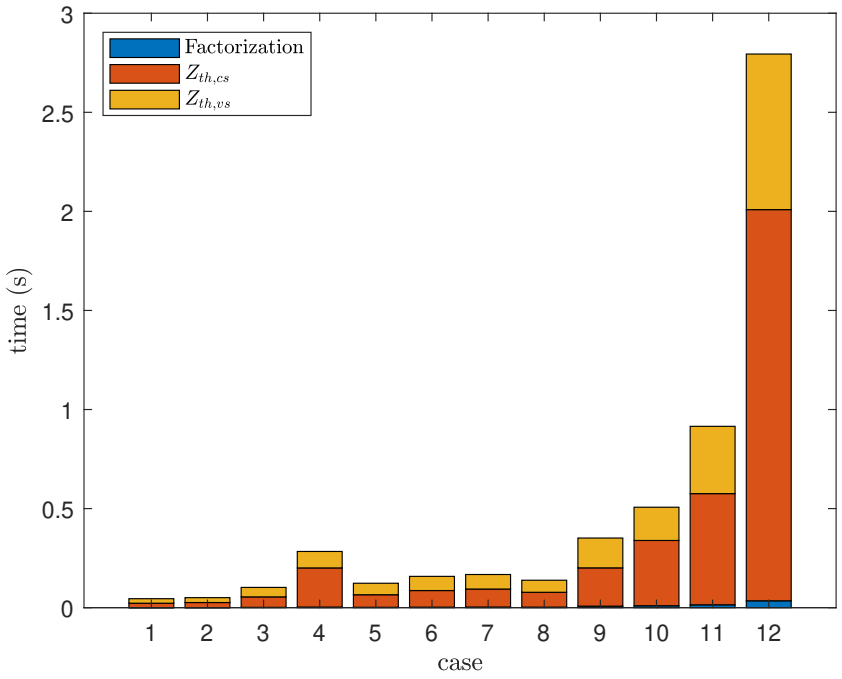


Figure 3.4: Distribution of the runtime of Algorithm 2 on to each part of the algorithm for the optimal number of cores.

read by row first instead of column first. For computations of the memory it doesn't matter if the matrix is stored as CCS or CRS, since all matrices are square i.e. $n = m$. nnz is multiplied by 2 since the matrices contains complex values meaning two doubles are needed to represent them.

Storing a full matrix stores all entries and therefore the memory required will be

$$mem_{\text{full}} = 2 \cdot m \cdot n \cdot b_{\text{double}}. \tag{3.17}$$

Again the coefficient matrix is square hence $n = m$. Therefore

$$mem_{\text{full}} = 2 \cdot n^2 \cdot b_{\text{double}}. \tag{3.18}$$

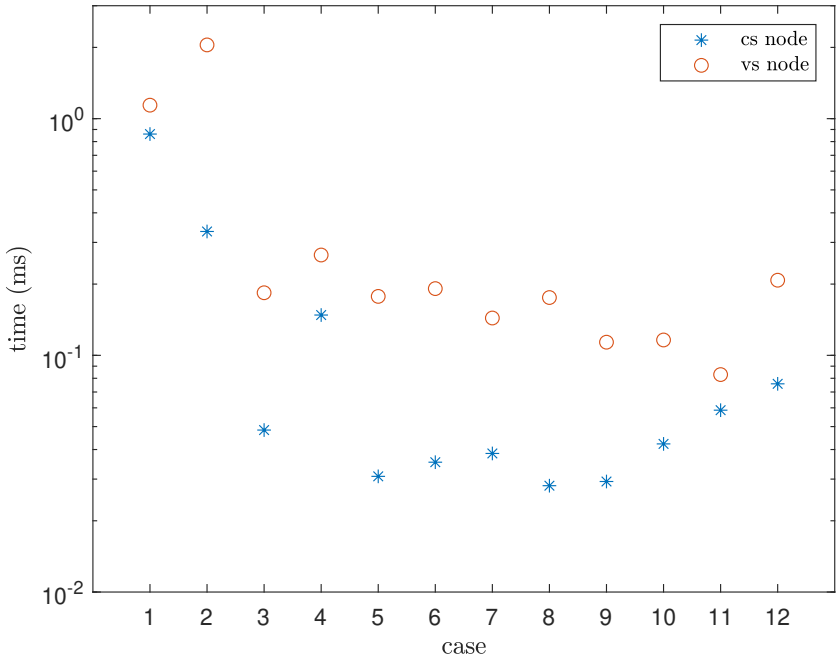


Figure 3.5: Runtime for computing the Thévenin impedances for each cs node and vs node using the *factor-solve* method run in parallel with optimal number of cores.

As mentioned earlier the KLU factorization of a matrix $\mathbf{A}$ is defined as

$$\mathbf{PRAQ} = \mathbf{LU} + \mathbf{F} \tag{3.19}$$

$\mathbf{P}$, $\mathbf{Q}$ are permutations stored as vectors, $\mathbf{R}$ is a scaling matrix optimally stored as a vector, $\mathbf{L}$, $\mathbf{U}$ are complex sparse matrices and $\mathbf{F} = 0$. Storing the factorization from KLU can therefore be computed as

$$mem_{\text{KLU}} = mem_{\mathbf{L}} + mem_{\mathbf{U}} + mem_{\mathbf{P}} + mem_{\mathbf{Q}} + mem_{\mathbf{R}}. \tag{3.20}$$

The memory for the vectors are

$$mem_{\text{vector}} = n \cdot b, \tag{3.21}$$

where n is the length of the vector and b the bits which for $\mathbf{P}$ and $\mathbf{Q}$ will be b_{integer} and for $\mathbf{R}$ it will be b_{double}.

In MATLAB integers like doubles are stored using 64 bits therefore $b_{\text{integer}} = b_{\text{double}} = 64$. The resulting memory requirements for storing $\mathbf{K}$ and the KLU factorization of $\mathbf{Y}_{cs}$ can be seen in Table 3.6. The memory for the coefficient matrix is showed for storing in both as a full and as a sparse matrix.

Storing the factorization requires less memory than storing the coefficient matrix for all systems no matter how $\mathbf{K}$ is stored. For the systems with a density of more than around 65% the coefficient matrix is more efficiently stored as a full matrix since the number of non-zeros gets closer to the total number of elements. Therefore, storing the row and column index on top of the number of non-zeros will result in more elements stored than just storing the full matrix. For the largest test system EECC-PSSE-33-0 the memory required storing the coefficient matrix optimally as a full matrix still requires over 2400 times more memory than just storing the factorization. The factor for Polish-Winter99 and Nordic32 is around 220 and 4.6 respectively. The big difference in the amount of memory save is due to the difference in scaling. The memory for $\mathbf{K}$ scales close to quadratic, while the memory for the factorization of $\mathbf{Y}_{cs}$ scales close to linear.

Table 3.6: Memory requirements for coefficient matrix $\mathbf{K}$ in sparse and full format and the factorization of $\mathbf{Y}_{cs}$.

Case	$\mathbf{K}$ (full)	$\mathbf{K}$ (sparse)	Factorization of $\mathbf{Y}_{cs}$
Nordic32	33.1 kB	13.2 kB	2.9 kB
Pegase89	123.8 kB	180.3 kB	17.7 kB
Pegase1354	28.0 MB	25.6 MB	164.3 kB
PTI-WECC-1648	41.4 MB	39.1 MB	233.6 kB
Polish-Winter99	86.6 MB	64.7 MB	298.8 kB
Polish-Winter03	115.1 MB	69.3 MB	349.4 kB
Pegase2869	125.6 MB	67.8 MB	390.5 kB
Polish-Winter07	138.4 MB	96.3 MB	400.4 kB
PTI-EECC-7991	0.9 GB	1.0 GB	1.2 MB
Pegase9241	1.3 GB	1.0 GB	1.4 MB
Pegase13659	2.8 GB	4.1 GB	2.1 MB
EECC-PSSE-33-0	13.3 GB	19.1 GB	5.6 MB

3.4 Discussion and Conclusion

Runtimes for computing Thévenin voltages using the *factor-solve* method is below 6ms for all systems. PMUs normally delivers data in the rate of system frequency. This means that measurements would be received every 16-20ms, and therefore with the *factor-solve* method it will be possible to compute the Thévenin voltages for every measurement.

Runtimes for Algorithm 2 dominate the *factor-solve* method, however this is only run whenever system topology changes. The implementation for Algorithm 2 is a little faster than Algorithm 1, but in general the complexity for both is quadratic. The most important change between the reference method and the *factor-solve* method is that the algorithm is going from sequential matrix multiplications to a sequential and a parallel part making it possible to parallelize. The optimal number of cores used in the parallelization of Algorithm 2 increases as the system size increases. The runtime when using parallelization is lower compared to the reference method for most systems. However, some systems does not receive a speed-up.

The smallest systems does not benefit from using the *factor-solve* method. The remaining systems gets a speed-up computing the Thévenin voltages using the *factor-solve* method but only some benefit from using Algorithm 2 over Algorithm 1. A suggestion for these system would be to use a combination of the two methods to have optimal runtime as Algorithm 1 computes everything needed in the *factor-solve* method. The Thévenin impedances and the factorization can for these system be determined using Step 1-8 of Algorithm 1 and then the Thévenin voltage can be determined using the *factor-solve* method.

A clear benefit from the *factor-solve* method is the decrease in memory usage, which was considerable for all systems. This as well as the now linear complexity for computing Thévenin voltages will make the method viable to be used in methods for real-time stability method. Furthermore, the use of fewer computational resources will make room for other methods to ensure stability of future power systems.

CHAPTER 4

Voltage Stability Boundary Monitoring Method

This chapter introduces the improved voltage stability monitoring method from [47, 48], which accounts for non-linearity in the Thévenin voltages. The runtime is optimized by developing a binary search method to find the point of maximum injectable power to a non-controlled load. The search is further optimized by fitting points to a second order polynomial to find the maximum. The different iteration algorithms is tested on a range of systems and compared to a naïve implementation. Further investigation is conducted to determine the impact of doing block-wise calculations when computing Thévenin equivalents. The main results of this chapter is based on **[Pub. C]**.

4.1 Voltage stability boundary monitoring method

The improved voltage stability method was introduced in [47, 48]. The method is developed in the SARP project. Earlier voltage stability methods determines the maximum power transfer as the point when the magnitude of the Thévenin impedance equals the magnitude of the load impedance under the assumption that the Thévenin voltage remains constant as the load impedance changes. This matching of the impedances is known as the impedance match criterion. The limitation of using this has been investigated in [13]. The voltage stability method takes in to account the changes occurring in the Thévenin voltage seen from the load. It was shown that the actual point of voltage instability is before the boundary determined by impedance match criterion.

For the method proposed in [47, 48] two assumptions are required

1. Power is injected into nodes of constant voltage magnitude. The synchronous generators are represented as a voltage source $V\angle\theta$. The voltage magnitude is constant either at the generator terminal if an AVR is present, or behind the synchronous reactance X_d depending on the excitation system.

2. Loads are represented as impedances. The method needs the instantaneous representation of the system conditions to accurately determine the changes in Thévenin voltages.

The representation due to the second assumption will therefore mean that the system changes from being divided in to voltage sources (vs nodes) and current sources (cs nodes) to voltage controlled (vc nodes) and non-controlled (nc nodes).

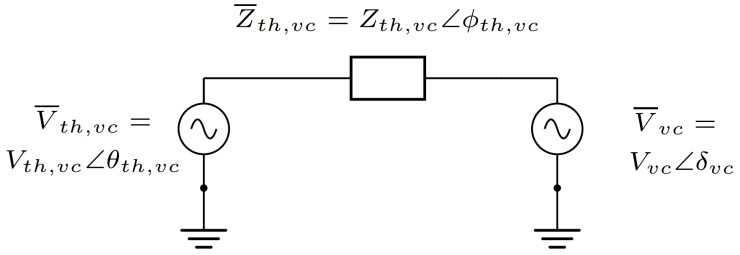


Figure 4.1: Thévenin equivalent seen from a generator

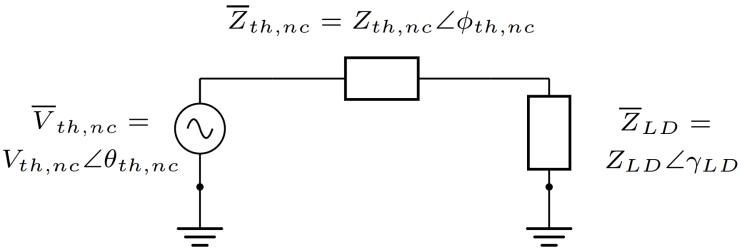


Figure 4.2: Thévenin equivalent seen from a non-controlled load modelled as an impedance

4.1.1 Thévenin equivalent computations

The computations for the Thévenin equivalent is revisited in terms of the new representation of the system in this setting. Generators is still represented as voltage sources as seen in Figure 4.1, which is the same as the representation in Chapter 2, however naming convention is changed from vs to vc. Loads are represented as impedances. Figure 4.2 shows the equivalent for a nc load $Z_{LD} = R_{LD} + jX_{LD}$. The impedances is outside the equivalent i.e. the equivalent is independent of the load impedance, which is important, when the impedance is changed to find the voltage stability boundary.

By representing loads as impedances the current injected at nc buses will be 0. The admittance matrix will then be block-wise partitioned as

$$\begin{bmatrix} 0 \\ I_{vc} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{nc} & \mathbf{Y}_{v\to n} \\ \mathbf{Y}_{n\to v} & \mathbf{Y}_{vc} \end{bmatrix} \begin{bmatrix} V_{nc} \\ V_{vc} \end{bmatrix} \tag{4.1}$$

Eliminating I_{vc} and plugging this in to the expression for the Thévenin equivalent as done in Section 2.1 gives the following expression for the Thévenin voltages

$$V_{th,nc} = -\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}V_{vc} \tag{4.2}$$

$$V_{th,vc} = \left(\mathcal{I} - \mathcal{D}\left(\overline{Z}_{th,vc}\right)\left[\mathbf{Y}_{vc} - \mathbf{Y}_{n\to v}\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}\right]\right)V_{vc} \tag{4.3}$$

Using the expression for the Schur Complement $\mathbf{Y}_{eq}$ gives

$$V_{th,nc} = -\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}V_{vc} \tag{4.4}$$

$$V_{th,vc} = \left(\mathcal{I} - \mathcal{D}\left(\overline{Z}_{th,vc}\right)\mathbf{Y}_{eq}\right)V_{vc} \tag{4.5}$$

The coefficients will be defined as $\mathbf{K}_{nc}$ and $\mathbf{K}_{vc}$ respectively

$$\mathbf{K}_{nc} = -\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n} \tag{4.6}$$

$$\mathbf{K}_{vc} = \mathcal{I} - \mathcal{D}\left(\overline{Z}_{th,vc}\right)\mathbf{Y}_{eq} \tag{4.7}$$

As earlier the Thévenin impedance is determined as

$$\overline{Z}_{th,i} = \begin{cases} \mathbf{Z}_{nc}(i,i) & i \in nc \\ \mathbf{Y}_{eq}(i,i)^{-1} & i \in vc \end{cases} \tag{4.8}$$

The admittance of an nc load will be included in the diagonal of $\mathbf{Y}_{nc}$. As the impedance is seen outside the equivalent (Figure 4.2) this should be accounted for when determining $\overline{V}_{th,nc}$ and $\overline{Z}_{th,nc}$.

Algorithm 3 Thévenin equivalent for nc load

for $i = 1..|\text{nc load}|$ **do**
  $i_g \leftarrow$ global_index(i)
  $\mathbf{Y}_{nc}(i_g,i_g) \leftarrow \mathbf{Y}_{nc}(i_g,i_g) - Y_{load}(i)$
  $\mathbf{L}_{nc}, \mathbf{U}_{nc} \leftarrow$ factorization of $\mathbf{Y}_{nc}$
  $\mathbf{U}_{\mathbf{Z}_{nc}} \leftarrow$ solve$(\mathbf{L}_{nc}, \mathcal{I})$
  $\mathbf{L}^T_{\mathbf{Z}_{nc,i_g}} \leftarrow$ solve$(\mathbf{U}^T_{nc}(:,i_g), \mathcal{I})$
  $\mathbf{Z}_{nc,i_g} \leftarrow \mathbf{L}_{\mathbf{Z}_{nc,i_g}}\mathbf{U}_{\mathbf{Z}_{nc}}$
  $Z_{th,nc}(i) \leftarrow \mathbf{Z}_{nc,i_g}(i_g)$
  $\mathbf{K}_{nc}(i,:) \leftarrow -\mathbf{Z}_{nc,i_g}\mathbf{Y}_{v\to c}$
  $\mathbf{Y}_{nc}(i_g,i_g) \leftarrow \mathbf{Y}_{nc}(i_g,i_g) + Y_{load}(i)$
end for
return $Z_{th,nc}$ and $\mathbf{K}_{nc}$

The algorithm is only run for nc loads as these are the ones that is relevant for the voltage stability method. Therefore, the index i for the nc load is converted to the global index i_g as load i might not correspond to the i'th node in the entire system. The load admittance should be removed from $\mathbf{Y}_{nc}$ to compute $\mathbf{K}_{nc}$ and $\mathbf{Z}_{nc}$ for each load. Therefore, the computations are done independently for each row of $\mathbf{K}_{nc}$ and optimized such that only the parts needed to determine the row is computed. $\mathbf{Z}_{nc,i_g}$ is the i_g'th row of $\mathbf{Z}_{nc}$. In Algorithm 3 $Z_{th,nc}$ and $\mathbf{K}_{nc}$ is using the load index i and not the global index i_g, since they are only computed for the nc loads.

The impedance of the load is part of the equivalent for the vc nodes and therefore the entire coefficient matrix for these $\mathbf{K}_{vc}$ can be determined without having to modify $\mathbf{Y}_{nc}$ repeatedly.

4.1.2 Maximum deliverable power to a load

The maximum power transfer $P_{LD,max}$ to a given load determines a margin for voltage stability as margin to the boundary of voltage stability as

$$\%\Delta P_{LD} = \frac{P_{LD,max} - P_{LD}}{P_{LD,max}} \cdot 100\%, \tag{4.9}$$

where P_{LD} is the current power transfer for the load.

The margin can be determined more accurately by accounting for changes in the Thévenin voltage magnitude seen from the load. The power transfer $P_{LD,i}$ to load i for a given change in $\overline{Z}_{LD}$ can be determined as

1. Update the admittance matrix $\mathbf{Y}_{nc}$ with the new load admittance $\overline{Y}_{LD} = \overline{Z}_{LD}^{-1}$
2. Compute the Thévenin equivalents $(V_{th,vc}, Z_{th,vc})$ seen from the generators i.e. vc nodes by recomputing $\mathbf{K}_{vc}$
3. Compute the new rotor angle of the generators δ_{vc}
4. Determine $\overline{V}_{th,nc,i}$ for the load
5. Compute the power transfer to the load $P_{LD,i}$

By iterating over different values of Z_{LD} from the current load to the impedance load Z_{th} it is possible to determine the maximum injectable power to the load $P_{LD,max}$.

The rotor angle of the generators δ_{vc} are computed as

$$\delta_{vc} = \arccos\left(\frac{V_{vc}\cos\phi_{th,nc} - P_{inj}Z_{th,vc}}{V_{th,vc}V_{vc}}\right) + \theta_{th,vc} - \phi_{th,vc}, \tag{4.10}$$

where P_{inj} is the power injected by the generator, which will be assumed to be constant as the load changes. The remaining quantities can be seen in Fig. 4.1. The power transfer to the load P_{LD} can be determined as

$$P_{LD} = \left|\frac{\overline{V}_{th,nc}}{\overline{Z}_{th,nc} + \overline{Z}_{LD}}\right|^2 R_{LD} \tag{4.11}$$

The coefficients for the nc nodes $\mathbf{K}_{nc}$ (Algorithm 3) needs to be determined only once, since each row is independent of the load. The coefficients for the vc nodes $\mathbf{K}_{vc}$ needs to be determined for each change in Z_{LD}, since they depend on the load.

To decrease runtime only necessary computations are included. This means that $\mathbf{K}_{nc}$ as shown in Algorithm 3 is not determined for all nc buses but only for nc loads. Furthermore, $\mathbf{K}_{nc}$ is used to determine the contributing generators to a load. If $\mathbf{K}_{nc}(i,j) \neq 0$ this means that generator j contributes to the computations for load i. Therefore, the Thévenin equivalent and the new rotor angle is only determined for these contributing generators.

$\mathbf{K}_{vc}$, $Z_{th,vc}$, $V_{th,vc}$ will be computed for each nc load for each change in load impedance for the generators contributing (cg) as

$$\mathbf{Y}_{eq}(gc,:) = \mathbf{Y}_{vc}(gc,:) - \mathbf{Y}_{n\to v}(gc,:)\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n} \tag{4.12}$$

$$Z_{th,vc,gc} = \mathcal{D}(\mathbf{Y}_{eq}(gc,gc))^{-1} \tag{4.13}$$

$$\mathbf{K}_{vc}(gc,:) = \mathcal{I}(gc,:) - \mathcal{D}\left(Z_{th,vc,gc}\right)\mathbf{Y}_{eq}(gc,:) \tag{4.14}$$

$$V_{th,vc,gc} = \mathbf{K}_{vc}(gc,:)V_{vc} \tag{4.15}$$

Algorithm 4 shows the steps for determining the maximum power transfer to all nc loads in the system using the above equations.

Algorithm 4 Find maximum deliverable power to nc loads

```
1: Compute K_nc and Z_th,nc
2: Determine contributing generators to each ncload
3: for each nc load do
4:    for each change in Z_LD do
5:       Update Y_nc with new value of Y_LD
6:       Compute Z_th,vc and K_vc for generators contributing
7:       Compute V_th,vc for generators contributing
8:       Compute δ_vc for generators contributing
9:       Compute V̄_th,nc for the load using K_nc
10:      Compute P_LD
11:   end for
12:   Determine P_LD,max
13: end for
```

4.2 Iteration algorithms

Computing $P_{LD,max}$ for each nc load should require as few changes in Z_{LD} such that $\mathbf{K}_{vc}$ is recomputed as few times as possible. Finding the maximum power delivered to a load should be fast but the value should also be computed accurately. The following 3 algorithms will be used to determine values for Z_{LD} to be used in the computations:

- Reference algorithm
- Binary search
- Binary Search with Polynomial Fitting (BSPF)

4.2.1 Reference algorithm

The reference algorithm is a naïve algorithm, where the interval between the initial impedance $Z_{LD,0}$ and the Thévenin impedance Z_{th} is split in to k evenly spaced points. The power transfer to the load P_{LD} for each of these are then computed and the maximum of these are taken as $P_{LD,max}$. The computations start from $Z_{LD,0}$.

P_{LD} will increase with the load up to $P_{LD,max}$ and then start to decline or be limited by a contributing generator's rotor angle δ_{vc} becoming imaginary. Therefore it is unnecessary to compute P_{LD} for larger loads when the value for P_{LD} starts to decline. When the rotor angle of a generator becomes imaginary it indicates that that generator will start to lose synchronism [45]. This happens because the generator can not deliver the amount power that is required. This type of solution is therefore not considered to be a stable operating point and any further increase in load will only put a further strain on system conditions.

To improve the naïve method the remaining computations are therefore skipped if either P_{LD} starts to decline or if a contributing generator's rotor angle δ_{vc} becomes imaginary.

4.2.2 Binary Search

The binary search determines the steps smarter by choosing each new Z_{LD} using the knowledge gained from the previous step. A traditional binary search takes the middle of an interval and use this middle value to a determine if the maximum will be either to the left or the right of the middle and then takes the middle of this interval and so forth. This method is similar but uses earlier experience.

The initial load impedance $Z_{LD,0}$ and it's power transfer $P_{LD,0}$ is the first feasible point. Initially the end of the interval $Z_{LD,end}$ is set to Z_{th}. The first first step $Z_{LD,1}$ is taken as the point 15% from Z_{th} i.e. $1.15 \cdot Z_{th}$, instead of taking the middle point between $Z_{LD,0}$ and $Z_{LD,end}$ as in a regular binary search. This was chosen using previous experience [13, 47] as the point of maximum power transfer is often found close to the boundary predicted by the impedance match criterion. This was seen in experiment to greatly improve performance. If an imaginary rotor angle is computed or the values for P_{LD} keeps declining $Z_{LD,end}$ will be updated.

For each point in the computations the current point is deemed feasible if the rotor angle is non-imaginary and infeasible otherwise.

The $Z_{LD,2}$ is chosen based on what happens for $Z_{LD,1}$ If $Z_{LD,1}$ is infeasible or $P_{LD,1} < P_{LD,0}$ the end point of the interval is updated to $Z_{LD,end} = Z_{LD,1}$ and $Z_{LD,2}$ is taken as the middle between $Z_{LD,0}$ and $Z_{LD,1}$. If $Z_{LD,1}$ is feasible and does not have lower power transfer $Z_{LD,2} = Z_{LD,end} = Z_{th}$ is tested, such that it is known if the end point of the interval is feasible, and in this case what the power transfer of the point is.

The search is then done iteratively as

1. Compute $Z_{LD,i}$ as midpoint between $Z_{LD,i-1}$ and $Z_{LD,end}$. If $Z_{LD,i-1}$ was infeasible the midpont between $Z_{LD,i-2}$ and $Z_{LD,end}$ will be used instead. When $Z_{LD,i-1}$ the endpoint will be updated and $Z_{LD,end} = Z_{LD,i-1}$.
2. If $Z_{LD,i}$ is infeasible, update the end of the interval $Z_{LD,end} = Z_{LD,i}$ else compute the power transfer $P_{LD,i}$
3. Whenever 3 feasible points has been computed investigate their relationship and eliminate the obsolete point depending on the 3 scenarios below.

Whenever there is 3 feasible points it is possible to compare them and determine, which point is obsolete and in which interval the search should be continued. The reason a traditional binary search can not be used is that for some loads the end of the interval might not be feasible and it might not be known what the last feasible value is. Therefore a traditional comparison of the point to the start and end point of the interval might not be possible. Figure 4.3 shows the 3 different scenarios where P_{LD} is plotted against $Y_{LD} = Z_{LD}^{-1}$.

1. *Scenario 1*: (Figure 4.3a) The power transfer P_{LD} is increasing with the load admittance Y_{LD}. $P_{LD,max}$ will therefore be either between 2 and 3 or after 3. If the end of the interval (Z_{th}) was determined feasible, point 3 will be $Z_{LD,end} = Z_{th}$ and the maximum is therefore known to be between 2 and 3.

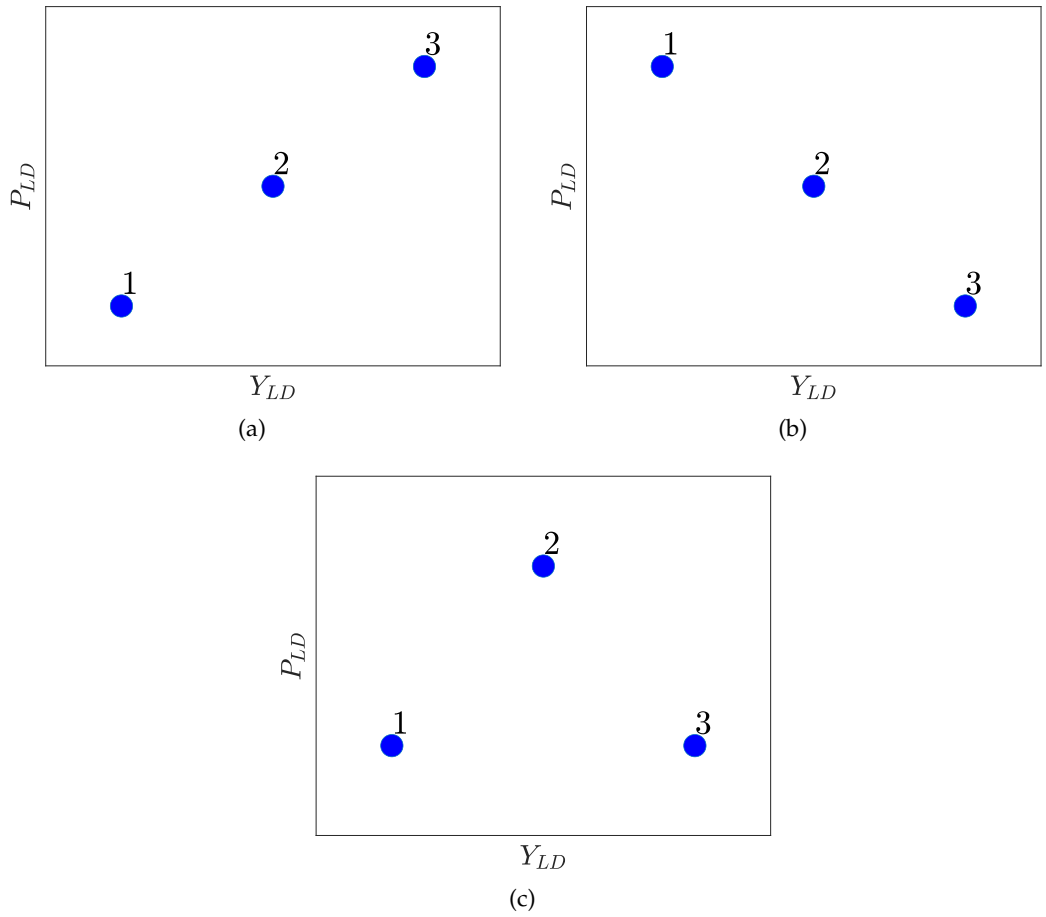


Figure 4.3: Possible scenarios for 3 feasible (Y_{LD}, P_{LD}) points: (a) Scenario 1: increasing - (b) Scenario 2: decreasing - (c) Scenario 3: midpoint largest

2. *Scenario 2*: (Figure 4.3b) The power transfer P_{LD} is decreasing with the load admittance Y_{LD}. Point 1 will in this scenario be $Z_{LD,0}$, i.e. the initial load of the node. $P_{LD,max}$ will therefore be between 1 and 2 and $Z_{LD,end}$ is updated with the load impedance of point 2. If the system has passed the boundary of voltage stability $P_{LD,max} = P_{LD,0}$ and the margin to the boundary will be 0, since the interval before $P_{LD,0}$ is not investigated.

3. *Scenario 3*: (Figure 4.3c) $P_{LD,max}$ will be between 1 and 3. For this scenario the midpoint to the left and the right of the middle is computed iteratively. After each computations the obsolete point will be removed such that there is always 3 points. Figure 4.4 shows the 2 possible scenarios for a point computed between 1 and 2. In 4.4a point 1 is obsolete and therefore removed. In 4.4b point 3 is obsolete and therefore removed.

The iterations are stopped using the threshold τ. For scenario 1 and 2 the search is stopped when the difference between the 2 relevant points are less than τ i.e. when the obsolete point is eliminated the 2 remaining points should be close. The difference in both P_{LD} and Z_{LD} is investigated to avoid a scenario where the difference in P_{LD} is small but the points are actually on either side of the maximum, which would result in a big difference in Z_{LD}. In scenario 3 the search is stopped when the relative difference on both sides of the midpoint is less than τ.

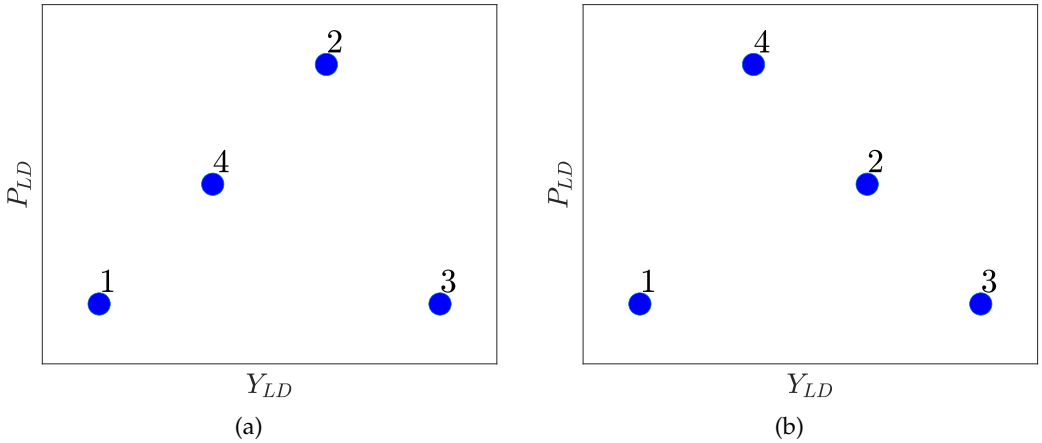


Figure 4.4: Possible scenarios for the 4th point in scenario 3. Points are plotted as (Y_{LD}, P_{LD}) points: (a) Scenario 3a: Point 4 smaller than point 2 - (b) Scenario 3b: Point 4 larger than point 2

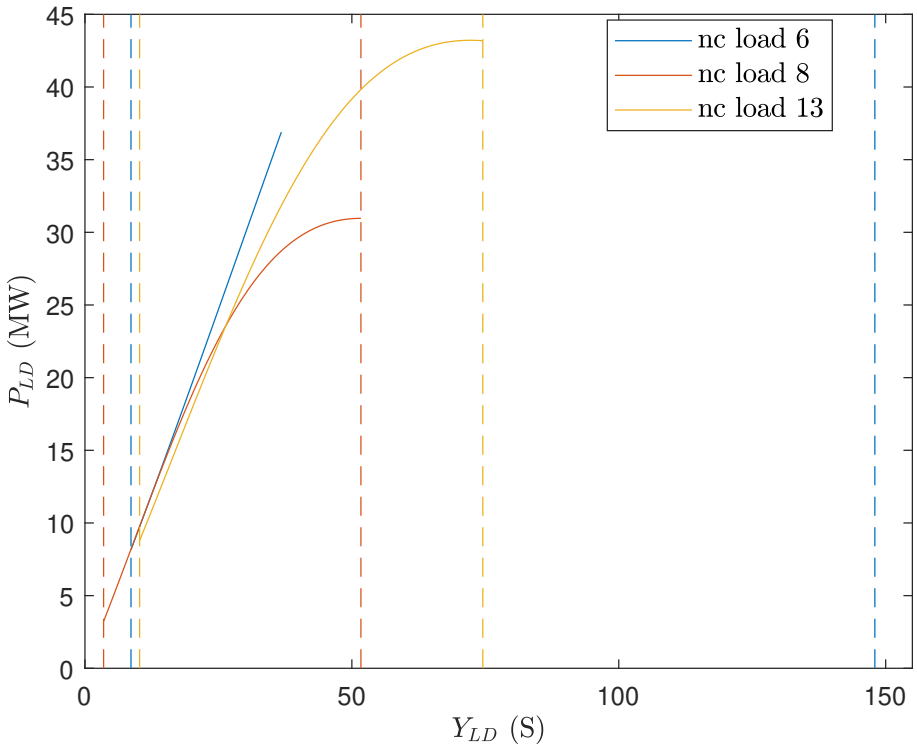


Figure 4.5: $(Y_{LD}, P_{LD}) = \left(Z_{LD}^{-1}, P_{LD}\right)$ curves for 3 different nc loads for the system Nordic32, Table 2.1. The vertical dashed lines show the interval from $Y_{LD,0} = Z_{LD,0}^{-1}$ to $Y_{th} = Z_{th}^{-1}$ for each load.

4.2.3 Binary Search with Polynomial Fitting (BSPF)

BSPF uses the binary search introduced in the last section, but utilizes knowledge of the shape of the (Y_{LD}, P_{LD})-curves. Figure 4.5 shows the (Y_{LD}, P_{LD})-curve for 3 different nc loads in the Nordic32 system, Table 2.1. The vertical dashed lines show the interval from $Y_{LD,0} = Z_{LD,0}^{-1}$ to $Y_{th} = Z_{th}^{-1}$ for each load, which is the interval investigated when finding $P_{LD,max}$.

The shape of the curve for each loads are:

- *nc load 6*: The curve is almost linear and only have feasible points in part of the interval. This happens when at least one generator start to lose synchronism before the nc load reaches it's maximum potential for power transfer.
- *nc load 8*: The curve looks like a second order polynomial and have feasible points in the entire interval. $P_{LD,max}$ occurs for this nc load at $Z_{LD} = Z_{th}$.
- *nc load 13*: The curve looks like a second order polynomial and have feasible points in the entire interval. $P_{LD,max}$ occurs for this nc load just before Z_{th}.

Using the shape of nc load 8 and 13 the idea is to fit any 3 feasible points to a second order polynomial and use the maximum of this curve as the next step. This approach will not work for nc load 6 and therefore the method is combined with the binary search to be able to predict the maximum for this type of load.

Whenever 3 feasible points are given they are fitted to a second order polynomial $f(x) = ax^2+bx+c$ and the position of the maximum $\tilde{m} = -\frac{b}{2a}$ is computed. The maximum is then determined to be either feasible or infeasible. The maximum is considered infeasible if

- a is positive as this means that $\tilde{m}$ is a minimum
- the maximum is outside the given interval i.e. before $Z_{LD,0}$ or after $Z_{LD,end}$
- the maximum is close to $Z_{LD,end}$ and this is known to be an infeasible point

If the maximum is infeasible the 3 points are reduced to 2 points like in scenario 1 and 2 of the binary search and a new point is computed before redoing the polynomial fitting.

If the maximum is feasible the power transfer P_{LD} is computed and the now 4 points are reduced to 3 points using the same logic as for the binary search to remove the obsolete point and then the fitting is repeated with 3 remaining points.

BSPF stops using the threshold τ either as in the binary search when the relative difference between 2 points is low or when $\tilde{m}$ is feasible and close to the previously computed $\tilde{m}$. The curve is not an exact second order polynomial and therefore to ensure an accurate $\tilde{m}$ the difference in P_{LD} for the feasible points can not be more than 50%. This will keep outlying points of the interval from skewing the results and at least one extra point will be computed.

If the difference in Y_{LD} of the 3 feasible points are close without P_{LD} being close the fitting is skipped and the binary search is used instead. This will only happen without a plausible solution being found, if the load is similar to nc load 6 (Figure 4.5).

4.2.4 Implementation and test

The iteration algorithms are implemented in MATLAB and evaluated with respect to runtime and accuracy of the results. The runtime is tested on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. The systems tested will be the first 8 systems given in Table 2.1. The systems are shown in Table 4.1, where the number of nc loads is shown as not all nc nodes are loads.

The number of points k for the reference method is chosen to be 50, since this was found to give reasonable runtimes. The threshold $\tau = 10^{-3}$ as this gave reasonable runtimes. Lower thresholds did not give a much better accuracy and resulted in the algorithms taking small insignificant steps.

Table 4.1: Test systems for iteration algorithms

Case	no. of buses	no. of nc nodes	no. of nc loads
Nordic32	46	26	18
Pegase89	89	77	35
Pegase1354	1354	1094	673
PTI-WECC-1648	1648	1335	1004
Polish-Winter99	2383	2056	1504
Polish-Winter03	2746	2372	1661
Pegase2869	2869	2359	1491
Polish-Winter07	3012	2665	1939

Table 4.2: Runtime for each iteration algorithm for voltage stability boundary monitoring

Case	Runtime (s) Reference	Runtime (s) Binary search	Runtime (s) BSPF
Nordic32	0.11	0.03	0.04
Pegase89	0.58	0.12	0.09
Pegase1354	165.93	36.80	19.26
PTI-WECC-1648	553.10	116.94	57.80
Polish-Winter99	731.55	169.44	97.91
Polish-Winter03	711.06	165.35	95.28
Pegase2869	726.86	164.80	84.88
Polish-Winter07	777.23	181.19	101.56

In some cases the next point $Z_{LD,i}$, which computed as the midpoint between $Z_{LD,i-1}$ and $Z_{LD,end}$, will give result in a point close to one of the feasible points. This will result in a very small step size and make the search quite slow. In these cases, it would be better to recompute $Z_{LD,i}$ and instead take the midpoint between the two feasible point, that remained after the last removal of obsolete points. This will ensure that the feasible points will be more evenly spread and then after removing the new obsolete point the steps are taken more efficiently. Therefore, another threshold $\tau_2 = 10^{-4}$ is introduced and if the difference between $Z_{LD,i}$ and one of the feasible points is lower than τ_2 $Z_{LD,i}$ is recomputed.

The runtime for the different iteration algorithms can be seen in Table 4.2. The first part of Algorithm 4 (Step 1 and 2) is the same for each method and therefore not included in the runtime. Algorithm 3 is computed once and does not depend on the iteration algorithm, and therfore the runtime of this is not tested either.

The runtime is plotted against the number of nc loads in Figure 4.6. For the largest system Polish-Winter07 the binary search is over 4 times faster than the reference method, while BSPF lowers runtime by an extra 40% to a factor of 7.5 in total.

The plot shows, that the complexity is close to quadratic to the number of nc loads for all the algorithms. This seems reasonable as the computations in the loops are quadratic as they are similar to the computations of Chapter 3, which were quadratic. This means that even though

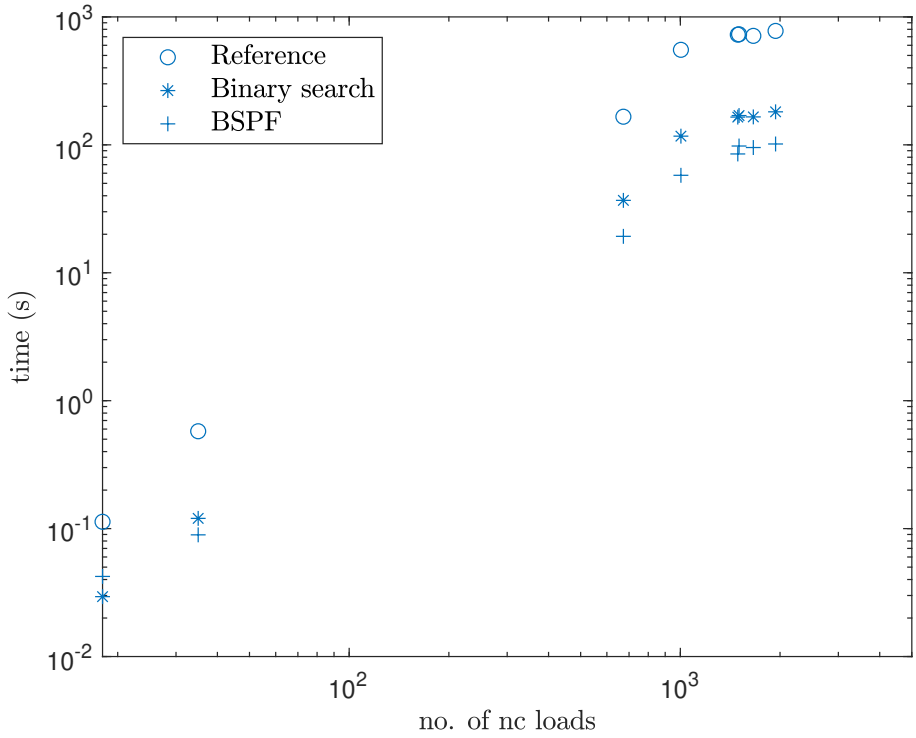


Figure 4.6: The runtime for each iteration algorithm (Reference, Binary search and BSPF) depending on the number of nc loads. The plot is logarithmic.

lower runtimes are seen for the search algorithms the scaling of the methods are similar. The binary search and BSPF both takes fewer steps to find the maximum power transfer to each load, which result in lower runtimes. However recomputing in each step is quadratic, and therefore the total runtime will be quadratic as well.

Runtime is important however, the accuracy of the computed $P_{LD,max}$ is just as important to be able to correctly determine the distance to the voltage stability boundary. All iterations algorithms computes P_{LD} for different values of load impedance. Therefore, if an algorithm computes a larger $P_{LD,max}$ it has been able to choose a load impedance that more accurately result in the actual $P_{LD,max}$.

The percentage difference between the reference and binary search algorithm is computed as

$$\Delta P\% = \frac{P_{LD,max,search} - P_{LD,max,ref}}{P_{LD,max,ref}} \cdot 100\%, \tag{4.16}$$

where $P_{LD,max,search}$ is the values computed by the binary search and $P_{LD,max,ref}$ is the values computed by the reference method. $P_{LD,max,search}$ is larger than $P_{LD,max,ref}$ for $\Delta P\% > 0$ and the opposite for $\Delta P\% < 0$. The maximum, minimum and average difference can be seen in Table 4.3.

The numbers show that generally the binary search computes a $P_{LD,max}$ that is considerable larger than the reference algorithm, which means that the binary search is more accurate. On average the percentage difference is largely in favour of the binary search and the biggest difference in favour of the reference is less than 0.41%. If more steps were taken in the reference algorithm it would be possible to get better results, however this would also result in runtimes to worsen. Tests were

Table 4.3: $P_{LD,max}$ difference between reference algorithm and binary search

Case	Maximum difference (%)	Average difference (%)	Minimum difference (%)
Nordic32	16.19	1.56	0
Pegase89	46.38	1.33	0
Pegase1354	$1.19 \cdot 10^4$	66.06	-0.407
PTI-WECC-1648	$5.47 \cdot 10^3$	32.76	0
Polish-Winter99	$2.78 \cdot 10^4$	93.59	0
Polish-Winter03	$1.06 \cdot 10^5$	449.58	0
Pegase2869	$5.08 \cdot 10^9$	$3.42 \cdot 10^6$	-0.405
Polish-Winter07	$2.36 \cdot 10^5$	566.28	0

Table 4.4: $P_{LD,max}$ difference between binary search and BSPF

Case	Maximum difference (%)	Average difference (%)	Minimum difference (%)
Nordic32	0.004	$4.74 \cdot 10^{-4}$	$-5.71 \cdot 10^{-4}$
Pegase89	0.009	$1.81 \cdot 10^{-4}$	-0.003
Pegase1354	0.970	0.006	-0.244
PTI-WECC-1648	2.789	0.009	-0.197
Polish-Winter99	5.238	0.012	-0.406
Polish-Winter03	5.522	0.011	-0.312
Pegase2869	0.946	0.004	-0.708
Polish-Winter07	1.602	0.005	-0.368

conducted using $k = 250$ for the reference algorithm, which resulted in runtimes 4 times larger than for $k = 50$, however $\Delta P\%$ was still large and some systems even ended up with a larger average difference. The reference method splits the interval between $Z_{LD,0}$ and Z_{th} evenly and therefore if k is not large enough there is a chance that the splitting chosen might miss the actual maximum.

BSPF is compared to the binary search as

$$\Delta P\% = \frac{P_{LD,max,BSPF} - P_{LD,max,search}}{P_{LD,max,search}} \cdot 100\% \tag{4.17}$$

Again $P_{LD,max,BSPF}$ is larger than $P_{LD,max,search}$ for $\Delta P\% > 0$ and the opposite for a $\Delta P\% < 0$. The difference between the two can be seen in Table 4.4.

The difference between the binary search and BSPF is minimal and show similar accuracy but on average the BSPF computes a more accurate result at a lower runtime. Comparing BSPF to the reference algorithm (Table 4.5) the minimum difference is at machine accuracy i.e. 0. Therefore BSPF is better at computing $P_{LD,max}$ than the reference algorithm for all nc loads, whereas the binary search had a couple of nodes, where it was a little worse than the reference algorithm.

4.2.5 Parallelization

The voltage stability boundary monitoring is a stability assessment meant to operate in real-time, but with the current implementation this is not feasible. However, runtimes can be further

Table 4.5: $P_{LD,max}$ difference between reference algorithm and BSPF

Case	Maximum difference (%)	Average difference (%)	Minimum difference (%)
Nordic32	16.19	1.56	0
Pegase89	46.37	1.33	0
Pegase1354	$1.19 \cdot 10^4$	66.08	0
PTI-WECC-1648	$5.47 \cdot 10^3$	32.79	0
Polish-Winter99	$2.78 \cdot 10^4$	93.62	0
Polish-Winter03	$1.06 \cdot 10^5$	449.64	0
Pegase2869	$5.08 \cdot 10^9$	$3.42 \cdot 10^6$	0
Polish-Winter07	$2.36 \cdot 10^5$	566.39	0

Table 4.6: Runtime for each iteration algorithm run in parallel on 24 cores.

Case	Runtime (s) Reference	Runtime (s) Binary search	Runtime (s) BSPF
Nordic32	0.061	0.059	0.061
Pegase89	0.119	0.098	0.103
Pegase1354	8.575	2.196	1.201
PTI-WECC-1648	30.071	6.288	3.865
Polish-Winter99	36.546	9.363	5.881
Polish-Winter03	36.803	9.465	5.816
Pegase2869	35.970	8.411	4.683
Polish-Winter07	39.450	9.530	5.873

improved by introducing parallelization. The computations for the loads are distributed on to several cores by parallelizing the outer loop i.e. line 3 in Algorithm 1. The number of iterations for each load is different and therefore the distribution of work will be split dynamically.

The 3 iteration algorithms are parallelized on a machine with 2 CPUs of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz with 12 cores each, which means 24 cores is available in total. Testing showed that the lowest runtime for all systems occurred, when all 24 cores where in use. However, the last couple of cores only gave a small improvement, since overhead starts to get more dominant, as was seen by the parallelization in Chapter 3. The results from parallelization are shown in Table 4.6.

Figure 4.7 shows the runtime for parallelizing the algorithms on 24 cores along the original runtimes in serial. The parallelization of the reference algorithm result in runtimes below 40s for all systems, which is a considerable improvement from the almost 780s, when run in serial. The binary search have runtimes below 10s, while BSPF gives runtimes below 6s. These improved runtimes makes the method feasible also for larger systems, when using parallelization.

4.3 Block-wise calculations

Runtimes was considerable better using BSPF and parallelization, however the complexity can still be improved. The splitting of $\mathbf{Y}_{nc}$ in to blocks using BTF can be used for further optimization.

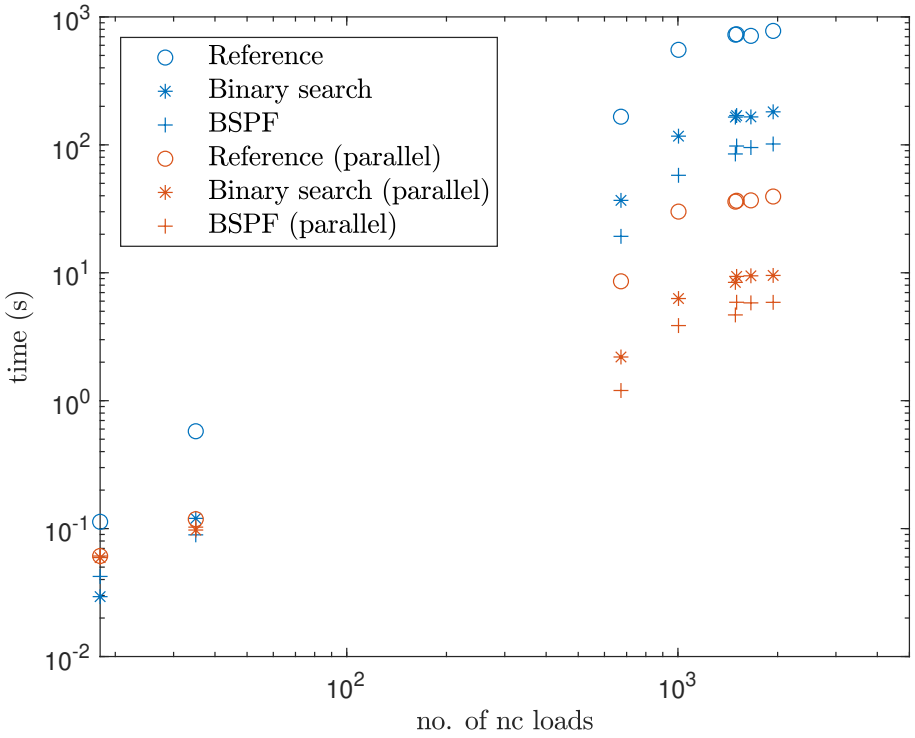


Figure 4.7: The runtime for each iteration algorithm (Reference, Binary search and BSPF) depending on the number of nc loads and the runtime for the algorithms run in parallel on 24 cores is plotted. The plot is logarithmic.

Each block is a strongly connected component, which means that the block is independent from the remainder of the system. Therefore, computations for a node can be done by only using the block, that it is part of.

$\mathbf{K}_{nc}$ is computed one row at a time and therefore it is possible to limit computations to only the block that the node is part of. Computing $\mathbf{K}_{nc}$ is done as in Algorithm 3 but instead of using the entire matrix $\mathbf{Y}_{nc}$, the matrix $\mathbf{Y}_{nc,b}$ is used, which is the block b in BTF that the node is part of. Using this it is possible to compute $\mathbf{Z}_{nc,b}$ and as in Algorithm 3 only the row concerning the node is used $\mathbf{Z}_{nc,b}(i_b,:)$, where i_b is the index of the load in the block b. The vectors (p_b, q_b) are the indices for the rows and columns for the block. Using $\mathbf{Z}_{nc,b}(i_b,:)$ $\mathbf{K}_{nc}(i,:)$ can be computed as

$$\mathbf{Z}_{nc}(i, q_b) = \mathbf{Z}_{nc,b}(i_b,:) \tag{4.18}$$

$$\mathbf{K}_{nc}(i,:) = \mathbf{Z}_{nc}(i, q_b)\mathbf{Y}_{v\to c} \tag{4.19}$$

The remaining part of $\mathbf{Z}_{nc}(i,:)$ will be 0, since only the elements in the block affects the node.

Computing $\mathbf{K}_{vc}$ block-wise is a bit more complex. Computing $\mathbf{Y}_{eq}$ requires contributions from all blocks. Therefore when iterating over values of Z_{LD} only the block that the node is part of is recomputed and all the contributions from the remaining blocks need to be added. $\mathbf{Y}_{eq}$ is computed as below with $\mathbf{Y}_b$ being the contribution from each block.

$$\mathbf{Y}_b = \mathbf{Y}_{c\to v}(:, q_b)\mathbf{Y}_{nc}^{-1}(p_b, q_b)\mathbf{Y}_{v\to c}(p_b,:) \tag{4.20}$$

$$\mathbf{Y}_{eq} = \mathbf{Y}_{vc} - \sum_b \mathbf{Y}_b \tag{4.21}$$

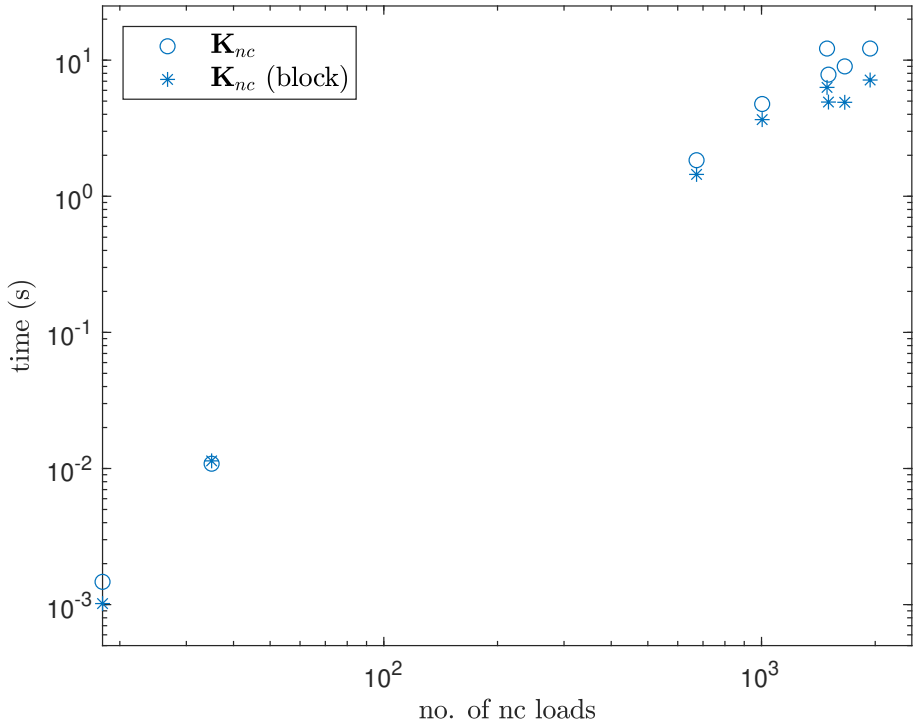


Figure 4.8: The runtime for computing $\mathbf{K}_{nc}$ and $Z_{th,nc}$ by Algorithm 3 and by utilizing block-wise computations. The plot is logarithmic.

When Z_{LD} changes for one node, $\mathbf{Y}_b$ for the block it is part of can be recomputed and $\mathbf{Y}_{eq}$ can be computed. Therefore the initial value of all $\mathbf{Y}_b$ needs to be computed for all blocks such that they are available when computing the voltage stability margins.

For smaller blocks i.e. some blocks are only 1x1 the computations are considerable faster than having to compute using the entire matrix. For a 1x1 matrix everything can be done in constant time i.e. $O(1)$. Inverting will now be division with a scalar and multiplication is with a scalar as well. For smaller matrix's inverting directly might be faster than first computing the LU factorization.

This is utilized in the computations for $\mathbf{K}_{nc}$ and $Z_{th,nc}$ from Algorithm 3, where the 1x1 block scenario is implemented as a special case, where the block is treated as a scalar instead.

4.3.1 Implementation and test

The block-wise computations are implemented in MATLAB and evaluated with respect to runtime and accuracy of the results. The runtime is tested on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz.

The runtimes for computation of $\mathbf{K}_{nc}$ and $Z_{th,nc}$ can be seen in Figure 4.8. The runtimes for Algorithm 3 seem to have a complexity somewhere between 1 and 2 around 3/2=1.5. The runtimes for the block-wise computations are lower than for the original implementation however the complexity is similar if not the same as the original implementation.

The runtime for each iteration algorithm with the original and the block-wise implementation is seen in Figure 4.9. As explained earlier the contribution from each block $\mathbf{Y}_b$ needs to be computed before hand and this runtime is seen in Figure 4.10.

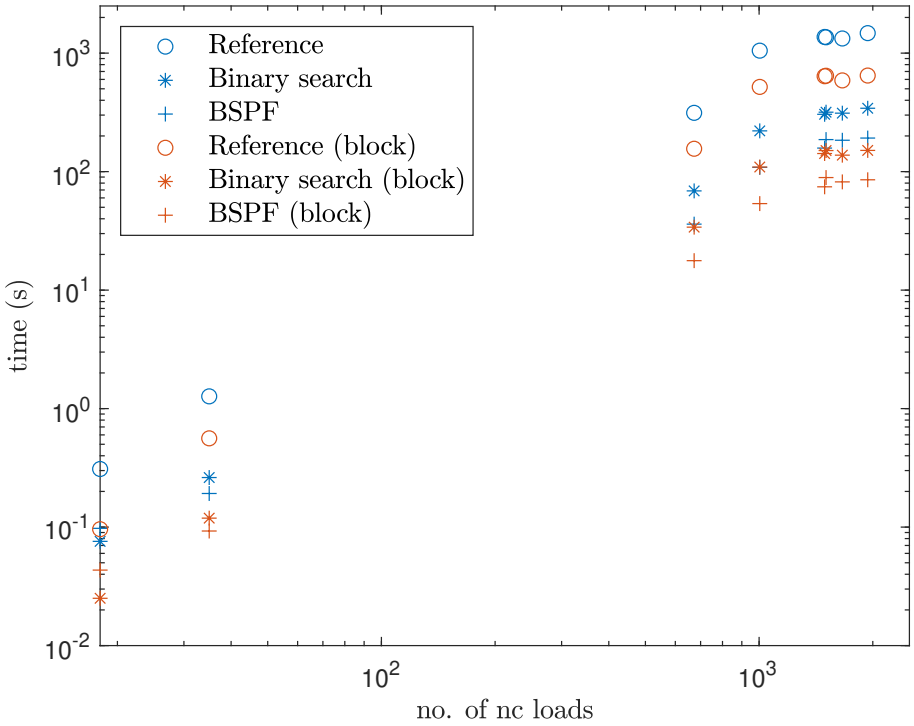


Figure 4.9: The runtime for each iteration algorithm (Reference, Binary search and BSPF) depending on the number of nc loads and the runtime for the algorithms when doing block-wise computations. The plot is logarithmic.

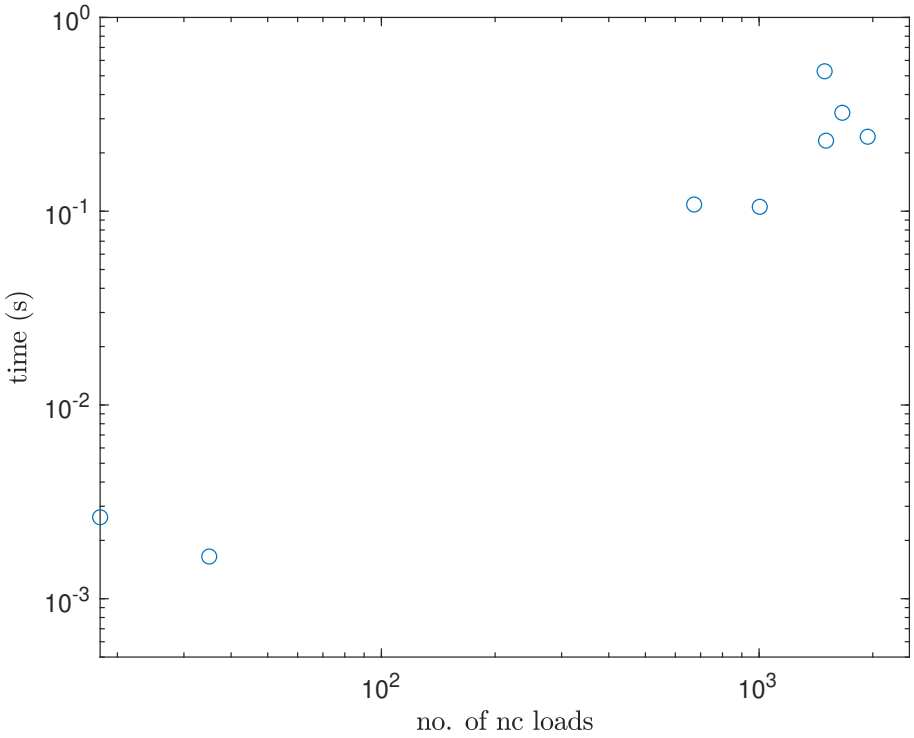


Figure 4.10: The runtime for setting up the blocks $\mathbf{Y}_b$ prior to running the iteration algorithms. The plot is logarithmic.

Including the runtime from setting up the blocks still gives lower runtime doing the block-wise computations compared to the original implementation. However, as seen the complexity is the same for both implementations, and therefore the gain from doing the block-wise computations is limited.

The reason for this is that BTF is dominated by one large block as seen in Section 2.3.4. Therefore, this will also be the dominating factor, when doing block-wise computations. It seems reasonable that this scales with system size giving similar complexity.

This can of course also be parallelized, however each part of the loop does a different amount of work since blocks are of different sizes and therefore it is harder to split the work evenly on to different cores. This might result in some cores doing most work, while the remaining cores just wait for these to finish.

4.4 Conclusion

The binary search with polynomial fitting is able to determine the maximum power transfer to loads faster and more accurately than the reference algorithm. Comparing BSPF to using binary search shows that the polynomial fitting improves runtimes considerably. The smallest system has faster runtime using the binary search, whereas the remaining systems are fastest using BSPF. The polynomial fitting is only an improvement for some buses as seen in Figure 4.5. For the buses, where the power transfer is limited by a generator starting to lose synchronism, the polynomial fitting is a waste of time, since the shape is closer to linear. It is possible to get even better accuracy by lowering the threshold however this is at the cost of runtime. Runtime is improved further by use of parallelization but the complexity remains the same for all algorithms, and therefore computations need to be improved further.

Testing with block-wise computations showed that it was possible to achieve lower runtimes by doing this however the complexity was still the same. This shows that it is needed to use other ways to optimize the runtime such that complexity for computing $\mathbf{K}_{vc}$ and $\mathbf{V}_{th,vc}$ can be lowered.

CHAPTER 5

Efficient Refactorization using Hierarchical Matrices

This chapter introduces the hierarchical matrix structure used to generate an approximate LU factorization. This is used to do an efficient refactorization, when an element in the diagonal changes. The refactorization is tested on a range of systems.

5.1 Hierarchical matrices

The hierarchical matrix structure presented here is described in [49]. Given a matrix each level in the hierarchical representation is a level of discretization for this matrix. The top level is the entire matrix and on each sublevel the matrix is divided in to smaller and smaller blocks. Each level i divide the columns and rows in to 2^i blocks. An 8x8 matrix can have a maximum of 3 levels where the lowest level, i.e. the leaf nodes, gives the highest discretization and is an exact representation of the matrix, where non-zero elements are edges between the leaf nodes. The higher levels i.e. a higher discretization will be approximated in the algorithm to limit fill-in of the factorization.

Figure 5.1 show an example matrix $\mathbf{A}$, where non-zero entries are colored. This matrix is used to construct the hierarchical tree in Figure 5.2. Each level of a hierarchical tree except the root level has 2^{i-1} black-nodes each with two red children.

The naming convention for nodes in the hierarchical tree is:

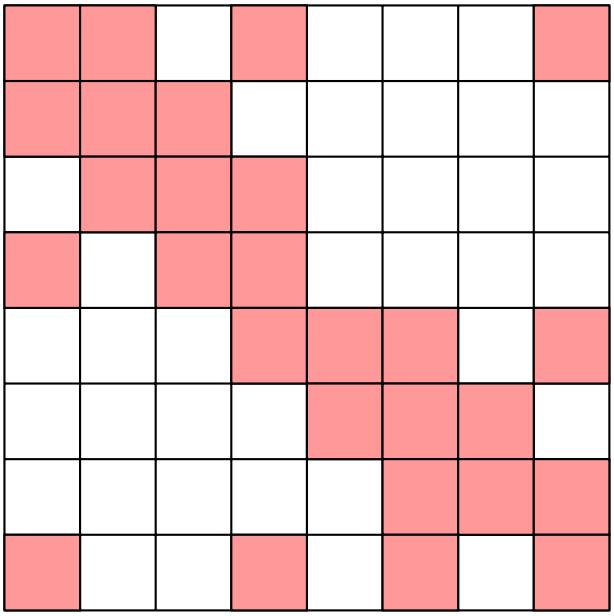


Figure 5.1: Example of an 8x8 matrix $\mathbf{A}$ with non-zero pattern marked red.

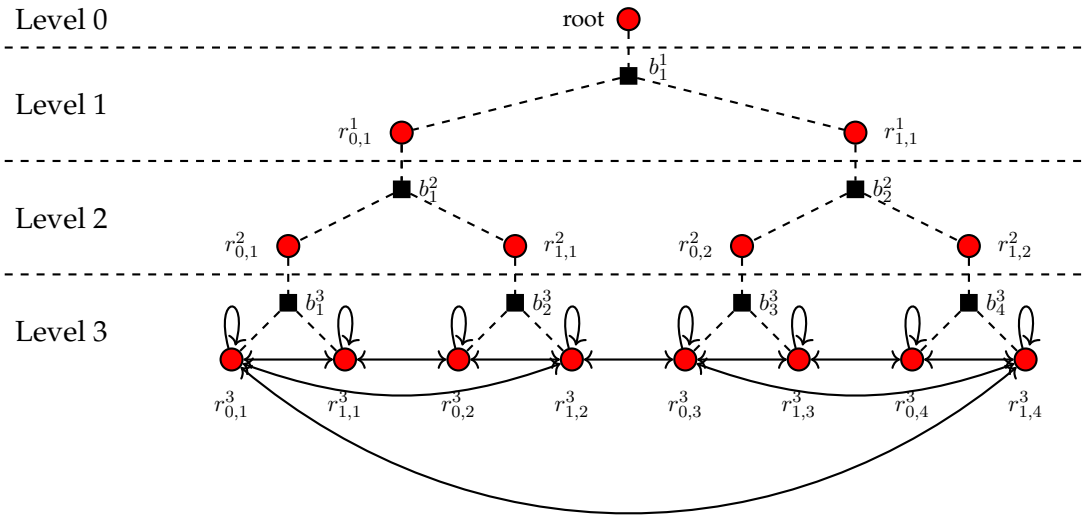


Figure 5.2: Example of hierarchical tree for the matrix **A** (Figure 5.1). Interaction edges (solid) are initially only present at the leaf nodes. Parent-child edges (dashed) show the relationships between the nodes.

- black-node: b_j^i is the j'th black-node at level i
- red-node: $r_{k,j}^i$ is the k'th child of black-node j at level i

The parent-child relationships between the nodes is shown with dashed lines in the tree. On the leaf level there will be interaction edges (solid lines) between the nodes corresponding to the non-zeros in the matrix. The red-nodes at the leaf level correspond to the rows and columns of the matrix in Figure 5.1. The diagonal of the matrix is zero free and therefore all nodes have a self-edge. As an example (1,2) and (2,1) is non-zero and therefore there is a bidirectional edge between the first 2 red-nodes i.e. $r_{0,1}^1$ and $r_{1,1}^1$. This is then done with all the non-zeros and results in the interaction edges seen in Figure 5.2.

For more elaborate implementations the splitting of the intervals might not be as naïve and permutation could be used to split the matrix in more favourable ways depending on the non-zero pattern of **A**. Furthermore the leaf level might not for larger matrices represent individual rows and columns and instead the discretization can be coarser and leaf nodes will represent blocks in the matrix instead. Using as fine a discretization as in the example is inefficient and will result in higher runtimes. However, a too coarse discretization would also be slow, since the block will become too big.

Computing the approximate factorization of the matrix will for each level have the steps

1. Merge red-nodes to super-nodes
2. For each super-node
   a) Compress well-separated edges
   b) Eliminate super-node
   c) Eliminate black-node (parent of super-node)

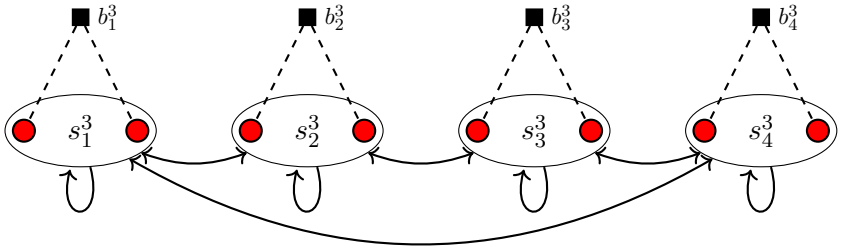


Figure 5.3: Example of red siblings merged to super-nodes. This is the leaf nodes of Figure 5.2 being merged.

5.1.1 Forming super-nodes

Super-nodes are created by merging red siblings, resulting in the structure shown in Figure 5.3. Super-node s_j^i is the j'th super-node of level i. The edges between the super-nodes are formed using the edges between the red siblings.

As an example the edge $e_{s_1^3 \to s_2^3}$ from super-node s_1^3 to super-node s_2^3 will be created as

$$e_{s_1^3 \to s_2^3} = \begin{bmatrix} e_{r_{0,1}^3 \to r_{0,2}^3} & e_{r_{1,1}^3 \to r_{0,2}^3} \\ e_{r_{0,1}^3 \to r_{1,2}^3} & e_{r_{1,1}^3 \to r_{1,2}^3} \end{bmatrix} \tag{5.1}$$

Some of these edges might not exist and the entry of the matrix will therefore be 0. For the above example the edge will be

$$e_{s_1^3 \to s_2^3} = \begin{bmatrix} 0 & e_{r_{1,1}^3 \to r_{0,2}^3} \\ e_{r_{0,1}^3 \to r_{1,2}^3} & 0 \end{bmatrix}, \tag{5.2}$$

since the remaining edges are 0 i.e. doesn't exist (Figure 5.2).

After merging the super-nodes the elimination process starts.

5.1.2 Compression

Well-separated edges will be compressed prior to elimination of the super-node. For the first node there will be no well-separated edges, as these emerge during elimination. Well-separated edges are edges between well-separated nodes, which are nodes that are not adjacent. Leaf nodes are adjacent if they have at least one edge between them. Nodes further up the tree are adjacent if their leaf descendants are adjacent. An edge between two well-separated nodes is defined as well-separated.

Well-separated edges are essentially fill-in created in the factorization, which is often numerically low rank. The idea is that these can be approximated using a low-rank factorization. The well-separated interactions between the super-nodes are essentially pushed to the parent level, which will be considered as a set of auxiliary variables.

Figure 5.4 shows an example of a super-node which has well-separated edges to t nodes. The matrices representing the edges are vertically concatenated and a low-rank approximation method

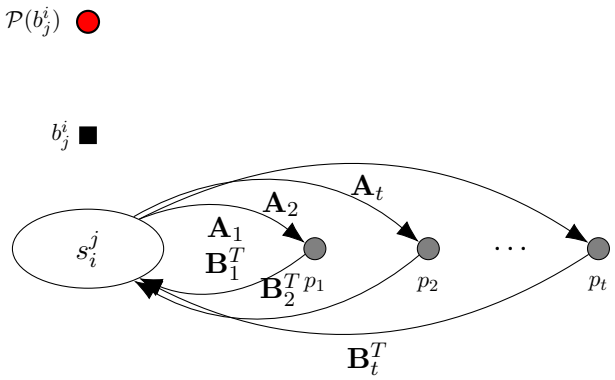


Figure 5.4: Example of a super-node with t well-separated edges.

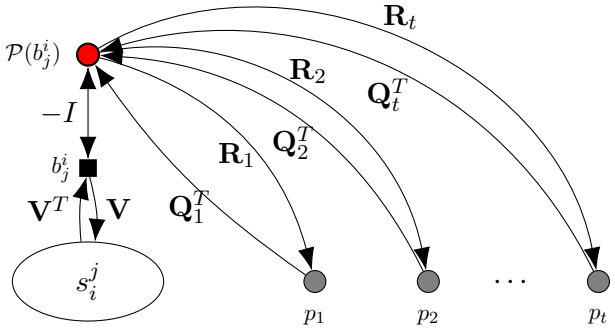


Figure 5.5: Example of a super-node with t well-separated edges, which have been compressed to the parent level.

(e.g. Singular Value Decomposition (SVD)) is used to approximate the edges as

$$\begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_t \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_t \end{bmatrix} \simeq \begin{bmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_t \\ \mathbf{Q}_1 \\ \vdots \\ \mathbf{Q}_t \end{bmatrix} \mathbf{V}^T \tag{5.3}$$

The matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ are both of size m_k x m, while $\mathbf{R}_k$ and $\mathbf{Q}_k$ are of size m_k x r and $\mathbf{V}$ is of size m x r. r is the rank of the approximation, which determines the accuracy. The rank will be chosen by use of a threshold τ. All singular values below the threshold will be excluded i.e. the rank r is chosen such that for the singular values S_i it holds that $S_i \geq \tau$ for $i \leq r$ and $S_i < \tau$ for $i > r$. Other ways of determining the rank could also be used. For example by choosing the rank such that $S_i/S_1 \geq \tau$ [49].

The resulting system after compression of the super-node in Figure 5.4 is shown in Figure 5.5.

Not all nodes will have well-separated edges and therefore the compression will be skipped for these. This results in the parent node not being used as there will be no edges to it.

5.1.3 Elimination

After compression the super-node will be eliminated. The compression process ensures that the super-node is only connected to it's original "neighbours", which preserves sparsity in the matrix, however there will be a larger number of equations due to the inclusion of the parent nodes as auxiliary nodes.

The elimination process is a standard block Gauss elimination. Elimination of node v_i creates new edges or updates existing edges. For every pair of outgoing edges $e_{v_i \to v_k}$ and incoming edges $e_{v_j \to v_i}$ a new edge is created between node v_k and v_j or an existing edge is updated. The new edge or the contribution to the existing edges $e_{v_j \to v_k}$ is computed as

$$- e_{v_i \to v_k} e^{-1}_{v_i \to v_i} e_{v_j \to v_i} = -\mathbf{A}_{k,i} \mathbf{A}^{-1}_{i,i} \mathbf{A}_{i,j}, \tag{5.4}$$

where the $\mathbf{A}_{j,i}$ is block j, i of the matrix $\mathbf{A}$. Block $\mathbf{A}_{k,j}$, which corresponds to the edge $e_{v_j \to v_k}$ will be updated by the above computation.

This elimination process is what causes new edges to occur in the factorization introducing additional fill-in, which is what this method tries to overcome by compression. By comparison in direct methods this fill-in will be computed.

5.1.4 Algorithm

The overall algorithm for the factorization will be

Algorithm 5 Hierarchical matrix factorization

```
Initialize H tree
for i = l ... 1 do
    for j = 1 ... 2^{i-1} do
        s^i_j ← MergeRedNodes(r^i_{0,j}, r^i_{1,j})
    end for
    for j = 1 ... 2^{i-1} do
        Compress(s^i_j)
        Eliminate(s^i_j)
        Eliminate(b^i_j)
    end for
end for
```

Here l is the number of levels for the specific system. The size of the matrix puts a limit on the size of l. In general a lower l gives a better accuracy since the computations will get closer to the direct methods as fewer nodes means fewer occurrences of well-separated edges, which are the edges approximated in the compression.

5.1.5 Solve

After factorization the solve process consist of a forward and backwards traversal of the nodes for any right hand side. This is similar to the forward and backwards substitutions in the standard LU factorization. The forward traversal visit all nodes in the order they were eliminated to update the right hand side. The backwards traversal visit all the nodes in the reverse order to compute for x.

As auxiliary variables are introduced, when the well-separated edges are compressed, the solution computed will be for all variables and not only the original ones. This means a little extra work is done in the solve step but the number of auxiliary variables is of the same order as the original number of variables and therefore does not affect complexity.

5.2 Refactorization

In the voltage stability assessment method introduced in Chapter 4 the factorization has to be recomputed for each change in Z_{LD}. Testing with block-wise computations showed that a big block will still dominate the computation. Therefore, the idea is to use the hierarchical structure on this big block and do a fast refactorization to optimize computations for this block. The remaining blocks of $\mathbf{Y}_{nc}$ are small and efficient to compute and are therefore handled as before.

The fast refactorization is done by only recomputing the elements that is affected by changing Z_{LD}. This is done by splitting nodes of each level in to affected and unaffected nodes. The elimination of each level then starts by eliminating the unaffected nodes and then the affected such that the affected node does not affect any super-nodes on the same level but only nodes on the parent level. This will limit the amount of nodes being affected in the factorization.

Z_{LD} is part of the diagonal and therefore for the affected leaf node s_a^l only the self-edge is affected. Therefore, any edges created or updated in the elimination step will be affected as the self-edge is used in computation of these as seen in (5.4). If the super-node has well-separated edges the black parent and it's red parent will also be affected since the self-edge on these is affected in the elimination step.

Refactorization should be done without having to start over and eliminate the unaffected nodes again. Therefore, the contribution from the unaffected elimination to affected edges need to be stored as well as the value of edges prior to being affected.

5.2.1 Algorithms

The algorithm for the factorization given in Algorithm 5 is changed a bit to store values needed for refactorization. The new algorithm is shown in Algorithm 6.

Algorithm 6 Hierarchical matrix factorization prior to refactorization

```
Initialize H tree
for i = l ... 1 do
    for j = 1 ... 2^(i-1) do
        s_j^i ← MergeRedNodes(r_{0,j}^i, r_{1,j}^i)
    end for
    for j ∈ N_u^i do
        Compress(s_j^i)
        Eliminate_unaffected(s_j^i)
        Eliminate_unaffected(b_j^i)
    end for
    for j ∈ N_a^i do
        Compress_affected(s_j^i)
        Eliminate_affected(s_j^i)
        Eliminate_affected(b_j^i)
    end for
end for
```

$\mathcal{N}_u^i$ is the set of unaffected nodes and $\mathcal{N}_a^i$ is the set of affected nodes of level i. An affected node is defined as a node having at least one edge affected by the change of Z_{LD}.

The methods in Algorithm 6 are explained below:

- Unaffected $\mathcal{N}_u^i$
  - `Compress`: Compression pushes edges to the parent level. Since edges on the parent level isn't affected until the affected super-nodes of this level is eliminated, it is possible to do regular compression for the unaffected nodes.
  - `Eliminate_unaffected`: During the elimination process if an affected edge is updated, the contribution from the elimination is stored in a variable called `unaffected_contribution`, which will be used in the refactorization.
- Affected $\mathcal{N}_a^j$
  - `Compress_affected`: In the compression process it is determined if any of the well-separated edges are affected. If an affected well-separated edge is part of the compression, the edges created to the parent node will be marked as affected, and the compression needs to be recomputed during refactorization. The edges for compression is stored in a list on the super-node to make the refactorization more efficient.
  - `Eliminate_affected`: If the self-edge is affected any new edges or edges updated will be affected. If the self-edge is not affected but another edge is, only the edges created or updated using this edge will be set to affected. The value of an edge before being affected is stored in the variable `pre_affected` to be able to reset the edge before refactorization. For a new edge this will be 0.

The refactorization algorithm is shown in Algorithm 7. The refactorization only eliminate the affected super-nodes, which limits the work needed to recompute the factorization.

Algorithm 7 Hierarchical matrix refactorization

```
1: UpdateAffectedLeafNode
2: for i = l ... 1 do
3:    SetPreAffectedValue(i)
4:    for j ∈ N_a^i do
5:       Compress_recalc(s_j^i)
6:       Eliminate_recalc(s_j^i)
7:       Eliminate_recalc(b_j^i)
8:    end for
9: end for
```

The methods in Algorithm 7 are explained below:

- `UpdateAffectedLeafNode`: The self-edge of the affected red-node at the leaf level is update with the new value of Z_{LD}.
- `SetPreAffectedValue`: The value of affected edges needs to be reset to the value they would be in a regular factorization after the unaffected super-nodes have been eliminated. This resetting will differ for the different type of nodes. The elements are set one level at a time, since super-nodes will be affected by edges of the red siblings, they were merged from.

  For every affected super-node every affected edge is reset to the value it had before it was affected (`pre_affected`). If the edge was created during the merging of red siblings, it depend on the value of the edges from the red siblings. These sub-edges are used to update

the value of the matrix if the sub-edge is affected. This will then be the value the edge had prior to elimination of the unaffected nodes. The contribution from the elimination of unaffected nodes is then added (`unaffected_contribution`), which set the edge to the value it will have after the unaffected nodes have been eliminated.

Affected edges of both black and red-nodes will have no contribution from the unaffected elimination as they are not affected before the elimination of the affected super-nodes. Therefore, every affected edge of black-nodes of level i is set to the pre-affected value and the same is done for every affected edge of red-nodes on level $j-1$.

- Affected $\mathcal{N}_a^j$
  - `Compress_recalc`: This is not run unless the node was marked for recompression in the initial factorization i.e. if there is affected well-separated edges for the super-node. The compression is then rerun as normal and the resulting edges are updated, since they were already created in the initial factorization.
  - `Eliminate_recalc`: The elimination is run as in the initial factorization. The only difference is that all resulting edges have already been created in the initial factorization and the value is only updated if the edge is affected. If the self-edge is affected all edges will be affected and therefore updated, while only some edges are affected otherwise.

5.3 Toy Example

Initially the factorization and refactorization is used on a toy example. The below 16x16 matrix is used in the computations

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 3 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 4 & 3 & 0 & 0 & 1 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 5 & 0 & 0 & 0 & 3 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 3 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 3 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 1 & 1 & 1 & 4 & 3 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 1 & 4 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{5.5}$$

The chosen hierarchical structure will have 4 levels such that each leaf node corresponds to one row/column in the matrix. The affected red-node will be chosen to be the last node i.e. the element updated for the refactorization will be $\mathbf{A}_{16,16}$. This is done such that the order of nodes in the hierarchical tree will be the order that nodes are eliminated for the illustration in this example.

However, in practise the affected red-node could be any of the leaf nodes, and this would just change the order the nodes are eliminated in on the leaf level.

Level 2-4 of the hierarchical tree is shown in Figure 5.6. Parent nodes not used in the factorization (since the super-node had no well-separated edges) is transparent. Nodes affected by the change of $\mathbf{A}_{16,16}$ is marked blue and so is the outline of the affected super-nodes.

Level 2 is the last level used in the factorization, since it has no well-separated edges and therefore the parent nodes are not used. This will generally be the case since it would otherwise mean that the two blocks is separated. This should not be possible if the algorithm is to be used on the big block of $\mathbf{Y}_{nc}$, which is a strongly connected component. If it was possible to split the nodes in 2 and have them not be connected, this would not satisfy the criteria of a strongly connected component.

Even though only 1 out of 8 super-nodes are affected on level 3 it is seen that 3 out of 4 are affected on level 2 increasing the work load considerably on level 2. However the number of nodes is still significantly lower in the factorization, which is also seen in the resulting runtime.

5.4 Implementation and test

The factorization and refactorization was initially implemented in MATLAB using matrices instead of the tree structure introduced here. This was done by extending the matrix when the auxilliary variables was introduced in the compression. However, this showed dissatisfying runtimes with no chance of competing with the standard LU factorization.

Therefore, the implementation was moved to C++ and implemented using the structure with nodes and edges instead of matrices. The implementation is partly made reusing the code made available in [49][1]. The implementation was tested on an Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz.

In testing the accuracy and residual is determined. They are calculated as

$$\text{residual} = \frac{||\mathbf{A}\tilde{x} - b||_2}{||b||_2} \tag{5.6}$$

$$\text{accuracy} = \frac{||x - \tilde{x}||_2}{||x||_2} \tag{5.7}$$

where $\tilde{x}$ is the solution computed using the hierarchical factorization and x is the actual solution to $\mathbf{A}x = b$.

Some problems occurred in testing. The smallest systems worked without problem however for the larger systems the factorization and/or refactorization would fail for some of the higher levels of discretizations. Problems occured due to some issues with certain edges getting very large or very small resulting in issues with either self-edges becoming close to 0 or actually 0.

The test results using the toy example and the systems used in Chapter 4 can be seen in Table 5.1. For the systems the matrix used in the factorization is the biggest block from BTF of $\mathbf{Y}_{nc}$. They are tested at the highest discretization level i.e. using the maximum number of levels possible and some lower number of levels until the resulting residual and accuracy is on a somewhat reasonable level. The chosen tolerance for choosing the rank of the low-rank approximation in the compression is set to $\tau = 10^{-4}$.

[1] http://bitbucket.org/hadip/lorasp

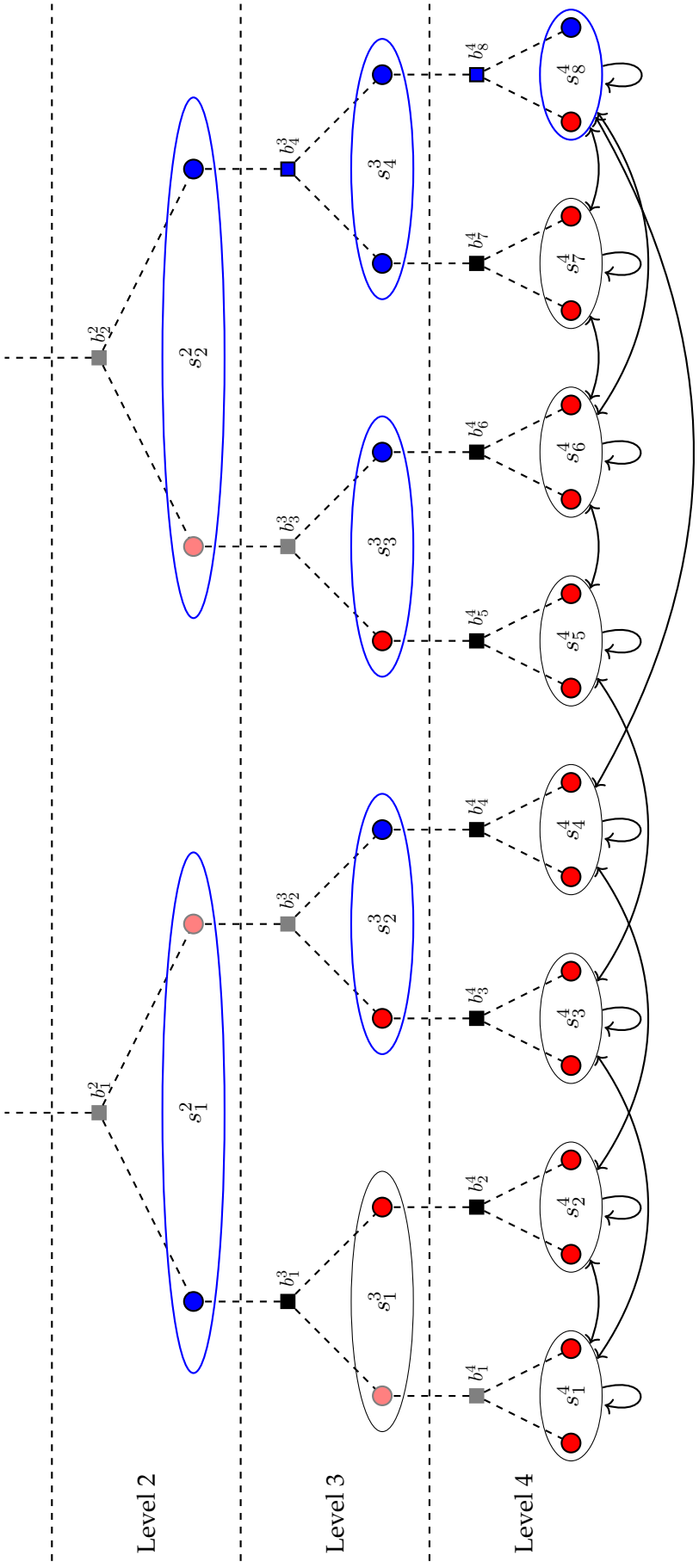


Figure 5.6: Hierarchical structure for the 16x16 matrix given in (5.5). The initial interactions on the leaf level is shown by solid lines and parent relationship by dashed lines. Non-leaf nodes that has no interactions in the factorization are shown as transparent. Affected red-nodes, black-nodes and super-nodes are colored blue.

Table 5.1: Runtime for factorization and refactorization and the residual and accuracy for the refactorization for different system with different levels of discretization and $\tau = 10^{-4}$.

Case	Block size	Levels	Runtime (ms) Algorithm 6	Runtime (ms) Algorithm 7	Residual/ Accuracy
Toy example	16	4	0.30	0.04	10^{-15} / 10^{-14}
Nordic32	13	3	0.08	0.01	10^{-16} / 10^{-16}
Pegase89	76	6	1.29	0.13	10^{-5} / 10^{-3}
		5	0.48	0.02	10^{-16} / 10^{-14}
Pegase1354	888	9	7.74	-	- / -
		8	4.70	0.22	10^{-7} / 10^{-6}
		7	6.13	0.13	10^{-12} / 10^{-10}
PTI-WECC-1648	1062	10	21.02	-	- / -
		9	-	-	- / -
		8	10.37	0.77	10^{-6} / 10^{-4}
		7	10.98	0.24	10^{-14} / 10^{-14}
Polish-Winter99	1425	10	-	-	- / -
		9	-	-	- / -
		8	-	-	- / -
		7	18.35	0.30	10^{-6} / 10^{-3}
		6	48.33	0.36	10^{-14} / 10^{-12}
Polish-Winter03	1476	10	-	-	- / -
		9	9.49	0.86	10^{-6} / 10^{-4}
		8	10.16	-	- / -
		7	19.71	0.19	10^{-9} / 10^{-7}
Pegase2869	1462	10	19.13	-	- / -
		9	10.94	0.86	10^{30} / 10^{31}
		8	11.07	0.38	10^{2} / 10^{4}
		7	20.11	0.25	10^{-9} / 10^{-7}
Polish-Winter07	1800	10	-	-	- / -
		9	15.34	2.19	10^{-4} / 10^{-1}
		8	15.27	0.75	10^{-4} / 10^{-2}
		7	31.68	0.37	10^{-11} / 10^{-8}

The residual and accuracy is determined by computing a random x and computing $b = Ax$, which is then fed to the solve algorithm. The accuracy is computed for the refactorization but could just as well be computed for the initial factorization. The accuracy of both should be the same, when feeding the refactorization the same value for Z_{LD}. However, in testing for a couple of systems this did not happen and is another error that needs to be investigated.

Table 5.1 shows the issues with the implementation where it seems a bit random, which number of levels that result in issues, and for Pegase2869 the refactorization can be computed for more levels than similar sized systems, but the error is quite big for some of them. The runtimes show that refactorization (Algorithm 7) is considerablely faster than the initial factorization (Algorithm 6).

It can be seen that for the larger systems the lowest discretization for Algorithm 6 is slow, then it gets a little faster and then it gets slower again. The reason for this is that a the lowest level there is a lot of nodes to be handled, which takes time. For the higher levels there is fewer nodes but the blocks are a lot larger making the method slow again. In between there will be a level that is faster, since there is a "good" amount of nodes compared to the block size.

The refactorization on the other hand gets faster as the number of levels gets lower. This happens because the amount of nodes gets lower with the number of level, and thereby fewer nodes can be affected. This means that fewer nodes have to be recomputed in the refactorization, resulting in lower runtimes.

Therefore, the optimal number of levels for a system will have to be investigated and will depend on the number of times the factorization is to be recomputed and what level of error can be accepted. If the factorization is recomputed many times it might be feasible to choose a lower number of levels to ensure a fast refactorization and a low error.

One thing to note is that the refactorization algorithm only tracks the nodes affected by one specific change in the diagonal. In practise it can be used to refactorize for change in any diagonal element belonging to the same super-node so depending on the discretization it can be used for many nodes. However, the initial factorization has to be recomputed if a diagonal element belonging to another super-node is to be changed. This might also be an argument for the chosen discretization, since a lower number of level makes it possible to use the same factorization to recompute for several nodes.

The runtimes using the the original factorization (Algorithm 5) is similar to the runtimes of Algorithm 6, even though some extra work is done in Algorithm 6 to ensure data required for the refactorization. For a couple of systems the original factorization was a bit slower than Algorithm 6. The reason for this is the order of elimination. Affected nodes are eliminated last and this will for the larger systems result in a changed order of elimination for some of the levels. The order of elimination will result in a different number of edges eliminated and compressed, since the fill-in changes depending on the ordering of the matrix, which affect runtime.

5.5 Conclusion

The hierarchical structure makes it possible to factorize a matrix and avoid fill-in by approximating this with a low-rank approximation such as SVD. The hierarchical structure makes it possible to track what elements are affected by changing one diagonal element of the matrix. This way refactorization can be done fast and efficiently.

The level of discretization determines the runtime for both the initial factorization and the refactorization. Refactorization gets faster with fewer levels, whereas the initial factorization is slowest at the highest and lowest levels of discretization and the optimal level therefore has to be determined for each system.

The implemented algorithm had some different issues for larger systems, which still needs to be investigated. It has to be determined what gives rise to this and how it can be mitigated such that the algorithms work for all matrices at all levels of discretization. Furthermore, it would be interesting to see if the solve could be modified such that for the refactorization hopefully could be done faster as well.

CHAPTER 6
Conclusion and Future Work

This chapter concludes the research conducted in the PhD project, which has been presented in this thesis. It ends with a discussion of future work to be considered.

6.1 Conclusion

This PhD project was concerned with the development of high performance algorithms for real-time stability and security assessment for future power systems. To accomplish this 4 goals was listed to help further the development of such methods. This included investigation of critical elements of methods, optimization of algorithms to achieve real-time computation by exploiting graph theoretical properties, avoiding repeated calculations and parallel processing. Finally, methods are to be tested.

A long range of real-time stability and security assessment methods make use of Thévenin equivalents as it is efficient and gives a credible representation of the system. Therefore Thévenin equivalent computations were investigated to determine how these could be optimized. The computations are optimally done by utilizing a factorization to decompose the system.

Initially it was investigated what effect the chosen factorization method had on the computations. The factorization methods used was the standard LU factorization in MATLAB (UMFPACK), KLU and ILU. UMFPACK and KLU are direct methods computing the exact solution, whereas ILU computes an approximation and requires a tolerance level in computations.

Testing showed that the factorization step was in fact a negligible part of the computations and the computationally heavy part was to determine the impedance matrix for the current sources. For UMFPACK and KLU the accuracy of solutions was the same except for numerical cancellation, whereas the accuracy and runtime of ILU depended on the tolerance. To get runtimes competible with UMFPACK and KLU the error using ILU was dissatisfying making ILU have limited applicability for this kind of computations.

It was shown how the density of the impedance matrix for the current sources and in consequence the coefficient matrix for determining Thévenin voltages could be determined using BTF for the admittance matrix for the current sources. Furthermore, one large block was seen to dominate the BTF and thereby being a dominating factor in computations.

The *factor-solve* method was developed on the basis of the conclusion from the investigation. It avoids computing the impedance matrix for the current sources to reduce both runtime and memory requirements. The method takes advantages of the block back substitution of KLU and uses the factorization directly instead of computing the coefficient matrix.

The complexity of the algorithm for computing the factorization and the Thévenin impedance was shown to be the same as for the reference method, however the *factor-solve* method is more easily parallelized making it considerably faster for larger systems.

This biggest advantage was being able to compute Thévenin voltages in linear time compared to the close to quadratic complexity for the reference method. This made it possible to compute Thévenin voltages in 6ms for a 30.000 bus system, which ensures that they can be recomputed for every measurement provides by PMUs, that normally arrive every 16-20ms.

Furthermore, the memory requirements for the *factor-solve* method was much lower. The *factor-solve* method only stores the factorization which takes up a few megabytes for the largest system compared to several gigabytes for the coefficient matrix in the reference method.

During the SARP project, which this PhD is a part of, a method was developed for determining the voltage stability boundary by accounting for non-linearity in the Thévenin voltage. This determines the maximum power transfer to a load by changing the load impedance multiple times and ideally this should be found accurately in as few steps as possible.

A binary search is suggested to find this maximum, and it is furthermore improved by including a second order polynomial fitting as part of the search. This is done since nodes not limited by a generator starting to loose synchronism will have a shape similar to that of a second order polynomial. Therefore, it is possible to find the maximum of the second order polynomial to optimize the search for the maximum power transfer for the node.

Runtimes are lowered by using the binary search and the maximum determined compared to a naïve algorithm is more accurate. The complexity is however still the same for all implementations and shows that each step computed should be optimized for better performances.

Block-wise computations are therefore tested to achieve lower runtimes. Only nodes belonging to the same block in BTF has to be considered in the computations for that specific load. However, as determined earlier BTF is dominated by a big block, and this scales with system size, meaning that even though runtimes are lower it still has the same complexity as before.

To find the maximum power transfer the admittance matrix has to be refactorized multiple times. Therefore, the hierarchical structure is investigated to create a fast refactorization method to be used on the biggest block in BTF. The hierarchical structure is a hierarchical tree, where the leaf nodes is a discretization of the matrix. The higher levels are used, when fill-in generated in the factorization by eliminataion of nodes is pushed to the parent level and approximated using SVD.

The number of levels i.e. the level of discritization determines the runtime for both the initial factorization and the refactorization. Both the number of nodes and the resulting block size affect runtime. For the initial factorization the runtime is lowest somewhere in between the highest and lowest discretization. High discretization means a lot of nodes computed, while low discretization means that block gets so large that it takes too long to compute. For the refactorization the lower the discritization the faster, since fewer nodes are affected by the change in load impedance as the amount of nodes gets lower.

The implementation of the algorithms for the hierarchical factorization and refactorization had some different issues, which needs to be determined. The forward and backwards traversal of the solve step was not modified for recalculation.

6.2 Future Work

Based on the research conducted in this PhD project some topics was identified, which could enhance the work presented here. The topics are explained and listed below:

- **Thévenin equivalent computations**
  - *Investigation of power systems structure and the affect on computations*
    In the different methods for computing the Thévenin equivalents it was seen that the complexity of the methods was highly dependent on the fill-in of the matrices. For systems of similar size this could differ and therefore it would be interesting to investigate what properties of the admittances matrices that effect runtimes more than other. This could make it possible to potentially modify computation or matrices to ensure better runtimes.
  - *Implement factor-solve method on SW-platform and use in real-time stability methods*
    It would be interesting to use the *factor-solve* method in stability methods to test their runtime using this and potentially identify if other areas of the methods should be optimized. The methods would ideally be implemented on a SW-platform as the one developed in the SOSPO project to test the method in a real-time setting.
- **Voltage stability boundary monitoring**
  - *Iteration algorithms tested for systems close to boundary*
    The iteration algorithms was tested for systems far from the voltage instability boundary, and it would therefore be interesting to test their performance as the systems nears instability. This could be done in a real-time setting on a SW-platform, such that the algorithms are fed a range of snapshot generated in a time-domain simulation and tested as the system nears the boundary.
  - *Maximum power transfer to load when boundary has been passed*
    The algorithms implemented all assume that the point of maximum power transfer to the load hasn't occurred yet and therefore the distance to the instability boundary will be 0, when the boundary has passed. It would be interesting to determine the distance to the actual maximum power transfer in these cases, since this could potentially be used to determine a control measure to return the system to a secure state.
- **Hierarchical matrix structure**
  - *Code debugging for larger systems*
    The algorithms implemented in C++ resulted in some errors for larger systems and some levels of discretization. Some of the problems occurred either when edges were either extremely large or below machine precision. This could be a result of the values in the test system being quite different. Self-edges are inverted during elimination of the node, and numerically small self-edge might therefore result in a large values of new edges created. Some initial testing where nodes with numerically small self-edges was eliminated last did not result in any improvement. Further testing therefore is needed to determine what happens for these systems.

- *Efficient solve for the refactorization*
  Due to the issues in the factorization and refactorization mentioned above the solve step was not investigated. The solve step has both a forward and backwards traversal of nodes. The idea is that it might be possible to do something similar in the first traversal of nodes in the solve as in the refactorization making it faster. The backwards traversal would have to be recomputed fully for any refactorization.
- *Testing in voltage stability method*
  The refactorization was not tested in the voltage stability method, which it was intended for and it would be interesting to see if it would have any effect on runtime to use this here. Furthermore, testing should check the accuracy of the maximum power transfer found to ensure that errors due to the low-rank approximation is not too big.

Bibliography

[1] European Commission. Clean energy for all Europeans. Technical report, 2019.

[2] Danish Energy Agency. Denmark's Energy and Climate Outlook 2020 (DECO20). Technical report, 2020.

[3] Danish Energy Agency. Energy Strategy 2050 - from coal, oil and gas to green energy. Technical report, 2011.

[4] Fangxing Li, Wei Qiao, Hongbin Sun, Hui Wan, Jianhui Wang, Yan Xia, Zhao Xu, and Pei Zhang. Smart Transmission Grid: Vision and Framework. *IEEE Transactions on Smart Grid*, 1(2):168–177, 9 2010.

[5] Arun G. Phadke. Synchronized phasor measurements - A historical overview. In *Proceedings of IEEE/PES Transmission and Distribution Conference and Exhibition*, Yokohama, Japan, 2002. IEEE.

[6] Arun G. Phadke and James S. Thorp. *Synchronized Phasor Measurements and Their Applications*. Power Electronics and Power Systems. Springer US, Boston, MA, 2008.

[7] Arun G. Phadke, Hector Volskis, Rui Menezes de Morares, Tianshu Bi, R. N. Nayak, Y. K. Sehgal, Subir Sen, Walter Sattinger, Enrique Martinez, Olof Samuelsson, Damir Novosel, Vahid Madani, and Yuri A. Kulikov. The Wide World of Wide-Area Measurement. *IEEE Power and Energy Magazine*, 6(5):52–65, 9 2008.

[8] Srdjan Skok and Igor Ivankovic. Applications based on PMU technology for improved power system utilization. In *Proceedings of 2007 IEEE Power Engineering Society General Meeting*, Tampa, FL, USA, 2007. IEEE.

[9] Vladimir Terzija, Gustavo Valverde, Devu Cai, Pawel Regulski, Vahid Madani, John Fitch, Srdjan Skok, Miroslav M. Begovic, and Arun G. Phadke. Wide-Area Monitoring, Protection, and Control of Future Electric Power Networks. *Proceedings of the IEEE*, 99(1):80–93, 2011.

[10] Yousu Chen, Zhenyu Huang, Yan Liu, Mark J. Rice, and Shuangshuang Jin. Computational Challenges for Power System Operation. In *Proceedings of 45th Hawaii International Conference on System Sciences (HICSS)*, pages 2141–2150, Maui, HI, USA, 2012.

[11] Prabha Kundur, John Paserba, Venkat Ajjarapu, Göran Andersson, Anjan Bose, Claudio Canizares, Nikos Hatziargyriou, David Hill, Alex Stankovic, Carson Taylor, Thierry Van Cursem, and Vijay Vittal. Definition And Classification of Power System Stability IEEE/-CIGRE Joint Task Force on Stability Terms and Definitions. *IEEE Transactions on Power Systems*, 19(3):1387–1401, 2004.

[12] Guangya Yang, Hjörtur Jóhannsson, Morten Lind, Rodrigo Garcia-Valle, Mogens Blanke, Arne Hejde Nielsen, and Jacob Østergaard. Addressing the security of a future sustainable power system: The Danish SOSPO project. In *Proceedings of 9th IET International Conference on Advances in Power System Control, Operation and Management (APSCOM)*, Hong Kong, China, 2012.

[13] Angel Perez, Hjörtur Jóhannsson, Pieter Vancraeyveld, and Jacob Østergaard. Suitability of voltage stability study methods for real-time assessment. In *Proceedings of 4th IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Copenhagen, Denmark, 10 2013. IEEE.

[14] Angel Perez, Hjörtur Jóhannsson, and Jacob Østergaard. Evaluation of enhancements to Thevenin equivalent based methods for real-time voltage stability assessment. In *Proceedings of 5th IEEE PES Innovative Smart Grid Technologies (ISGT Europe)*, Istanbul, Turkey, 2014. IEEE.

[15] Angel Perez, Hjörtur Jóhannsson, and Jacob Østergaard. Wind farms generation limits and its impact in real-time voltage stability assessment. In *Proceedings of 11th IEEE PowerTech*, Eindhoven, Netherlands, 2015. Institute of Electrical and Electronics Engineers Inc.

[16] Angel Perez, Hjörtur Jóhannsson, Jacob Østergaard, Mevludin Glavic, and Thierry Van Cutsem. Improved Thévenin equivalent methods for real-time voltage stability assessment. In *Proceedings of 4th IEEE International Energy Conference (ENERGYCON)*, Leuven, Belgium, 2016. IEEE.

[17] Tilman Weckesser, Hjörtur Jóhannsson, Jacob Østergaard, and Thierry Van Cutsem. Sensitivity based assessment of transient voltage sags caused by rotor swings. In *Proceedings of 18th Power Systems Computation Conference*, Wroclaw, Poland, 2014. IEEE.

[18] Jakob Glarbo Møller, Hjörtur Jóhannsson, and Jacob Østergaard. Computation of steady state nodal voltages for fast security assessment in power systems. In *Proceedings of 2014 Electric Power Quality and Supply Reliability Conference (PQ)*, pages 51–55, Rakvere, Estonia, 2014. IEEE.

[19] Jakob Glarbo Møller, Hjörtur Jóhannsson, and Jacob Østergaard. Thevenin equivalent method for dynamic contingency assessment. In *Proceedings of IEEE Power & Energy Society's General Meeting*, Denver, CO, USA, 2015.

[20] Evgenia Dmitrova, Hjörtur Jóhannsson, and Arne Hejde Nielsen. Assessment of the impact that individual voltage source has on a generator's stability. In *Proceedings of 10th International Power & Energy Conference (IPEC)*. IEEE, 2012.

[21] Evgenia Dmitrova, Martin Lindholm Wittrock, Hjörtur Jóhannsson, and Arne Hejde Nielsen. Early Prevention Method for Power System Instability. *IEEE Transactions on Power Systems*, 30(4):1784–1792, 2015.

[22] Jakob Glarbo Møller, Hjörtur Jóhannsson, and Jacob Østergaard. Super-Positioning of Voltage Sources for Fast Assessment of Wide-Area Thévenin Equivalents. *IEEE Transactions on Smart Grid*, 8(3):1488–1493, 2017.

[23] Fuzhen Zhang. *The Schur Complement and Its Applications*. Springer, 2005.

[24] Florian Dorfler and Francesco Bullo. Kron Reduction of Graphs With Applications to Electrical Networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2013.

[25] Yunfei Wang, Iraj Rahimi Pordanjani, Weixing Li, Wilsun Xu, Tongwen Chen, Ebrahim Vaahedi, and Jim Gurney. Voltage stability monitoring based on the concept of coupled single-port circuit. *IEEE Transactions on Power Systems*, 26(4):2154–2163, 2011.

[26] Stefan Sommer and Hjörtur Jóhannsson. Real-time thevenin impedance computation. In *Proceedings of IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*, Washington, DC, USA, 2013.

[27] Stefan Sommer, Andreas Aabrandt, and Hjörtur Jóhannsson. Reduce-Factor-Solve for Fast Thevenin Impedance Computation and Network. *IET Generation, Transmission & Distribution*, 11 2018.

[28] Haoyu Yuan and Fangxing Li. A comparative study of measurement-based Thevenin equivalents identification methods. In *Proceedings of 46th North American Power Symposium (NAPS)*, Pullman, WA, USA, 2014.

[29] Luc Giraud, Azzam Haidar, and Yousef Saad. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D. *Numerical Mathematics Theory Methods and Applications*, 3(3):276–294, 2010.

[30] Hjörtur Jóhannsson. *Development of early warning methods for electric power systems*. PhD thesis, Technical University of Denmark, 2011.

[31] Timothy A. Davis. Algorithm 832: UMFPACK V4.3 - An unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.

[32] Timothy A Davis. Algorithm 907 : KLU , A Direct Sparse Solver for Circuit Simulation Problems. *ACM Transactions on Mathematical Software*, 37(3):1–17, 2010.

[33] Hugo Morais, Pieter Vancraeyveld, Allan Henning Birger Pedersen, Morten Lind, Hjörtur Jóhannsson, and Jacob Østergaard. SOSPO-SP: Secure Operation of Sustainable Power Systems Simulation Platform for Real-Time System State Evaluation and Control. *IEEE Transactions on Industrial Informatics*, 10(4):2318–2328, 2014.

[34] Hjörtur Jóhannsson, Hugo Morais, Allan Henning Birger Pedersen, Qiuwei Wu, and Dean Ouellette. SW-platform for R&D in Applications of Synchrophasor Measurements for Wide-Area Assessment, Control and Visualization in Real-Time. *CIGRE US National Committee 2014 Grid of the Future Symposium*, 2014.

[35] Christina Hildebrandt, Bahtiyar Can Karatas, Jakob Glarbo Møller, and Hjörtur Jóhannsson. Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment. In *Proceedings of 20th Power Systems Computation Conference (PSCC)*, Dublin, Ireland, 2018.

[36] Christina Hildebrandt Lüthje Jørgensen, Jakob Glarbo Møller, Stefan Sommer, and Hjörtur Jóhannsson. A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment. *IEEE Transactions on Power Systems*, 2019.

[37] Christina Hildebrandt Lüthje Jørgensen, Bahtiyar Can Karatas, Hjörtur Jóhannsson, and Stefan Sommer. Binary Search and Fit Algorithm for Improved Voltage Stability Boundary Monitoring. In *Proceedings of 9th IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Bucharest, Romania, 2019.

[38] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An Approximate Minimum Degree Ordering Algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[39] Timothy A. Davis. *Direct Methods For Sparse Linear Systems*. SIAM, Gainesville, Florida, 2006.

[40] Youcef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[41] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sanchez, and Robert John Thomas. MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.

[42] CIGRÉ TF38.02.08. Long Term Dynamics Phase II, Final Report. Technical report, 1993.

[43] IEEE Standards Association. C37.118.1-2011 IEEE Standard for Synchrophasor Measurements for Power Systems. Technical report, 2011.

[44] P. Kessel and H. Glavitsch. Estimating the Voltage Stability of a Power System. *IEEE Trannsactions on Power Delivery*, 1(3):346–354, 1986.

[45] Hjörtur Jóhannsson, Arne Hejde Nielsen, and Jacob Østergaard. Wide-Area Assessment of Aperiodic Small Signal Rotor Angle Stability in Real-Time. *IEEE Transactions on Power Systems*, 28(4):4545–4557, 11 2013.

[46] Tilman Weckesser, Hjörtur Jóhannsson, and Jacob Østergaard. Real-Time Remedial Action Against Aperiodic Small Signal Rotor Angle Instability. *IEEE Transactions on Power Systems*, 31(1):387–396, 2016.

[47] Bahtiyar Can Karatas, Hjörtur Jóhannsson, and Arne Hejde Nielsen. Improved Voltage Stability Boundary Monitoring by Accounting for Variations in Thevenin Voltage Magnitude. In *Proceedings of 8th Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, Sarajevo, Bosnia and Herzegovina, 2018.

[48] Bahtiyar Can Karatas, Hjörtur Jóhannsson, and Arne Hejde Nielsen. Voltage stability assessment accounting for non-linearity of Thévenin voltages. *IET Generation, Transmission & Distribution*, 14(16):3338–3345, 8 2020.

[49] Hadi Pouransari, Pieter Coulier, and Eric Darve. Fast Hierarchical Solvers for Sparse Matrices using Extended Sparsification and Low-Rank Approximation. *SIAM Journal on Scientific Computing*, 39(3):A797–A830, 2017.

A. First Appendix

Collection of relevant publications

[**Pub. A**] Christina Hildebrandt, Bahtiyar Can Karatas, Jakob Glarbo Møller, and Hjörtur Jóhannsson. Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment. In *Proceedings of 20th Power Systems Computation Conference (PSCC)*, Dublin, Ireland, 2018

[**Pub. B**] Christina Hildebrandt Lüthje Jørgensen, Jakob Glarbo Møller, Stefan Sommer, and Hjörtur Jóhannsson. A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment. *IEEE Transactions on Power Systems*, 2019

[**Pub. C**] Christina Hildebrandt Lüthje Jørgensen, Bahtiyar Can Karatas, Hjörtur Jóhannsson, and Stefan Sommer. Binary Search and Fit Algorithm for Improved Voltage Stability Boundary Monitoring. In *Proceedings of 9th IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Bucharest, Romania, 2019

[Pub. A] Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment

Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment

Christina Hildebrandt, Bahtiyar Can Karatas, Jakob Glarbo Møller and Hjörtur Jóhannsson
Department of Electrical Engineering
Technical University of Denmark
Kgs. Lyngby, Denmark
{chhil, bcakara, jglmo, hjjo}@elektro.dtu.dk

***Abstract*—Thévenin equivalents are used by a range of power system stability indicators, such as the L-index for voltage stability and the aperiodic small signal rotor angle stability indicator. This paper investigates the effect of using different factorization methods for computing coefficients for wide-area Thévenin equivalents. Direct and incomplete factorization methods are compared with respect to runtime, accuracy and amount of fill-in. The paper introduces a proof that the block triangular form of bus admittance matrices will have no non-zero entries in the off-diagonal. KLU factorization is found to perform almost twice as fast as the standard LU factorization with no cost of accuracy. It is, however, shown that the largest computational workload is associated with dense matrix multiplications. An incomplete method reduces the fill-in of coefficient matrices at the cost of accuracy in Thévenin voltages. It is shown, that inaccuracies are amplified as the L-index approaches the stability limit.**

***Index Terms*—Power system analysis computing, Power system stability, Real-time assessment, Thévenin equivalent, Wide-area monitoring**

I. Introduction

Thévenin equivalent methods have been proposed for reliable assessment of several modes of power system instability - including long-term voltage instability and steady-state instability of generators [1], [2]. These two types of instability can be strongly connected, since long-term voltage instability is provoked by trying to supply more power to a load than the maximum power transfer capabilities of the system, while aperiodic generator instability is driven by the maximum power transfer from a generator to the system. These limits describe the bounds of stable steady-state operation of power systems and may be used to identify the set of feasible solutions of power flow problems. Fast and efficient computation of Thévenin equivalents is a necessary condition for the application of such indicators in real-time and on larger scale. With the increasing usage of phasor measurement units (PMU) [3], [4], complex bus voltages and complex branch currents can be obtained at the rate of system frequency and together with information of the system topology, the Thévenin equivalent methods can be applied in real-time.

When using Thévenin equivalent methods one may choose from two general approaches for obtaining the equivalent parameters; methods based on local measurements [5], and methods based on the full state of the system (wide-area measurements) [6]. Both approaches have their advantages and drawbacks. Local assessment is better suited for use in distributed controllers while wide-area methods may be a better choice for central monitoring and control or sensitivity calculations. The scope of this paper is limited to the wide-area methods and it is assumed that a true system state is available. Thereby this paper focus on computational performance only. For studies on the impact of measurement uncertainty the reader is referred to [7].

The factorization method has previously been proven to have an impact on computations for determining Thévenin impedances [8]. However, when determining coefficients for super-position the use of different factorization methods has not previously been evaluated. The super-position principle can be used to determine the contribution of each voltage- or current source on the Thévenin voltage.

In [6] it is shown, how a Schur complement of the bus admittance matrix can be exploited to efficiently obtain Thévenin equivalents seen from all nodes of a meshed system. The Schur complement is in general considered to be dense [9], but as noted in [6], several of the fill-ins are small and seem quite insignificant. This observation is here used to give room for increased degree of sparsity.

This paper investigates the effect of applying different factorization methods when obtaining coefficients for super-position for wide-area Thévenin equivalents. The highest possible degree of sparsity is pursued in order to speed up computations. The investigation will focus on the resulting runtime as well as accuracy. Candidates for factorization methods are the standard LU factorization in MATLAB (UMFPACK) [10], "Clark Kent" LU factorization (KLU) [11] and incomplete LU factorization (ILU) [12]. Ordering scheme candidates are Block Triangular Form and Approximate Minimum Degree (AMD).

The special attributes of the KLU algorithm makes it unsuited for some inverse problems. This paper contributes

with a proof of how KLU can be used for calculations involving Thévenin equivalents. The proof shows that the block triangular form of a bus admittance matrix has no non-zero entries in the off-diagonal blocks. This proof has not been provided in earlier publications.

Section II introduces the voltage stability indicator for loads from [13] and aperiodic small signal rotor angle stability margin for generators from [2] and explains how a Schur complement is used to obtain the Thévenin equivalents as proposed in [6]. Section III introduces the different factorization methods and proves why KLU can be used in computation of Thévenin equivalents. Section IV evaluates the methods by computational time, degree of sparsity and their accuracy of the resulting stability indicators. Furthermore, the error in Thévenin voltages introduced by ILU is investigated. Section V discusses the results and gives some perspectives on further work, while section VI concludes the paper.

II. Background

A. Voltage stability indicator

A local voltage stability margin for loads is defined in [13]. For a node i the local indicator L_i is defined by the node voltage $\overline{V}_i$ and the Thévenin voltage $\overline{V}_{th,i}$ seen from node i

$$L_i = \left| 1 - \frac{\overline{V}_{th,i}}{\overline{V}_i} \right|. \tag{1}$$

For stable situations $L_i \leq 1$ must not be violated for any node i. Hence the global indicator for voltage stability of the entire system is given by

$$L = \max_{i \in cs} \{L_i\}, \tag{2}$$

where cs represent the loads. Voltage instability may be inferred in the case where $L > 1$.

The importance of accuracy when assessing Thévenin equivalents for system stability studies can be easily demonstrated by adding a random vector-error $\overline{\varepsilon}$ to the Thévenin voltage. This gives the following L-index;

$$L_i = \left| 1 - \frac{\overline{V}_{th,i} + \overline{\varepsilon_i}}{\overline{V}_i} \right| = \left| 1 - \frac{\overline{V}_{th,i}}{\overline{V}_i} - \frac{\overline{\varepsilon_i}}{\overline{V}_i} \right|. \tag{3}$$

The closer the system is to voltage instability the lower the magnitude of the node voltage, V_i will be. Therefore, the error will have a larger influence close to the stability boundary. Figure 1 shows the calculation of the L-index represented with phasors. The worst case is an error vector that is orthogonal to $\frac{\overline{V}_{th,i}}{\overline{V}_i}$. This will result in an L-index that indicate the system is more stable, than it actually is.

B. Aperiodic small signal rotor angle stability margin

In [2] a power margin of the injected power to the maximum power injection is defined. This margin describes the distance from a generator's operating point to the stability boundary of aperiodic small signal rotor angle stability. In [14] this margin is reformulated in terms of voltages instead of impedances. A percentage margin to the maximum injectable power is then defined as

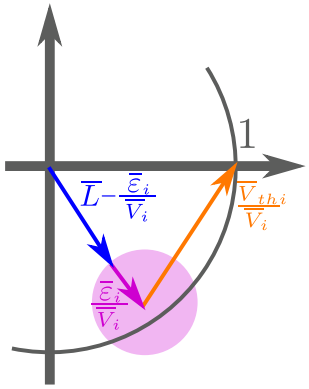


Figure 1. L-index for node i, where an error $\overline{\varepsilon_i}$ affect the resulting stability indicator. The L-index is represented as a phasor in the complex plane. The local voltage stability indicator for the node will be the magnitude of this.

$$\%\Delta P_{inj} = \frac{P_{inj,max} - P_{inj}}{P_{inj}} \cdot 100\% \tag{4}$$

$$= \frac{\cos(\delta + \phi_{th}) + 1}{1 + \frac{V}{V_{th}} \cos \phi_{th}} \cdot 100\%, \tag{5}$$

where the generator is represented as a voltage source $V \angle \delta$ and the remaining grid by its Thévenin equivalent with a voltage source of magnitude V_{th} and an impedance $Z_{th} \angle \phi_{th}$. $\overline{V}_{th}$ is used as the phase angle reference.

If, for any generator, $\%\Delta P_{inj} < 0$, that generator will lose synchronism. This may destabilize the entire system. Therefore, the overall systems stability margin is defined as the minimum $\%\Delta P_{inj}$.

C. Schur complement and Thévenin equivalents

An approach for computing Thévenin equivalents is described in [6], which uses a Schur complement to optimize the calculations. Thévenin equivalents consist of a Thévenin impedance $\overline{Z}_{th}$ and Thévenin voltage $\overline{V}_{th}$. The Thévenin equivalent seen from node i will satisfy

$$\overline{V}_{th,i} = \overline{V}_i - \overline{Z}_{th,i} \overline{I}_i \tag{6}$$

$\overline{V}_i$ is node voltage and $\overline{I}_i$ is current injected at node i.

Nodes in a network can be partitioned in to two sets - current sources (cs) and voltage sources (vs). A floating node may be seen as a current source injecting 0 current. Loads are represented as current sources and generators with automatic voltage regulator (AVR) or internal voltages of manually excited machines as voltage sources. This distinction of nodes is important as it is recalled that the L-index is an indicator of voltage instability of load buses while the aperiodic small signal rotor angle stability margin is associated with the maximum power transfer from generators.

The admittance matrix for the system can then be block-wise partitioned as follows

$$\begin{bmatrix} I_{cs} \\ I_{vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{cs} & \mathbf{Y}_{v \to c} \\ \mathbf{Y}_{c \to v} & \mathbf{Y}_{vs} \end{bmatrix} \begin{bmatrix} V_{cs} \\ V_{vs} \end{bmatrix} \tag{7}$$

Eliminating V_{cs} from (7) yields

$$I_{vs} = \mathbf{Y}_{eq} V_{vs} - \mathbf{Q}_{ac} I_{cs} \tag{8}$$

with

$$\mathbf{Y}_{eq} = \mathbf{Y}_{vs} - \mathbf{Y}_{c\to v}\mathbf{Y}_{cs}^{-1}\mathbf{Y}_{v\to c} \tag{9}$$
$$\mathbf{Q}_{ac} = -\mathbf{Y}_{c\to v}\mathbf{Y}_{cs}^{-1} \tag{10}$$

$\mathbf{Y}_{eq}$ is the Schur complement and $\mathbf{Q}_{ac}$ is the accompanying matrix. This reduction of the network is also known as *Kron reduction* [15].

The Thévenin impedances seen from node i are determined as the diagonal of the impedance matrix

$$Z_{th,i} = \begin{cases} \mathbf{Z}_{cs}(i,i) & i \in cs \\ \mathbf{Y}_{eq}(i,i)^{-1} & i \in vs \end{cases} \tag{11}$$

where $\mathbf{Z}_{cs} = \mathbf{Y}_{cs}^{-1}$ [6].

Using the definition for Thévenin voltage given in (6) and the above network equations the Thévenin voltage for the cs and vs nodes respectively are defined as

$$V_{th,cs} = -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}V_{vs} + (\mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs}))I_{cs} \tag{12}$$
$$V_{th,vs} = (\mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq})V_{vs} + \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}I_{cs} \tag{13}$$

$\mathcal{I}$ is the identity matrix and $\mathcal{D}(Z_{th})$ is the diagonalization of the vector Z_{th} into a diagonal matrix. (12-13) can be written on the form

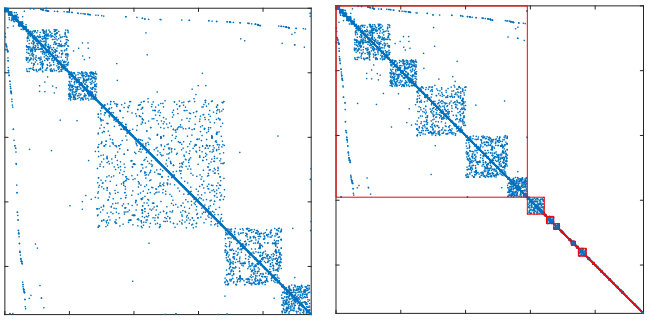
$$\begin{bmatrix} V_{th,cs} \\ V_{th,vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix} \begin{bmatrix} I_{cs} \\ V_{vs} \end{bmatrix} \tag{14}$$

with

$$\mathbf{Z}_c = \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs}) \tag{15}$$
$$\mathbf{K}_{v\to c} = -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c} \tag{16}$$
$$\mathbf{Z}_{c\to v} = \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac} \tag{17}$$
$$\mathbf{K}_v = \mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq} \tag{18}$$

The coefficients $\mathbf{K}_v$ were introduced in [6] and [16] as the grid transformation coefficients (GTC).

Algorithm 1 contains the steps for obtaining the coefficients and Thévenin impedances. The LU-factorization of $\mathbf{Y}_{cs}$ is used to optimize the computations. This approach is used in [6] and [8], since $\mathbf{L}$ and $\mathbf{U}$ are computationally more efficient to invert than the full matrix.



(a) (b)

Figure 2. (a) Sparsity pattern for $\mathbf{Y}_{cs}$ for the test system Polish-Winter03, Table I, used in the Section IV and (b) $\mathbf{Y}_{cs}$ on block triangular form

Algorithm 1 Thévenin equivalents

$\mathbf{L}_{cs}, \mathbf{U}_{cs} \leftarrow$ factorization of $\mathbf{Y}_{cs}$
$\mathbf{U}_{\mathbf{Z}_{cs}} \leftarrow$ solve$(\mathbf{L}_{cs}, \mathcal{I})$
$\mathbf{L}^T_{\mathbf{Z}_{cs}} \leftarrow$ solve$(\mathbf{U}^T_{cs}, \mathcal{I})$
$\mathbf{Z}_{cs} \leftarrow \mathbf{L}_{\mathbf{Z}_{cs}}\mathbf{U}_{\mathbf{Z}_{cs}}$
$Z_{th,cs} \leftarrow \mathcal{D}(\mathbf{Z}_{cs})$
$\mathbf{Q}_{ac} \leftarrow -\mathbf{Y}_{c\to v}\mathbf{Z}_{cs}$
$\mathbf{Y}_{eq} \leftarrow \mathbf{Y}_{vs} + \mathbf{Q}_{ac}\mathbf{Y}_{v\to c}$
$Z_{th,vs} \leftarrow \mathcal{D}(\mathbf{Y}_{eq})^{-1}$
$\mathbf{Z}_c \leftarrow \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs})$
$\mathbf{K}_{v\to c} \leftarrow -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}$
$\mathbf{Z}_{c\to v} \leftarrow \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}$
$\mathbf{K}_v \leftarrow \mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq}$
$Z_{th} \leftarrow \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}$
$\mathbf{K} \leftarrow \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix}$
return Z_{th} and $\mathbf{K}$

$\mathcal{D}(\mathbf{X})$ is a vector containing the diagonal of the matrix $\mathbf{X}$, while $\mathcal{D}(X)$ a diagonal matrix with the vector X along the diagonal.

The Thévenin voltages can be determined by (14) using the coefficients, $\mathbf{K}$, the current injected at cs nodes, $\mathbf{I}_{cs}$, and the voltage at vs nodes, $\mathbf{V}_{vs}$.

III. Factorization Methods

Different factorization methods will be used in Algorithm 1. The different methods investigated are

- UMFPACK with AMD
- KLU using block triangular form
- ILU

UMFPACK with AMD is the standard LU factorization of a sparse matrix in MATLAB. AMD is used prior to factorization, where the matrix is permuted to reduce the computation time and the fill-in in the factorization [17].

KLU is a factorization method optimized for sparse systems [11]. The method is part of the library SuiteSparse [18]. KLU convert the system to block triangular form, where the diagonal of the resulting matrix will contain square matrices with zero-free diagonal and the off-diagonal will contain potentially non-zero blocks.

$$\begin{bmatrix} A_{11} & \cdots & A_{1k} \\ & \ddots & \vdots \\ & & A_{kk} \end{bmatrix} \tag{19}$$

The blocks below the diagonal will be zero. Figure 2 shows $\mathbf{Y}_{cs}$ for the test system Polish-Winter03, Table I and its block triangular form.

The block elements are reordered using AMD and then factorized whereas the off-diagonal elements are kept as is. This gives the following structure for the KLU factorization of a matrix $\mathbf{A}$ [11]

$$\mathbf{PRAQ} = \mathbf{LU} + \mathbf{F} \tag{20}$$

$\mathbf{P}, \mathbf{Q}$ are permutation matrices, $\mathbf{R}$ is a diagonal scaling matrix, $\mathbf{L}, \mathbf{U}$ are the factorization of the diagonal elements and $\mathbf{F}$ represents the entire off-diagonal. KLU use block back substitution to solve a linear system from the factorization in (20). The structure of KLU in the setting of this paper will be treated later.

ILU is an incomplete solver, which will be used to test, how reducing fill-in can speed up the computations and affect the resulting Thévenin voltages. The chosen type of ILU is called the Crout version of ILU (ILUC) [12]. The tolerance for ILU determines when elements are set to 0. Elements will be set to 0, if their value is smaller than the tolerance multiplied by the norm of the column and the tolerance multiplied by the norm of the row. The tolerance in this study is chosen as 10^{-5}. The choice of tolerance is explained in detail in section IV. In addition ILU is set to preserve row sums as it has been found to significantly increase the accuracy of the results without affecting the runtime of the algorithm.

A. KLU of an admittance matrix

The structure of KLU, (20), does not fit in to the setting of Algorithm 1. $\mathbf{Y}_{cs}$ should be split into an $\mathbf{L}$ and $\mathbf{U}$ part, while KLU provides a factorization of the form $\mathbf{LU} + \mathbf{F}$. However, this will not be an issue, since for an admittance matrix with complex admittances $\mathbf{F} = \mathbf{0}$.

In [18] it is stated that the block triangular form of a square matrix, $\mathbf{A}$ with zero-free diagonal corresponds to finding the strongly connected components of a directed graph $G(\mathbf{A})$. An admittance matrix with complex admittances will always have a zero-free diagonal. Therefore, the block triangular form of $\mathbf{Y}_{cs}$ will correspond to finding the strongly connected components of $G(\mathbf{Y}_{cs}) = (V, E)$ with the nodes $V = \{1, \ldots, |cs|\}$ and the edges $E = \{(i, j) \mid \mathbf{Y}_{cs}(i, j) \neq 0\}$.

A strongly connected component is defined as maximal set of nodes such that for any pair of nodes in the set the paths $i \rightsquigarrow j$ and $j \rightsquigarrow i$ exists. This means that there will be a path both from i to j and from j to i in the directed graph.

The non-zero pattern of an admittance matrix is symmetric. This means, that $\mathbf{Y}_{cs}(i, j) \neq 0 \Leftrightarrow \mathbf{Y}_{cs}(j, i) \neq 0$ and in equality $\mathbf{Y}_{cs}(i, j) = 0 \Leftrightarrow \mathbf{Y}_{cs}(j, i) = 0$. Therefore, there are two scenarios for edges between two nodes i and j in the directed graph $G(\mathbf{Y}_{cs})$. Either there will be no edges between the nodes i and j or there will be an edge both from i to j, (i, j), and from j to i, (j, i), see Figure 3. This means that two nodes will either be in the same strongly connected component or they will be completely separated, since the graph only contains bidirectional edges. Hence the block triangular form of $\mathbf{Y}_{cs}$ will consist of the strongly connected components in the diagonal and the entire off-diagonal will be empty, since there is no connection between the components. Edges in the off-diagonal block would stem from asymmetries in the non-zero pattern, where $\mathbf{Y}_{cs}(i, j) \neq 0$ and $\mathbf{Y}_{cs}(j, i) = 0$. Therefore, KLU factorization of $\mathbf{Y}_{cs}$ will always satisfy $\mathbf{F} = \mathbf{0}$.

In the graph for the entire power system there will be a path between any two nodes. The algorithm however only factorize

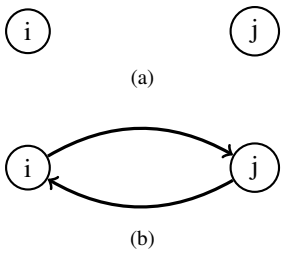


Figure 3. The directed graph $G(\mathbf{Y}_{cs})$ will either have (a) no edges between two nodes i and j or (b) 2 edges (i, j) and (j, i)

$\mathbf{Y}_{cs}$, and the cs nodes need not all be connected, since they can be connected through the vs nodes, which are left out. Figure 4 shows an example of a network divided in to cs and vs nodes. This contain 2 connected components when excluding the vs nodes. Furthermore, it can be noted, that finding the block triangular form will not be an advantage if factorizing the entire admittance matrix. The diagonal will in this case contain only one block element, which is the entire matrix, since all nodes will belong to the same strongly connected component.

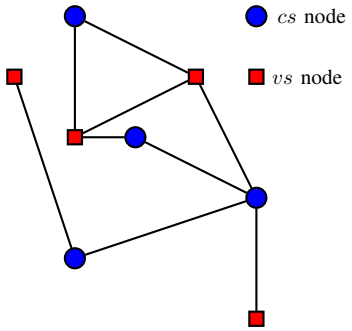


Figure 4. Example of a network divided in to cs and vs nodes

IV. Implementation and test

Algorithm 1 is implemented in MATLAB in order to assess the effect of the different factorization methods. The methods are evaluated with respect to both runtime, accuracy of the results and the number of non-zeros in the resulting coefficients. The runtime is tested on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.50GHz. The test systems can be seen in Table I.

The Pegase and Polish-Winter systems can be found in MATPOWER, [19], the PTI systems are included in the PSS®E 33.0 examples and Nordic32 can be found in [20].

A. The Tolerance of ILU

Before comparing runtimes, fill-ins and precission of the different factorization methods it is necessary to identify a suitable tolerance for ILU. This tolerance determines the sparsity, accuracy and computation time. Errors in Thévenin voltages obtained with ILU can be stated with results from UMFPACK as reference in the terms of a total vector error (TVE) [21]

$$\text{TVE } (\%) = \sqrt{\frac{(\tilde{X}_r - X_r)^2 + (\tilde{X}_i - X_i)^2}{X_r^2 + X_i^2}} \cdot 100\% \qquad (21)$$

TABLE I
TEST SYSTEMS

Case	no. of buses	no. of vs nodes	non-zeros in Y
Nordic32	46	20	160
Pegase89	89	12	501
Pegase1354	1354	260	4774
PTI-WECC-1648	1648	313	6680
Polish-Winter99	2383	327	8155
Polish-Winter03	2746	374	9344
Pegase2869	2869	510	10805
Polish-Winter07	3012	347	10144
PTI-EECC-7991	7917	1325	32211
Pegase9241	9241	1445	37655

where $\tilde{X}$ is the estimate (ILU) and X is the true value (UMFPACK).

Table II shows the sparsity of ILU for different tolerance levels as well as the runtime of Algorithm 1, runtime for computing Thévenin voltages and the TVE of the Thévenin voltages for the test system Pegase9241. Furthermore, the same information for UMFPACK is included in the table.

TABLE II
PERFORMANCE OF ILU DEPENDING ON CHOICE OF TOLERANCE FOR PEGASE9241

Tolerance	Non-zeros of $\mathbf{K}$	Runtime (s) Algorithm 1	Runtime (ms) V_{th}	Max TVE (%) V_{th}
UMFPACK	42515618	11.95	128.08	-
10^{-3}	9126625	2.14	35.69	4540.43
10^{-4}	29047282	7.13	92.22	14.80
10^{-5}	37618552	17.32	119.92	1.89
10^{-6}	42101927	29.48	126.32	0.27
10^{-7}	42423098	43.80	125.80	0.029
10^{-8}	42498272	47.31	129.70	$2.08{\cdot}10^{-3}$
10^{-9}	42506413	50.67	139.04	$2.28{\cdot}10^{-4}$

Increasing the tolerance increases the sparsity of the coefficient matrix while the time for calculating the Thévenin voltages is reduced along with the accuracy. For a tolerance larger than 10^{-6} the sparsity of $\mathbf{K}$ obtained with ILU is approaching that obtained with UMFPACK and the advantage of using ILU disappears.

At a tolerance of 10^{-4} ILU shows runtimes of Algorithm 1, that are comparable to UMFPACK for all systems. For Pegase9241 the runtime is lower, but the errors in Thévenin voltages at this level of tolerance were shown to be up to 15% TVE. An error in Thévenin voltage of a few percent might be accounted for by defining an appropriate trigger-margin for the stability indicators. Thus, the choice of a suitable tolerance for ILU must satisfy the criteria that there should be an advantage of ILU over UMFPACK in terms of runtime, and the resulting inaccuracy should be a few percent at most. On this basis a tolerance of 10^{-5} is chosen.

With the choice of tolerance in place the accuracy of ILU may be assessed for all the test systems. Figure 5 shows the TVE between ILU and UMFPACK of the Thévenin voltages for all test systems. The mean value of TVE is in general small in all the cases, which means that there are few large errors. The largest vector error is smaller than 2% for all test systems.

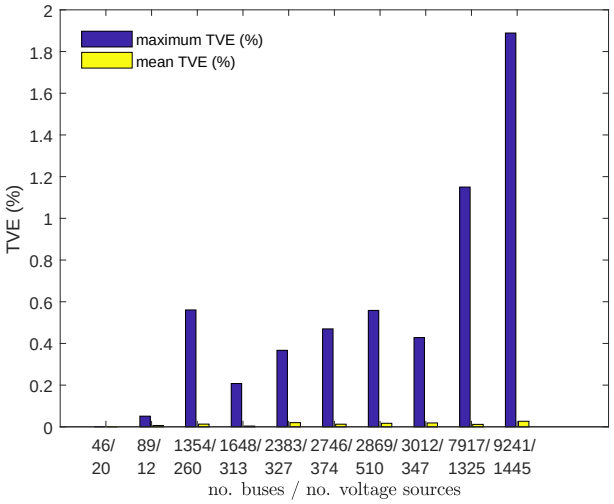


Figure 5. Maximum and mean TVE (%), when comparing Thévenin voltages from ILU to UMFPACK. The tolerance of ILU is set to 10^{-5}.

B. Comparing Factorization Methods

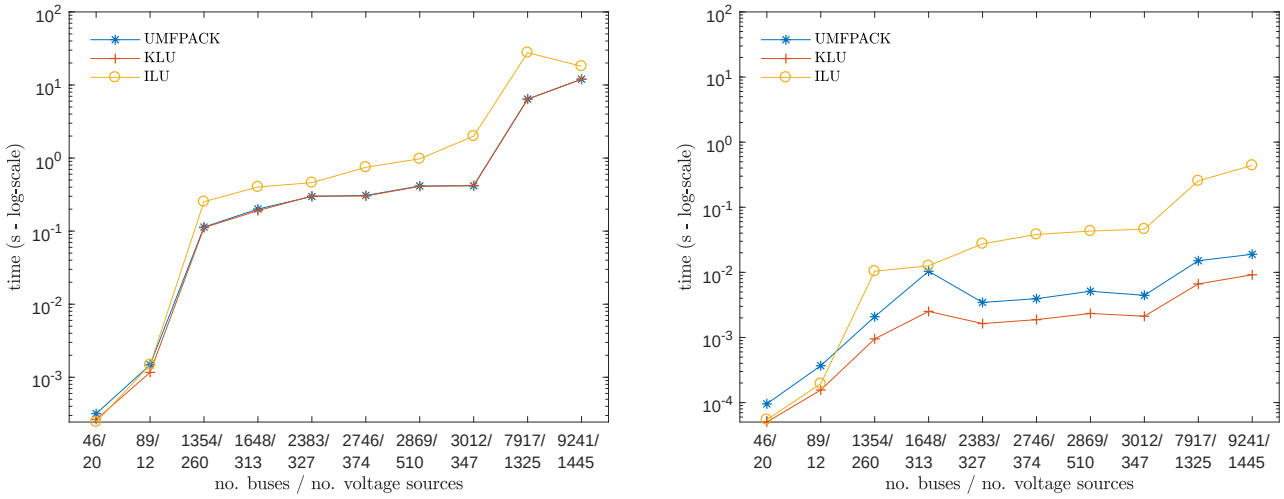
Figure 6a shows the total runtime of Algorithm 1 for each factorization method. The total runtime of UMFPACK and KLU are close with a small favor to KLU. ILU for the chosen tolerance is slower in all but the smallest cases, and the runtime seems to be very dependent on the structure of the network.

Figure 6b shows the runtime for the factorization step of Algorithm 1. KLU consistently performs the factorization almost twice as fast as UMFPACK and is the fastest factorization method in all cases. However, comparing the runtimes reported in Figure 6a and 6b it is evident that the time spent on factorization is negligible compared to the total runtime of Algorithm 1.

Looking at the ability to predict instability issues Table III and IV show that ILU with the chosen settings is close to the results from UMFPACK. The results for KLU show that the accuracy can be considered to be the same as for UMFPACK. In all cases ILU predicted the same node to be the critical node as both UMFPACK and KLU.

For all cases KLU and UMFPACK are close both in computation time and accuracy and in general the largest error of the Thévenin voltages was 10^{-13} when comparing the two. The larger error for ILU is explained by the inaccuracies in Thévenin voltages seen in Figure 5.

One advantage of ILU is the increased sparsity of the coefficient matrices. Table V shows the number of non-zeros of the coefficient matrix for each factorization method. For small systems ILU does not provide additional sparsity however as the systems grows larger it significantly reduces fill-in. The difference with respect to sparsity between KLU and UMFPACK is insignificant.



(a) (b)

Figure 6. For each factorization method versus size of power system (a) shows the total runtime of Algorithm 1 and (b) the runtime of the factorization step.

TABLE III
VOLTAGE STABILITY INDEX FOR LOADS (L-INDEX)

Case	L-index UMFPACK	Difference KLU	Difference ILU
Nordic32	0.183	0	0
Pegase89	0.316	0	$1.25 \cdot 10^{-4}$
Pegase1354	0.212	0	$-2.2 \cdot 10^{-16}$
PTI-WECC-1648	0.283	$-1.7 \cdot 10^{-15}$	$-1.9 \cdot 10^{-4}$
Polish-Winter99	0.066	$-1.2 \cdot 10^{-14}$	$7.39 \cdot 10^{-5}$
Polish-Winter03	0.105	$5.4 \cdot 10^{-15}$	$1.01 \cdot 10^{-4}$
Pegase2869	0.163	$3.9 \cdot 10^{-16}$	$6.04 \cdot 10^{-4}$
Polish-Winter07	0.075	$2.9 \cdot 10^{-15}$	$-6.3 \cdot 10^{-4}$
PTI-EECC-7991	0.311	$6.7 \cdot 10^{-16}$	$-1.1 \cdot 10^{-4}$
Pegase9241	0.176	$-5.3 \cdot 10^{-16}$	$-2.17 \cdot 10^{-3}$

TABLE IV
APERIODIC SMALL SIGNAL ROTOR ANGLE STABILITY MARGIN FOR GENERATORS ($\min \%\Delta P_{inj}$)

Case	$\min \%\Delta P_{inj}$ UMFPACK	Difference KLU	Difference ILU
Nordic32	38.01	0	0
Pegase89	94.33	0	0.004
Pegase1354	81.00	0	0.018
PTI-WECC-1648	32.82	0	0
Polish-Winter99	81.42	$5.0 \cdot 10^{-13}$	0.026
Polish-Winter03	88.79	$4.0 \cdot 10^{-13}$	$9.05 \cdot 10^{-4}$
Pegase2869	63.41	0	0
Polish-Winter07	79.93	$-7.7 \cdot 10^{-13}$	0.009
PTI-EECC-7991	44.54	0	$1.39 \cdot 10^{-4}$
Pegase9241	62.84	0	0

Reduced fill-in can reduce runtime of the Thévenin voltage calculations. In all cases the runtime of computing the Thévenin voltages were smaller for ILU than for the direct methods. If coefficients were to be used for contingency analysis as in [22] the computation time for the contingency assessments could be reduced using ILU. However, the runtime of Algorithm 1 is also increased significantly using ILU.

TABLE V
NON-ZEROS IN COEFFICIENT MATRIX, $\mathbf{K}$, FOR EACH FACTORIZATION METHOD

Case	Non-zeros UMFPACK	Non-zeros KLU	Non-zeros ILU
Nordic32	548	547	549
Pegase89	7668	7663	7664
Pegase1354	1120074	1120078	1025267
PTI-WECC-1648	1706590	1706580	1694777
Polish-Winter99	2825724	2825717	1835089
Polish-Winter03	3027546	3027555	2577363
Pegase2869	2961791	2961793	2790832
Polish-Winter07	4206611	4206582	4095535
PTI-EECC-7991	44852964	44852962	31146092
Pegase9241	42515624	42515659	37618552

V. DISCUSSION OF RESULTS

The use of KLU provides a small speedup of computations compared to UMFPACK at no cost of accuracy in the resulting coefficient matrices. The speedup is achieved through a faster factorization step, but it has been shown that the factorization step is a negligible part of total runtime of computing Thévenin impedances and the coefficient matrix.

The tolerance for ILU was chosen to be 10^{-5} to ensure errors that were smaller than 2% for all test system. It is clear that the error in these cases have a small effect on the resulting stability indicators in Table III and IV. The biggest error might not have occurred on the most critical node or in a direction that did not change the magnitude of the L-index considerably. However, there is no way of guaranteeing, that this will always be the case. The chosen tolerance result in a considerable increase in the runtime of Algorithm 1. For Pegase9241 in Table II the runtime of V_{th} is reduced by 8 ms, while the runtime of the algorithm is increased by 5.4 seconds. This means that for ILU to be an advantage the calculations for

V_{th} should be done more than 675 times before the coefficients needs to be recalculated.

Furthermore, the error in the Thévenin voltages for ILU will have a larger influence near the stability boundary, and here the system topology will also change more rapidly, which will result in recalculations of coefficients to be done more often. Seeing that ILU both introduce an error and give an increased calculation time for coefficients with no major decrease in runtime for Thévenin voltages it severely limits the applicability of ILU.

An alternative method for introducing sparsity could be to compute the full solution and then set elements to zero if they seem to have a small influence on the result. The evaluation of an element's contribution to the result could be done by a norm related tolerance like how ILU sets elements to zero.

Further investigation of calculations of coefficients show that the computational heavy part of the algorithm is determining $\mathbf{Z}_{cs} = \mathbf{Y}_{cs}^{-1}$. Future work with a focus on reducing the runtime of finding the impedance matrix for the cs nodes, would benefit the computations more than a change of factorization method. A sparse implementation of Fox's algorithm [23] was used in [6] to calculate a similar matrix product. Another approach would be to take advantage of backwards solve of KLU. $\mathbf{Z}_{cs}$ could be kept on factorized form, and whenever $\mathbf{Z}_{cs}$ is used in calculations backwards solve would compute the result. In [8] a similar approach is presented, where good performance is obtained, and the method if furthermore optimized by utilizing parallelization.

VI. Conclusion

In this paper different factorization methods for obtaining coefficients for wide-area Thévenin equivalents was evaluated with respect to accuracy, runtime and amount of fill-in.

The results show that the chosen factorization method has little impact on the algorithm for obtaining coefficients, since the factorization step has a short runtime compared to the runtime of the entire algorithm. Looking only at the runtime of the factorization step KLU was the fastest method in all cases. It was furthermore proved, that for an admittance matrix with complex admittances KLU can be used when obtaining coefficients for super-position.

The incomplete factorization method, ILU, was shown to be able to compute stability indicators with an accuracy close to that of UMFPACK and KLU. Using ILU will involve a trade-off between accuracy and sparsity, where sparsity also leads to reduced computation time for both the factorization step and the calculations to determine the Thévenin voltages. For the error of ILU to be in an acceptable range the runtime for computing coefficients is considerably longer than those of UMFPACK and KLU. Furthermore, the increasing influence of errors for a system close to the stability boundary, gives ILU limited applicability.

References

[1] T. Rahman and G. Jasmon, "A new technique for voltage stability analysis in a power system and improved loadflow algorithm for distribution network," in *Proceedings of International Conference on Energy Management and Power Delivery EMPD*, Singapore, 1995, pp. 714–719.

[2] H. Jóhannsson, A. H. Nielsen, and J. Østergaard, "Wide-Area Assessment of Aperiodic Small Signal Rotor Angle Stability in Real-Time," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4545–4557, 2013.

[3] M. Glavic and T. Van Cutsem, "Wide-Area Detection of Voltage Instability From Synchronized Phasor Measurements. Part I: Principle," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1408–1416, 2009.

[4] ——, "Wide-Area Detection of Voltage Instability From Synchronized Phasor Measurements. Part II: Simulation Results," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1417–1425, 2009.

[5] H. Yuan and F. Li, "A comparative study of measurement-based Thevenin equivalents identification methods," in *2014 North American Power Symposium (NAPS)*. IEEE, 2014, pp. 1–6.

[6] J. G. Møller, H. Jóhannsson, and J. Østergaard, "Super-Positioning of Voltage Sources for Fast Assessment of Wide-Area Thévenin Equivalents," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1488–1493, 2017.

[7] A. Perez, J. Moller, H. Johannsson, and J. Ostergaard, "Uncertainty in real-time voltage stability assessment methods based on Thévenin equivalent due to PMU's accuracy," in *Proceedings of the 5th IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conference*, Istanbul, Turkey, 2014, pp. 1–6.

[8] S. Sommer and H. Jóhannsson, "Real-time thevenin impedance computation," in *Proceedings of IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*, Washington, DC, USA, 2013, pp. 1–6.

[9] L. Giraud, A. Haidar, and Y. Saad, "Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D," *Numerical Mathematics-theory Methods and Applications*, vol. 3, no. 3, pp. 276–294, 2010.

[10] T. A. Davis, "Algorithm 832: UMFPACK V4.3 - An unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software*, vol. 30, no. 2, pp. 196–199, 2004.

[11] ——, "Algorithm 907 : KLU , A Direct Sparse Solver for Circuit Simulation Problems," *ACM Transactions on Mathematical Software*, vol. 37, no. 3, pp. 1–17, 2010.

[12] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[13] P. Kessel and H. Glavitsch, "Estimating the Voltage Stability of a Power System," *IEEE Trannsactions on Power Delivery*, vol. 1, no. 3, pp. 346–354, 1986.

[14] T. Weckesser, H. Jóhannsson, and J. Østergaard, "Real-Time Remedial Action Against Aperiodic Small Signal Rotor Angle Instability," *IEEE Transactions on Power Systems*, vol. 31, no. 1, pp. 387–396, 2016.

[15] F. Dorfler and F. Bullo, "Kron Reduction of Graphs With Applications to Electrical Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 150–163, 2013.

[16] E. Dmitrova, H. Jóhannsson, and A. H. Nielsen, "Assessment of the impact that individual voltage source has on a generator's stability," in *Proceedings of the 10th International Power and Energy Conference (IPEC)*, Ho Chi Minh City, Vietnam, 2012, pp. 184–189.

[17] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An Approximate Minimum Degree Ordering Algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[18] T. A. Davis, *Direct Methods For Sparse Linear Systems*, N. J. Higham, Ed. Gainesville, Florida: SIAM, 2006.

[19] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[20] CIGRÉ TF38.02.08, "Long Term Dynamics Phase II, Final Report," Tech. Rep., 1993.

[21] IEEE Standards Association, "C37.118.1-2011 IEEE Standard for Synchrophasor Measurements for Power Systems." Tech. Rep., 2011.

[22] J. G. Møller, H. Jóhannsson, and J. Østergaard, "Thevenin equivalent method for dynamic contingency assessment," in *Proceedings of IEEE Power & Energy Society's General Meeting*, Denver, CO, USA, 2015, p. 5.

[23] G. C. Fox, S. W. Otto, and A. J. G. Hey, "Matrix algorithms on a hypercube I: Matrix multiplication," *Parallel Computing*, vol. 4, no. 1, 1987.

[Pub. B] A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment

A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment

Christina Hildebrandt Lüthje Jørgensen, *Student Member, IEEE,* Jakob Glarbo Møller, *Member, IEEE,* Stefan Sommer and Hjörtur Jóhannsson, *Member, IEEE*

***Abstract*—This paper introduces a factor-solve method, which efficiently computes Thévenin equivalents for all buses in the power system. A range of real-time stability assessment methods rely on Thévenin equivalents, and it is therefore essential that these can be determined fast and efficiently. The factor-solve method has runtime for computing Thévenin voltage that scales linearly with system size resulting in runtime of only a few milliseconds even for systems with several thousand buses. The computations only need the sparse admittance matrix for the power system and a sparse factorization resulting in low memory requirements, and furthermore Thévenin impedances can be determined in parallel. The factor-solve method is compared to a reference method, which uses coefficients for super-position to determine the Thévenin equivalents. The reference method is shown to have dissatisfying runtime and complexity. The factor-solve method is tested, parallelized and analysed, which shows a considerable speed-up in computations of Thévenin equivalents enabling them to be computed in real-time.**

***Index Terms*—Algorithms, Power system analysis computing, Real-time assessment, Thévenin equivalent**

NOMENCLATURE

X	Vector of complex entries
$\mathbf{X}$	Matrix with complex entries
$\|\mathbf{X}\|$	Number of non-zeros in matrix $\mathbf{X}$
$\mathcal{I}$	Identity matrix
$\mathcal{D}(X)$	Diagonalization of the vector X into a diagonal matrix

I. INTRODUCTION

THÉVENIN equivalent computations are used in assessment of power system stability. Steady-state stability of a generator can be determined as a margin of the maximum power injection using Thévenin equivalents [1], [2]. In [3]–[5] methods for voltage stability assessment are introduced which takes advantage of Thévenin equivalents, while [6] derives sensitivities based on a Thévenin equivalent representation to detect transient voltage sags. In [7] the Thévenin equivalent static contingency assessment (TESCA) method is introduced, which uses Thévenin equivalents to solve the power-flow problem and to evaluate aperiodic small signal stability following a contingency.

C. Jørgensen, J. Møller and H. Jóhannsson are with Center for Electric Power and Energy, Department of Electrical Engineering, Technical University of Denmark, Kongens Lyngby, Denmark (e-mail: chhil@elektro.dtu.dk).

S. Sommer is with Department of Computer Science, University of Copenhagen, Copenhagen, Denmark.

The introduction of methods for stability and security assessment as well as methods for defining countermeasures will either result in a competition for the available computational resources or introduce a need for larger computational resources. It is therefore essential, that the methods are first of all fast, but they should also use few computational resources (CPU, memory, etc.). Therefore, Thévenin equivalents should be computed both fast and efficiently. Decomposing the system using a Schur complement has previously been proposed in order to reduce system size and optimize computations.

In [8] the super-postion principle is used to determine the contribution of each voltage or current source to the Thévenin voltage. The method take advantage of a Schur complement, which [9] use to decompose the system for dynamic power system computations. In [5] a Schur complement is used to limit the computational burden when finding Thévenin equivalents for load buses, while [10] and the extended version of this [11] applies a Schur complement to efficiently compute Thévenin impedances for generators.

A range of methods for computing Thévenin equivalents for loads is analysed in [12] in order to assess a voltage stability margin. One method uses a Schur complement, and the complexity is estimated to have a cubic overhead, which is concluded to potentially become a burden for large systems. The Schur complement is generally considered to be dense [10], [13] and therefore it is computationally inefficient to determine. In reality all the coefficients for super-position and not just the Schur complement can be dense [14], which means that computing the coefficients is computational expensive.

In [14] different factorization methods are analysed to optimize the method for finding Thévenin equivalents. It is identified, that the Clark Kent LU factorization (KLU) factorization is the most efficient factorization method, however as in [8] it is determined that the greatest share of the execution time is spend on matrix multiplications due to the density of the matrices involved. The coefficients for super-position and Thévenin impedances are computed, whenever the topology of the power system change, whereas the Thévenin voltage can be determined for every new system state to monitor the system or for every iteration of a steady-state analysis. In [8] the method for determining coefficients for super-position was optimized, however the computation of Thévenin voltages was not addressed.

KLU is a factorization method optimized for sparse systems [15]. The method transform the system to block triangular

form and use approximate minimum degree ordering of the blocks to minimize fill-in before factorizing each block separately. KLU uses block back substitution to solve a system of linear equations. The factorization and the fill-in generated for sparse systems is close to linear with systems size in the context of this paper [10]. KLU is part of the library SuiteSparse [16].

This paper introduces a new method for computing Thévenin equivalents, opposed to the reference method [8], [14]. The method builds on the ideas given in [10] of developing a *factor-solve* method, where the method here will compute the Thévenin equivalents for the entire system instead of only the Thévenin impedances for generators. The method will take advantage of block back substitution in KLU to avoid computing coefficients for super-positon. Thereby, the computationally expensive matrix multiplication of the reference method will be omitted, and additionally the method will be considerably more memory-efficient.

Although [12] estimates a cubic runtime, when using a Schur complement, it will be shown that this is not the case for the methods in this paper. The *factor-solve* method will facilitate a speed-up of the runtime of both the computation of the Thévenin impedances as well as the Thévenin voltages. The method is split in to a sequential and parallel part compared to the sequential reference method, making it suitable for parallelization, which often enables the performance to become considerably better.

Following the introduction Section I.A gives some examples of real-time stability assessment methods using Thévenin equivalents. Section II describes the reference method for computing Thévenin equivalents using a Schur complement and investigate the runtime of this method. A *factor-solve* method for computing Thévenin equivalents is proposed in Section III. The method is implemented and tested in Section IV and lastly parallelized for optimal performance. The scalability of the method is evaluated as well as the resulting runtime and memory requirements of the computations. Section V discusses the results and gives perspectives on the method, while Section VI concludes the paper.

A. Thévenin equivalents in stability and security assessment

A range of methods for stability and security assessment use Thévenin equivalents in their computations. To ensure that these method can run in real-time, the Thévenin equivalent computations should be fast and efficient. Below follows some examples of Thévenin equivalents used in assessment methods.

1) Aperiodic small-signal rotor angle stability: In [1] the maximal injectable power by a generator is determined. This is used to determine a margin to the boundary for aperiodic small-signal rotor angle stability as a percentage margin to the maximal injectable power. This is based on algebraically derived equations [17] and given as

$$\%\Delta P_{inj} = \frac{P_{inj,max} - P_{inj}}{P_{inj,max}} \cdot 100\% \tag{1}$$

$$= \frac{\cos(\delta + \phi_{th}) + 1}{1 + \frac{|V|}{|V_{th}|}\cos\phi_{th}} \cdot 100\%, \tag{2}$$

where the generator is represented as a voltage source V with angle δ and the remaining grid by its Thévenin equivalent with a voltage source V_{th} and an impedance Z_{th} with angle ϕ_{th}. V_{th} is used as the phase angle reference.

A real-time remedial action can be computed as a countermeasure to keep the system from becoming unstable [2]. This is done by computing the reduction in power needed for the critical generator to become N-1 secure. The power is then redispatched to the remaining generators in the system ensuring these also remain secure.

2) Voltage stability: In [5] a voltage stability margin for the loads is determined. This uses the impedance match criterion to determine the maximum deliverable power to the load, which is given by

$$S_{max} = \frac{|V_{th}^2|\left[|Z_{th}| - (\text{imag}(Z_{th})\sin\theta + \text{real}(Z_{th})\cos\theta)\right]}{2\left[\text{imag}(Z_{th})\cos\theta - \text{real}(Z_{th})\sin\theta\right]^2} \tag{3}$$

with θ being the power factor angle of the load. The margin is then determined using the apparent power of the load S_i

$$\%\Delta S_i = \frac{S_{max,i} - S_i}{S_{max,i}} \cdot 100\%, \tag{4}$$

which determines the distance to the boundary of voltage stability.

3) Post-contingency aperiodic small-signal stability: In [7] a method for security assessment is described that determines the aperiodic small-signal stability of the power system following a contingency. The post-contingency steady-state nodal voltages is determined by first computing the Thévenin impedances post-contingency and then computing the Thévenin voltage and nodal voltage angle iteratively until the steady state voltage is found. The voltage angle δ_i is determined at each iteration as

$$\delta_i = \arccos\left(\frac{P_i|Z_{th,i}|}{|V_i||V_{th,i}|} - \frac{R_{th,i}|V_i|}{|Z_{th,i}||V_{th,i}|}\right) \tag{5}$$

The resulting steady-state nodal voltage can then be used to determine the N-1 stability using the earlier mentioned margin for maximum injectable power by generators.

II. Reference method

A. Schur complement and Thévenin equivalents

In [8], [14] a method for computing Thévenin equivalents is described, which uses coefficients for super-position. Thévenin equivalents consist of a Thévenin impedance Z_{th} and Thévenin voltage V_{th}. The Thévenin equivalent seen from node i satisfies

$$V_{th,i} = V_i - Z_{th,i}I_i, \tag{6}$$

where V_i is the node voltage and I_i is the current injected in to the network at node i.

Sources in a circuit can as in power-flow calculations be partitioned in to two sets - current sources (cs) and voltage sources (vs). Floating nodes may be represented as a current source injecting 0 current; loads may be represented as impedances, dependent- or independent current sources; generators with automatic voltage regulator (AVR) may be

represented as voltage sources; internal voltages of manually excited machines may be represented as voltage sources.

The admittance matrix for the system can then be blockwise partitioned as follows

$$\begin{bmatrix} I_{cs} \\ I_{vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{cs} & \mathbf{Y}_{v\to c} \\ \mathbf{Y}_{c\to v} & \mathbf{Y}_{vs} \end{bmatrix} \begin{bmatrix} V_{cs} \\ V_{vs} \end{bmatrix} \tag{7}$$

Eliminating V_{cs} from (7) yields

$$I_{vs} = \mathbf{Y}_{eq} V_{vs} - \mathbf{Q}_{ac} I_{cs} \tag{8}$$

with

$$\mathbf{Y}_{eq} = \mathbf{Y}_{vs} - \mathbf{Y}_{c\to v}\mathbf{Y}_{cs}^{-1}\mathbf{Y}_{v\to c} \tag{9}$$

$$\mathbf{Q}_{ac} = -\mathbf{Y}_{c\to v}\mathbf{Y}_{cs}^{-1} \tag{10}$$

where $\mathbf{Y}_{eq}$ is the Schur complement and $\mathbf{Q}_{ac}$ is the accompanying matrix. This reduction of the network is also known as *Kron reduction* [18].

The Thévenin impedances seen from node i is determined by short circuiting all voltage sources and open-circuiting all current sources, which will be

$$Z_{th,i} = \begin{cases} \mathbf{Z}_{cs}(i,i) & i \in cs \\ \mathbf{Y}_{eq}(i,i)^{-1} & i \in vs \end{cases} \tag{11}$$

where $\mathbf{Z}_{cs} = \mathbf{Y}_{cs}^{-1}$ [8].

Using the definition for Thévenin voltage given in (6) and the above network equations (8)-(10) the Thévenin voltages for the cs and vs nodes respectively are defined as

$$\begin{bmatrix} V_{th,cs} \\ V_{th,vs} \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix} \begin{bmatrix} I_{cs} \\ V_{vs} \end{bmatrix} = \mathbf{K} \begin{bmatrix} I_{cs} \\ V_{vs} \end{bmatrix} \tag{12}$$

with

$$\mathbf{Z}_c = \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs}) \tag{13}$$

$$\mathbf{K}_{v\to c} = -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c} \tag{14}$$

$$\mathbf{Z}_{c\to v} = \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac} \tag{15}$$

$$\mathbf{K}_v = \mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq} \tag{16}$$

Algorithm 1 determines the Thévenin impedances and the coefficient matrix $\mathbf{K}$ needed for computing the Thévenin voltages.

Algorithm 1 Thévenin equivalents

$\mathbf{L}_{cs}, \mathbf{U}_{cs} \leftarrow$ factorization of $\mathbf{Y}_{cs}$
$\mathbf{U}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{L}_{cs}, \mathcal{I}$)
$\mathbf{L}^T_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{U}^T_{cs}, \mathcal{I}$)
$\mathbf{Z}_{cs} \leftarrow \mathbf{L}_{\mathbf{Z}_{cs}}\mathbf{U}_{\mathbf{Z}_{cs}}$
$Z_{th,cs} \leftarrow \mathcal{D}(\mathbf{Z}_{cs})$
$\mathbf{Q}_{ac} \leftarrow -\mathbf{Y}_{c\to v}\mathbf{Z}_{cs}$
$\mathbf{Y}_{eq} \leftarrow \mathbf{Y}_{vs} + \mathbf{Q}_{ac}\mathbf{Y}_{v\to c}$
$Z_{th,vs} \leftarrow \mathcal{D}(\mathbf{Y}_{eq})^{-1}$
$\mathbf{Z}_c \leftarrow \mathbf{Z}_{cs} - \mathcal{D}(Z_{th,cs})$
$\mathbf{K}_{v\to c} \leftarrow -\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}$
$\mathbf{Z}_{c\to v} \leftarrow \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}$
$\mathbf{K}_v \leftarrow \mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq}$
$Z_{th} \leftarrow \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}$
$\mathbf{K} \leftarrow \begin{bmatrix} \mathbf{Z}_c & \mathbf{K}_{v\to c} \\ \mathbf{Z}_{c\to v} & \mathbf{K}_v \end{bmatrix}$
return Z_{th} and $\mathbf{K}$

B. Complexity of reference method

All computations of the reference method are done with sparse matrices and the complexity of the computations will therefore depend on the density of these matrices.

Computing $\mathbf{Z}_{cs}$ will have the complexity $O(|\mathbf{L}_{\mathbf{Z}_{cs}}||cs|)$, which is the most expensive computation in Algorithm 1. By comparison inverting sparse matrices will at maximum have one computation per non-zero element in the matrix for each column in the identity matrix, which gives a complexity of $O(|\mathbf{L}_{cs}||cs|) = O(|\mathbf{U}_{cs}||cs|)$ for solve($\mathbf{L}_{cs}, \mathcal{I}$) and solve($\mathbf{U}^T_{cs}, \mathcal{I}$). Empirically $|\mathbf{U}_{cs}| \leq |\mathbf{L}_{\mathbf{Z}_{cs}}|$, and therefore inversion will be computationally cheaper than computing $\mathbf{Z}_{cs}$. The remaining computations of Algorithm 1 will also be of lower complexity, since they involve at least one sparse matrix, and as stated earlier the factorization is close to linear in complexity. Hence the complexity of the algorithm will be $O(|\mathbf{L}_{\mathbf{Z}_{cs}}||cs|)$.

Assuming that $|\mathbf{L}_{\mathbf{Z}_{cs}}|$ ($\simeq |\mathbf{U}_{\mathbf{Z}_{cs}}|$) scale close to linear with system size like it's the case in practise with the number of non-zeros and fill-in generated in the factorization in this context [10] the complexity will be $O(|cs|^2)$.

Determining Thévenin voltages (12) is $O(|\mathbf{K}|)$, and therefore the computation of Thévenin voltages will depend on the density of the coefficient matrix.

C. Test of reference method

The reference method for determining Thévenin equivalents using Algorithm 1 and equation (12) is analysed in MATLAB on a CPU of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. In the implementation KLU is used as the factorization method, due to its efficiency for sparse systems.

The test systems are given in Table I. The Pegase and Polish-Winter systems can be found in MATPOWER [19], the PTI systems are included in the PSS®E 33.0 examples and Nordic32 can be found in [20]. EECC-PSSE-33-0 is a representation of the American Eastern interconnection.

TABLE I
TEST SYSTEMS

Case	no. of buses	no. of vs nodes	non-zeros in $\mathbf{Y}$
Nordic32	46	20	160
Pegase89	89	12	501
Pegase1354	1354	260	4774
PTI-WECC-1648	1648	313	6680
Polish-Winter99	2383	327	8155
Polish-Winter03	2746	374	9344
Pegase2869	2869	510	10805
Polish-Winter07	3012	347	10144
PTI-EECC-7991	7917	1325	32211
Pegase9241	9241	1445	37655
Pegase13659	13659	4092	50909
EECC-PSSE-33-0	29827	3780	107527

Table II shows the density of the coefficient matrix for each test system. Here the density is given as the number of non-zeros and as the percentage of non-zeros to the maximum size of the matrix. Table III shows the resulting runtime.

TABLE II
DENSITY OF $\mathbf{K}$ FOR TEST SYSTEMS

Case	non-zeros in $\mathbf{K}$	density of $\mathbf{K}$ (%)
Nordic32	547	25.9
Pegase89	7663	96.7
Pegase1354	1120078	61.1
PTI-WECC-1648	1706580	62.8
Polish-Winter99	2825717	49.8
Polish-Winter03	3027555	40.2
Pegase2869	2961793	36.0
Polish-Winter07	4206582	46.4
PTI-EECC-7991	44852962	71.6
Pegase9241	42515659	49.8
Pegase13659	184786127	99.0
EECC-PSSE-33-0	854782395	96.1

TABLE III
RUNTIME OF ALGORITHM 1 AND OF COMPUTING THÉVENIN VOLTAGES

Case	Runtime (s) Algorithm 1	Runtime (ms) V_{th}
Nordic32	$2.98{\cdot}10^{-4}$	$1.27{\cdot}10^{-2}$
Pegase89	$1.34{\cdot}10^{-3}$	$3.13{\cdot}10^{-2}$
Pegase1354	0.11	2.74
PTI-WECC-1648	0.20	4.62
Polish-Winter99	0.30	7.08
Polish-Winter03	0.31	8.50
Pegase2869	0.41	7.65
Polish-Winter07	0.42	10.51
PTI-EECC-7991	6.31	118.13
Pegase9241	11.58	118.95
Pegase13659	36.70	530.13
EECC-PSSE-33-0	253.08	2519.75

For small systems the method is viable and the algorithm has a runtime of a few milliseconds, however as the system size and complexity grows, so does the runtime. Figure 1 shows the runtime of Algorithm 1 and the runtime for computing Thévenin voltages V_{th} (12) plotted against system size.

As expected, the figure shows complexity that is close to quadratic to system size for Algorithm 1. The complexity is dependent on $|\mathbf{L}_{\mathbf{Z}_{cs}}|$, which was assumed to scale close to linear with $|cs|$. $|cs|$ scales close to linear with system size as seen in Table I. This results in an almost quadratic complexity.

The complexity of computing Thévenin voltages was analysed to be $O(|\mathbf{K}|)$. An increased system size result in larger matrices and thereby also a possibility of a larger number of non-zeros, therefore the close to quadractic tendency for these computations is reasonable.

PTI-EECC-7991 and Pegase9241 has almost the same number of non-zeros in the coefficient matrix $|\mathbf{K}|$ even though the system size for Pegase9241 is considerably larger. As expected this gives nearly identical runtimes for computing V_{th}. Furthermore, the number of non-zeroes $|\mathbf{K}|$ for EECC-PSSE-33-0 is almost 5 times larger than for Pegase13659 resulting in a runtime for V_{th}, which is also 5 times larger. This is consistent with the analysed complexity.

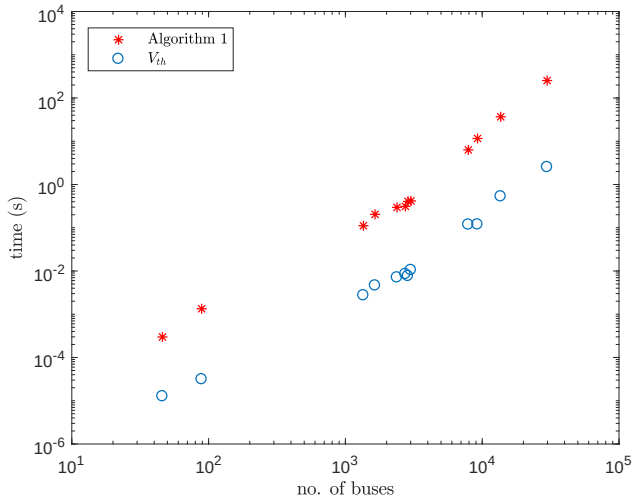


Fig. 1. Runtime for the initial method depending on the number of buses. The runtime is shown for Algorithm 1 and the Thévenin voltages (12) and the plot is logarithmic.

III. INTRODUCTION OF FACTOR-SOLVE METHOD

Performance of Algorithm 1 and the computation of Thévenin voltages is dissatisfying and doesn't scale well with system size, but it turns out that there is a potential for improvements. KLU solves a system by using block back substitution. By use of this solver it is possible to find the Thévenin voltages without computing the coefficient matrix.
The equations from (12) can be written as

$$V_{th,cs} = \left(\mathbf{Z}_{cs} - \mathcal{D}\left(Z_{th,cs}\right)\right) I_{cs} - \mathbf{Z}_{cs}\mathbf{Y}_{v\to c}V_{vs} \tag{17}$$

$$V_{th,vs} = \mathcal{D}(Z_{th,vs})\mathbf{Q}_{ac}I_{cs} + \left(\mathcal{I} - \mathcal{D}(Z_{th,vs})\mathbf{Y}_{eq}\right)V_{vs} \tag{18}$$

Defining $\tilde{V}$ as

$$\tilde{V} = \mathbf{Z}_{cs}\left(I_{cs} - \mathbf{Y}_{v\to c}V_{vs}\right) \tag{19}$$

and inserting this in to (17) and (18) gives

$$V_{th,cs} = \tilde{V} - Z_{th,cs}I_{cs} \tag{20}$$

$$V_{th,vs} = V_{vs} - Z_{th,vs}\left(\mathbf{Y}_{vs}V_{vs} + \mathbf{Y}_{c\to v}\tilde{V}\right) \tag{21}$$

Solving $\mathbf{Y}_{cs}x = b$ for x i.e. finding $x = \mathbf{Z}_{cs}b$ can be determined by block back substitution using the factors LU from the KLU factorization. This will be defined as klu(LU, b) and $\tilde{V}$ can then be calculated by klu$(LU, I_{cs} - \mathbf{Y}_{v\to c}V_{vs})$. It is hereby possible to determine the Thévenin voltages using only the factorization of $\mathbf{Y}_{cs}$ and the Thévenin impedances. This means, that the entire coefficient matrix $\mathbf{K}$ is no longer needed, which will simplify the algorithm.

The Thévenin impedances for cs and vs nodes are defined in (11). The diagonal of $\mathbf{Z}_{cs}$ and the diagonal of the Schur complement $\mathbf{Y}_{eq}$ is needed in these computations.

The diagonal of $\mathbf{Z}_{cs}$ determines the Thévenin impedances for cs nodes and is computed by taking $\mathbf{U}_{\mathbf{Z}_{cs}} = \mathbf{L}_{cs}^{-1}$ and $\mathbf{L}_{\mathbf{Z}_{cs}} = \mathbf{U}_{cs}^{-1}$ and multiplying the rows and columns, that result in the diagonal i.e.

$$Z_{th,cs,k} = \mathbf{L}_{\mathbf{Z}_{cs}}(k,:)\mathbf{U}_{\mathbf{Z}_{cs}}(:,k) \quad \forall k \in cs, \tag{22}$$

where $\mathbf{L}_{\mathbf{Z}_{cs}}(k,:)$ is the k'th row of $\mathbf{L}_{\mathbf{Z}_{cs}}$ and $\mathbf{U}_{\mathbf{Z}_{cs}}(:,k)$ is the k'th column of $\mathbf{U}_{\mathbf{Z}_{cs}}$.

The Thévenin impedances for the vs nodes are given as the inverse of the diagonal elements of the Schur complement $\mathbf{Y}_{eq}$.

$$Z_{th,vs,k} = \mathbf{Y}_{eq}(k,k)^{-1} \quad \forall k \in vs, \tag{23}$$

which is scalar inversion.

Using (9) this can be determined by

$$\mathbf{Y}_{eq}(k,k) = \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\mathbf{Z}_{cs}\mathbf{Y}_{v\to c}(:,k) \tag{24}$$

As with the Thévenin voltages block back substitution is used to determine part of the equation. The Thévenin impedances for the vs nodes are found as

$$\hat{\mathbf{U}}(:,k) = \mathbf{Z}_{cs}\mathbf{Y}_{v\to c}(:,k) \leftarrow \text{klu}\left(LU, \mathbf{Y}_{v\to c}(:,k)\right) \tag{25}$$

$$\mathbf{Y}_{eq}(k,k) = \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\hat{\mathbf{U}}(:,k) \tag{26}$$

$$Z_{th,vs,k} = \mathbf{Y}_{eq}(k,k)^{-1} \tag{27}$$

The computations for the vs nodes is similar to the computations given in [10], and the method will therefore be called a *factor-solve* method. The method factorize part of the system and then solves for part of the equations to efficiently compute the solution. This approach is used to find the Thévenin impedances for the vs nodes and to find the Thévenin voltages for the entire system.

Algorithm 2 factorize $\mathbf{Y}_{cs}$ and compute the Thévenin impedances possibly in parallel.

Algorithm 2 Thévenin equivalents

$\mathbf{L}_{cs}, \mathbf{U}_{cs} \leftarrow$ factorization of $\mathbf{Y}_{cs}$ {Output: LU}
$\mathbf{U}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{L}_{cs}, \mathcal{I}$)
$\mathbf{L}_{\mathbf{Z}_{cs}} \leftarrow$ solve($\mathbf{U}_{cs}^T, \mathcal{I}$)^T
for $k = 1..|cs|$ {In parallel} **do**
  $Z_{th,cs,k} \leftarrow \mathbf{L}_{\mathbf{Z}_{cs}}(k,:)\mathbf{U}_{\mathbf{Z}_{cs}}(:,k)$
end for
for $k = 1..|vs|$ {In parallel} **do**
  $\hat{\mathbf{U}}(:,k) \leftarrow$ klu($LU, \mathbf{Y}_{v\to c}(:,k)$)
  $\mathbf{Y}_{eq}(k,k) \leftarrow \mathbf{Y}_{vs}(k,k) - \mathbf{Y}_{c\to v}(k,:)\hat{\mathbf{U}}(:,k)$
  $Z_{th,vs,k} \leftarrow \mathbf{Y}_{eq}(k,k)^{-1}$
end for
$Z_{th} \leftarrow \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}$
return Z_{th} and LU

This way no computation time is spent on the coefficients and furthermore the computation of the Thévenin voltages is altered from matrix vector multiplication with the dense coefficient matrix to a block back substitution and matrix vector multiplications with sparse matrices.

A. Complexity of factor-solve method

The complexity of Algorithm 2 is split in to a sequential and a parallel part. The first part is sequential and will be $O(|cs|^2)$, since the inversions are $O(|\mathbf{L}_{cs}||cs|)$ and $O(|\mathbf{U}_{cs}||cs|)$ respectively and the fill-in scales linearly with system size. The factorization has negligible runtime compared to this due to the close to linear complexity.

The two loops run sequential will be $O(|cs|^2)$ and $O(|cs||vs|)$, due to the linear complexity of the computations in the loops. When run in parallel the runtime is theoretically determined by the number of cores. By Amdahl's law the total runtime will only be limited by the sequential part of the algorithm, since an unlimited number of cores can be added to completely parallelize the loops. In practise however there will be an overhead due to communication between the cores.

For the computation of the Thévenin voltages the complexity will be $O(|cs|)$. The matrix-vector multiplication is linear due to the sparsity of the matrices scaling with system size, and the block back substitution of KLU is close to linear. This complexity is lower than the reference method's complexity of $O(|\mathbf{K}|)$, which is almost quadratic to system size for the largest systems.

IV. Implementation and test of factor-solve method

The *factor-solve* method is implemented in MATLAB to evaluate the method with respect to runtime of both Algorithm 2 and the computation of Thévenin voltages as well as the accuracy of the results and the memory requirements. The method is tested on Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. The test systems were given earlier in Table I.

The resulting runtime for Algorithm 2 and the runtime for computing the Thévenin voltages (19)-(21) is shown in Table IV, while Fig. 2 shows a plot of the runtime.

TABLE IV
Runtime and speed-up and error for factor-solve method

Case	Runtime (s) Algorithm 2	Speed-up Algorithm 2	Runtime (ms) V_{th}	Speed-up V_{th}
Nordic32	0.045	0.01	0.084	0.15
Pegase89	0.050	0.03	0.101	0.31
Pegase1354	0.114	0.97	0.295	9.28
PTI-WECC-1648	0.544	0.37	0.731	6.32
Polish-Winter99	0.241	1.23	0.457	15.50
Polish-Winter03	0.319	0.98	0.540	15.72
Pegase2869	0.355	1.15	0.604	12.66
Polish-Winter07	0.300	1.39	0.565	18.60
PTI-EECC-7991	1.745	3.62	1.473	80.18
Pegase9241	2.983	3.88	1.708	69.63
Pegase13659	6.276	5.85	2.328	227.75
EECC-PSSE-33-0	20.592	12.29	5.522	456.28

Algorithm 2 is seen to have close to quadratic complexity as analysed earlier, while the Thévenin voltages is seen to have an almost linear complexity as analysed earlier. The change in complexity for the Thévenin voltages result in a significant decrease in runtime compared to the reference method. It is notable that, all test systems have runtimes for computing Thévenin voltages below 6 ms. The system PTI-WECC-1648 has a runtime that is considerably different compared to the other systems. The complexity is only close to linear and actually depends on the fill-in in the factorization, which can differ depending on the structure of the system.

The speed-up is calculated as a quantity for the performance of the *factor-solve* method compared to the reference method. This is defined as $\frac{t_1}{t_2}$, where t_1 is the runtime of the reference

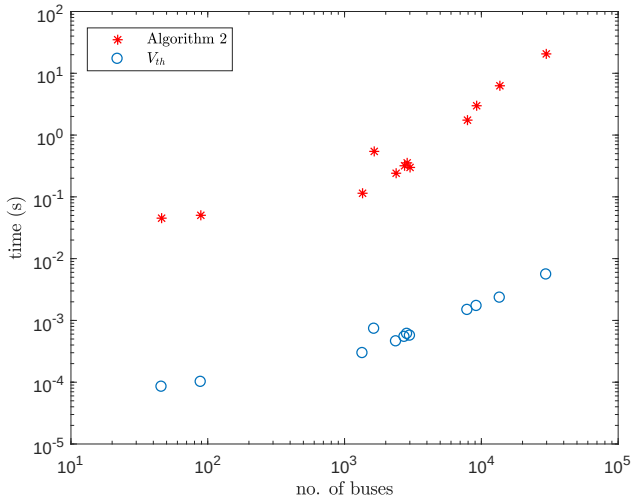


Fig. 2. Runtime for the *factor-solve* method depending on the number of buses. The runtime is shown for Algorithm 2 and the Thévenin voltages (12) and the plot is logarithmic.

method and t_2 is the runtime of the *factor-solve* method. The speed-up is shown alongside the runtimes in Table IV.

For the smaller systems neither the algorithm nor the calculations of Thévenin voltage receives a speed-up. Systems larger than 1000 buses is sped up in Thévenin voltage computations but only some benefit from Algorithm 2. Algorithm 2 requires large power systems to be faster than Algorithm 1.

Some systems benefit more from computing Thévenin voltages with the *factor-solve* method instead of the reference method. The systems Pegase2869 and Pegase9241 both have a lower speed-up than test systems of similar size. The reason for this is found in Table II. The coefficient matrix for both Pegase2869 and Pegase9241 is less dense than for the systems with similar size, and therefore weren't as slow with the reference method. The complexity has changed from being dependent on the number of non-zeros in the coefficients to be close to linearly dependent on system size, which now gives runtimes, that scale better with system size.

Errors in Thévenin voltages obtained with the two methods can be stated in terms of a total vector error (TVE) [21] using the reference method with the standard LU factorization in MATLAB (UMFPACK) as reference

$$\text{TVE } (\%) = \sqrt{\frac{(\tilde{X}_r - X_r)^2 + (\tilde{X}_i - X_i)^2}{X_r^2 + X_i^2}} \cdot 100\%, \quad (28)$$

where $\tilde{X}$ is the estimate (reference or *factor-solve* method) and X is the true value (reference method with UMFPACK).

The maximum TVE can be seen in Table V. The two methods only differs by a small margin in accuracy of the resulting Thévenin voltages.

A benefit from the *factor-solve* method is the amount of memory needed. The reference method need to store the Thévenin impedances along with the coefficient matrix $\mathbf{K}$, while the *factor-solve* method need to store the Thévenin impedances and the sparse factorization of $\mathbf{Y}_{cs}$. Especially for the larger systems there is a significant reduction in memory using the *factor-solve* method, since the coefficient matrix is

TABLE V
MAXIMUM TVE (%) FOR THE REFERENCE AND FACTOR-SOLVE METHOD

Case	Max TVE (%) (reference)	Max TVE (%) (*factor-solve*)
Nordic32	$2.43{\cdot}10^{-13}$	$2.23{\cdot}10^{-13}$
Pegase89	$1.71{\cdot}10^{-12}$	$1.71{\cdot}10^{-12}$
Pegase1354	$4.55{\cdot}10^{-12}$	$4.56{\cdot}10^{-12}$
PTI-WECC-1648	$6.98{\cdot}10^{-12}$	$7.02{\cdot}10^{-12}$
Polish-Winter99	$1.95{\cdot}10^{-11}$	$2.79{\cdot}10^{-11}$
Polish-Winter03	$2.98{\cdot}10^{-11}$	$2.98{\cdot}10^{-11}$
Pegase2869	$5.50{\cdot}10^{-12}$	$5.54{\cdot}10^{-12}$
Polish-Winter07	$1.52{\cdot}10^{-11}$	$1.52{\cdot}10^{-11}$
PTI-EECC-7991	$9.26{\cdot}10^{-12}$	$9.17{\cdot}10^{-12}$
Pegase9241	$2.13{\cdot}10^{-11}$	$2.13{\cdot}10^{-11}$
Pegase13659	$2.20{\cdot}10^{-11}$	$2.22{\cdot}10^{-11}$
EECC-PSSE-33-0	$1.30{\cdot}10^{-09}$	$8.01{\cdot}10^{-10}$

dense. Sparse matrices store the location and value of a non-zero entry, while full matrices store all entries of a matrix. The test systems with a coefficient matrix with a high density, see Table II, will therefore be more efficiently stored as a full matrix than a sparse.

The KLU factorization of a matrix $\mathbf{A}$ is defined as

$$\mathbf{PRAQ} = \mathbf{LU} + \mathbf{F} \quad (29)$$

$\mathbf{P}, \mathbf{Q}$ are permutations stored as vectors, $\mathbf{R}$ is a scaling matrix optimally stored as a vector, $\mathbf{L}, \mathbf{U}$ are complex sparse matrices and $\mathbf{F} = 0$ in this context [14].

The memory requirements for storing the the coefficient matrix (either in full or sparse format) compared to storing the sparse factorization can be seen in Table VI. Integers like doubles are stored using 64 bits.

TABLE VI
MEMORY REQUIREMENTS FOR COEFFICIENT MATRIX $\mathbf{K}$ IN SPARSE AND FULL FORMAT AND FOR THE FACTORIZATION OF $\mathbf{Y}_{cs}$

Case	$\mathbf{K}$ (full)	$\mathbf{K}$ (sparse)	Factorization of $\mathbf{Y}_{cs}$
Nordic32	33.1 kB	13.2 kB	2.9 kB
Pegase89	123.8 kB	180.3 kB	17.7 kB
Pegase1354	28.0 MB	25.6 MB	164.3 kB
PTI-WECC-1648	41.4 MB	39.1 MB	233.6 kB
Polish-Winter99	86.6 MB	64.7 MB	298.8 kB
Polish-Winter03	115.1 MB	69.3 MB	349.4 kB
Pegase2869	125.6 MB	67.8 MB	390.5 kB
Polish-Winter07	138.4 MB	96.3 MB	400.4 kB
PTI-EECC-7991	0.9 GB	1.0 GB	1.2 MB
Pegase9241	1.3 GB	1.0 GB	1.4 MB
Pegase13659	2.8 GB	4.1 GB	2.1 MB
EECC-PSSE-33-0	13.3 GB	19.1 GB	5.6 MB

Storing the factorization requires less memory than storing the coefficient matrix for all test systems i.e. the *factor-solve* method requires less than the reference method. As with the runtime for the Thévenin voltages it is also here the largest systems that have the biggest improvement. For the test system EECC-PSSE-33-0, which is optimally stored as a full matrix, the improvement is a factor of over 2400, while the factor is around 220 for Polish-Winter99 and only 4.6 for Nordic32.

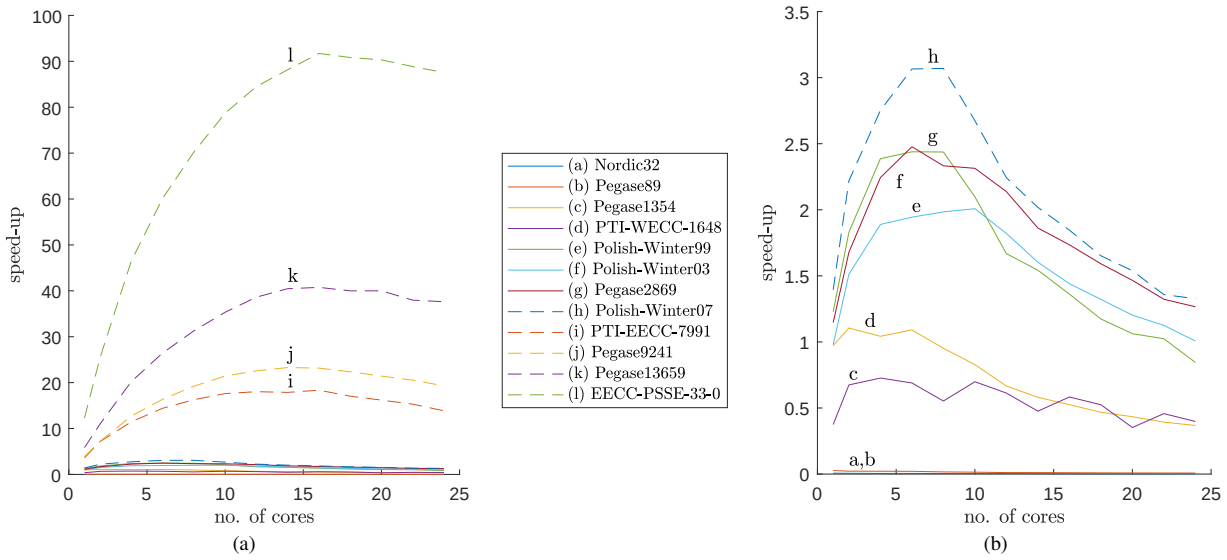


Fig. 3. Speed-up of Algorithm 2 compared to Algorithm 1 for all test systems - (a) shows all test systems and (b) is a zoom of the smaller systems.

This difference is due to the scaling of the memory. The memory for storing $\mathbf{K}$ as a sparse matrix is scaling close to quadratic with system size, while the memory for the sparse factorization of $\mathbf{Y}_{cs}$ is scaling linearly.

A. Parallelization of Algorithm 2

The runtime for computing Thévenin voltages has been decreased significantly, however Algorithm 2 is only considerable faster for the larger systems. A benefit from the *factor-solve* method is that Algorithm 2 can easily be parallelized. The runtime is therefore tested on a machine with 2 CPUs of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz with 12 cores each. The algorithm will be tested on the following number of cores $1, 2, 4, 6, \ldots, 22, 24$.

Fig. 3a shows the speed-up of Algorithm 2 compared to Algorithm 1 for different number of cores for each test system and Fig. 3b shows the same with the 4 largest systems excluded.

Parallelization decrease the runtime of Algorithm 2 considerably. As system size grows the speed-up increases significantly up to a factor of 90 for the largest system compared to a factor 12 without parallelization. The smallest systems still has no speed-up, since these systems already have a short runtime due to their limited size, and the introduction of loops and overhead time only slows them down. There is a range of test systems that when run sequential did not benefit from Algorithm 2, however when parallelized there is now a benefit. Fig. 3a and 3b furthermore show, that the optimal number of cores increase with system size. After the point of maximal speed-up it decreases due to the increased overhead time, which will be larger than the gain of adding additional cores.

The for loops in Algorithm 2 are implemented with the function `parfor` in Matlab for all cores. This will for 1 core i.e. the sequential version be a little slower than using `for`, since there is a small penalty when using `parfor`. The sequential version could therefore be a little faster.

Table VII shows the number of cores that maximize the speed-up for Algorithm 2 and thereby also minimize the runtime. It shows explicitly that the optimal number of cores increase with systems size and so does the speed-up. Table VII show a runtime of 2.8s for EECC-PSSE-33-0 with the parallelized *factor-solve* method compared to 253s for the reference method. This means that the *factor-solve* method will be able to respond faster to a sudden change in the system topology than the reference method.

TABLE VII
RESULTS FOR THE OPTIMAL NUMBER OF CORES FOR EACH TEST SYSTEM

Case	Optimal no. of cores	Runtime (s) Algorithm 2	Speed-up Algorithm 2
Nordic32	1	0.045	0.007
Pegase89	1	0.050	0.027
Pegase1354	2	0.101	1.105
PTI-WECC-1648	4	0.281	0.727
Polish-Winter99	6	0.121	2.439
Polish-Winter03	10	0.155	2.008
Pegase2869	6	0.164	2.477
Polish-Winter07	8	0.136	3.070
PTI-EECC-7991	16	0.344	18.354
Pegase9241	14	0.497	23.289
Pegase13659	16	0.901	40.750
EECC-PSSE-33-0	16	2.759	91.715

Fig. 4 show the distribution of runtime on each part of Algorithm 2 for the optimal number of cores given in Table VII. Factorization time is negligible as expected by its almost linear complexity compared to the quadratic complexity of the entire algorithm as for the reference method [14].

The majority of time is spent on computing the Thévenin impedances for the cs nodes. However, there is also a significantly larger number of cs nodes than vs nodes as seen in Table I. Dividing the runtimes by the number of cs and vs nodes respectively, gives an average runtime for cs nodes that is lower than for the vs nodes in every case. The reason might be found in the difference in the computations. However, it might also be that the optimal number of cores for the

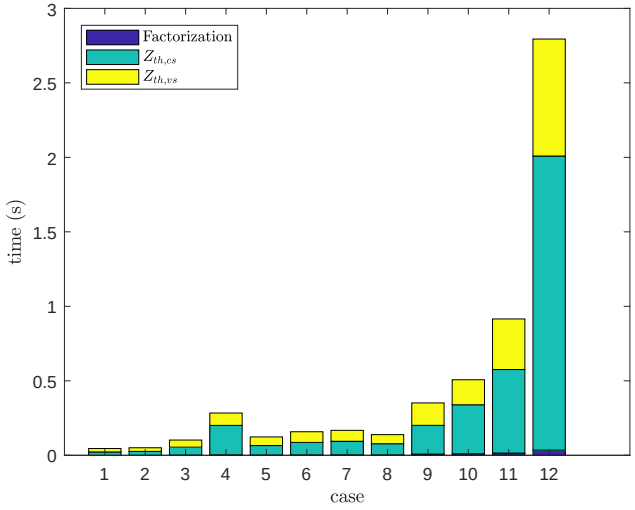


Fig. 4. Distribution of the runtime of Algorithm 2 on to each part of the algorithm for the optimal number of cores.

cs nodes is larger than for the vs nodes. The loops are independent, and therefore running both loops on the same number of cores might result in the computations for the vs nodes being more dominated by overhead. Computations for cs nodes are furthermore split in to a sequential part (inversion of factorization factors) and a parallel part, where the first part will not benefit from additional cores.

V. Discussion and Perspectives

The *factor-solve* method determines Thévenin voltages in linear time given the Thévenin impedances and the factorization of the admittance matrix for the cs nodes. This means that on the given CPU the Thévenin voltages for all buses in the system could be determined in under 6 ms for tests system up to a size of 30.000 buses. If the calculations where to be used in connection with Phasor Measurement Units (PMUs) [22], [23] the method would be able to determine the Thévenin voltages for every measurement, since these will normally be received at the rate of the system frequency (every 16-20ms).

In the *factor-solve* method the runtime of Algorithm 2 dominates the computations, but this is only run, whenever the system topology change. The factorization time is negligible due to linearity, thus the majority of time is spent on determining the Thévenin impedances. Runtime for the *factor-solve* method compared to the reference method is better especially for the larger systems. The complexity is still quadratic as with the reference method, however the implementation is a little faster. More importantly the method can easily be parallelized, since the *factor-solve* method is split in to a sequential and a parallel part compared to the reference method consisting of sequential matrix multiplications.

The fact that the algorithm can be parallelized enables the method to have an even lower runtime. Using only a couple of cores makes the method better than the initial method for systems with 2000 buses or more, while a single core is sufficient to decrease runtime for the larger systems. The system size and the density of the larger systems furthermore increases the gain from using parallelization.

The smallest systems does not benefit from the *factor-solve* method, since these are so small that the time spend on matrix multiplications is neglible and changing these computations in to loops only worsen the runtime due to the overhead.

A few systems benefit from computing Thévenin voltages by use of the *factor-solve* method without benefiting from Algorithm 2 even with parallelization. An alternative method for these system would be to combine the two methods. The Thévenin impedances along with the factorization could be computed by line 1-8 in Algorithm 1 and then the Thévenin voltages would be computed by the *factor-solve* method. These systems will then get the lowest possible runtime, and furthermore only running line 1-8 of Algorithm 1 will also result in a further decrease in the runtime of computing the Thévenin impedances.

This sort of hybrid method would be useable for systems consisting of between 1.000 and 2.000 busses. It should however be noted that the runtime for both methods for these systems is low enough for real-time computations and either one would be suitable. The method can be used for security assessment in for example the Thévenin equivalent static contingency assessment method [7]. Here Thévenin impedances is determined for all N-1 contingencies and Thévenin voltages are then computed several times when determining the steady-state nodal voltage. For contingency analysis it is important to use the fastest combination to ensure that assessment can be done in reasonable time.

An idea for future work would be to investigate the potential use of GPUs instead of doing all the computations on the CPU. Moving data to and from the GPU is expensive, but when the data is there the computations can be executed and parallelized more efficiently. It could potentially be used on the loops in Algorithm 2. However, since there is no reuse of data in the loops it might not give a better performance, since a GPU excel when doing the same computations multiple times. Furthermore, it would also be satisfactory if the sequential part of the algorithm could have it's complexity reduced in order to scale better or be changed to be able to run in parallel.

In the complexity analysis the runtime was determined to be dependent on the sparsity of the factors in the factorization, which is almost linear to the system size. From the results it can be seen that this is the case for most of the test systems, however the structure of some systems give rise to a larger fill-in and will affect the resulting runtime. It would be interesting to find specific reasons for the behaviour of these system.

A clear benefit from the factor-solve method is the decrease in memory usage, where a reduction in required memory was seen for all test systems. This together with the now linear complexity of the Thévenin voltages computations will make the method use fewer computational resources and thereby leave room for other assessment methods.

Future work would also be to include the *factor-solve* method in stability assessment methods to test their performance with the new calculations. It would then be possible to analyse the these methods and determine new areas suitable for optimization. It would furthermore be interesting to test the method in a real-time setting on a SW-platform like [24].

VI. Conclusion

The paper describes a reference method for determining the Thévenin equivalents for all nodes in a power system. The reference method was analysed to be insufficient especially for large power systems. The given complexity of the method is dependent on the density of the coefficients and will therefore be less viable for some systems compared to others due to the structure of the power system.

A *factor-solve* method was introduced, which takes advantage of the block back substitution in KLU. The method spends no computation time on generating coefficients and the calculations of Thévenin voltages computation can be done in linear time. Furthermore, the memory usage is significantly lower than for the reference method changing from gigabytes to a couple of megabytes for larger systems.

Computations of Thévenin impedances can take advantage of parallelization and runtime will therefore be dependent on the number of cores used. The optimal number of cores is shown to increase with system size. The runtime for determining Thévenin impedances is still not satisfying for the largest test system, however these will only be determined, when system topology change. Thévenin voltages are determined without relying on parallization and the linear scaling with system size, enables the *factor-solve* method to have considerably lower computation time than the reference method.

References

[1] H. Jóhannsson, A. H. Nielsen, and J. Østergaard, "Wide-Area Assessment of Aperiodic Small Signal Rotor Angle Stability in Real-Time," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4545–4557, 2013.

[2] T. Weckesser, H. Jóhannsson, and J. Østergaard, "Real-Time Remedial Action Against Aperiodic Small Signal Rotor Angle Instability," *IEEE Transactions on Power Systems*, vol. 31, no. 1, pp. 387–396, 2016.

[3] I. Šmon, G. Verbič, and F. Gubina, "Local voltage-stability index using Tellegen's theorem," *IEEE Transactions on Power Systems*, vol. 21, no. 3, pp. 1267–1275, 2006.

[4] S. Corsi and G. Taranto, "A Real-Time Voltage Instability Identification Algorithm Based on Local Phasor Measurements," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 1271–1279, 2008.

[5] Y. Wang, I. R. Pordanjani, W. Li, W. Xu, T. Chen, E. Vaahedi, and J. Gurney, "Voltage stability monitoring based on the concept of coupled single-port circuit," *IEEE Transactions on Power Systems*, vol. 26, no. 4, pp. 2154–2163, 2011.

[6] T. Weckesser, H. Jóhannsson, J. Østergaard, and T. Van Cutsem, "Sensitivity based assessment of transient voltage sags caused by rotor swings," in *Proceedings of the 18th Power Systems Computation Conference (PSCC)*, Wroclaw, Poland, 2014.

[7] J. G. Møller, H. Jóhannsson, and J. Østergaard, "Thevenin equivalent method for dynamic contingency assessment," in *Proceedings of IEEE Power & Energy Society's General Meeting*, Denver, CO, USA, 2015.

[8] ——, "Super-Positioning of Voltage Sources for Fast Assessment of Wide-Area Thévenin Equivalents," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1488–1493, 2017.

[9] P. Aristidou, D. Fabozzi, and T. Van Cutsem, "Dynamic Simulation of Large-Scale Power Systems Using a Parallel Schur-Complement-Based Decomposition Method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2561–2570, 2014.

[10] S. Sommer and H. Jóhannsson, "Real-time thevenin impedance computation," in *Proceedings of IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*, Washington, DC, USA, 2013, pp. 1–6.

[11] S. Sommer, A. Aabrandt, and H. Jóhannsson, "Reduce-Factor-Solve for Fast Thevenin Impedance Computation and Network," *IET Generation, Transmission & Distribution*, nov 2018.

[12] H. Yuan and F. Li, "A comparative study of measurement-based Thevenin equivalents identification methods," in *2014 North American Power Symposium (NAPS)*. Pullman, WA, USA: IEEE, 2014, pp. 1–6.

[13] L. Giraud, A. Haidar, and Y. Saad, "Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D," *Numerical Mathematics-theory Methods and Applications*, vol. 3, no. 3, pp. 276–294, 2010.

[14] C. Hildebrandt, B. C. Karatas, J. G. Møller, and H. Jóhannsson, "Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment," in *Proceedings of 20th Power Systems Computation Conference (PSCC)*, Dublin, Ireland, 2018.

[15] T. A. Davis, "Algorithm 907 : KLU , A Direct Sparse Solver for Circuit Simulation Problems," *ACM Transactions on Mathematical Software*, vol. 37, no. 3, pp. 1–17, 2010.

[16] ——, *Direct Methods For Sparse Linear Systems*, N. J. Higham, Ed. Gainesville, Florida: SIAM, 2006.

[17] H. Jóhannsson, J. Østergaard, and A. H. Nielsen, "Identification of critical transmission limits in injection impedance plane," *International Journal of Electrical Power & Energy Systems*, vol. 43, no. 1, pp. 433–443, 2012.

[18] F. Dorfler and F. Bullo, "Kron Reduction of Graphs With Applications to Electrical Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 150–163, 2013.

[19] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[20] CIGRÉ TF38.02.08, "Long Term Dynamics Phase II, Final Report," Tech. Rep., 1993.

[21] IEEE Standards Association, "C37.118.1-2011 IEEE Standard for Synchrophasor Measurements for Power Systems." Tech. Rep., 2011.

[22] M. Glavic and T. Van Cutsem, "Wide-Area Detection of Voltage Instability From Synchronized Phasor Measurements. Part I: Principle," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1408–1416, 2009.

[23] ——, "Wide-Area Detection of Voltage Instability From Synchronized Phasor Measurements. Part II: Simulation Results," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1417–1425, 2009.

[24] H. Jóhannsson, H. Morais, A. H. B. Pedersen, Q. Wu, and D. Ouellette, "SW-platform for R&D in Applications of Synchrophasor Measurements for Wide-Area Assessment, Control and Visualization in Real-Time," *CIGRE US National Committee 2014 Grid of the Future Symposium*, 2014.

Christina Hildebrandt Lüthje Jørgensen (S'17) received the M.Sc. degree in mathematical modelling and computation from the Technical University of Denmark in 2015, where she is currently pursuing the Ph.D. degree with the Centre of Electric Power and Energy, Department of Electrical Engineering. Her research interest includes developing high performance algorithms for assessing stability of power systems.

Jakob Glarbo Møller (M'17) received the M.Sc. and Ph.D. degrees in electrical engineering from the Technical University of Denmark in 2013 and 2017 respectively, where he is currently a postdoc with the Centre of Electric Power and Energy, Department of Electrical Engineering. His research interest includes algorithms for assessing operational security of power systems.

Stefan Sommer received his M.Sc. in mathematics in 2008 and his PhD in computer science in 2012 from the University of Copenhagen. He is currently Associate Professor at the Department of Computer Science, University of Copenhagen. His research interests cover aspects of mathematical modelling, numerical algorithms and statistics, including foundational and algorithmic problems in analysis of data with complex structure.

Hjörtur Jóhannsson (M'11) received the M.Sc. and PhD degrees in electrical engineering from the Technical University of Denmark in 2007 and 2011 respectively, where he is currently a Senior Scientific Consultant at the Center of Electric Power and Energy, Department of Electrical Engineering. His research interests concern the development of methods for secure and stable operation of power systems with a high share of RES based production, with special focus on real-time approaches.

[Pub. C] Binary Search and Fit Algorithm for Improved Voltage Stability Boundary Monitoring

Binary Search and Fit Algorithm for Improved Voltage Stability Boundary Monitoring

Christina Hildebrandt Lüthje Jørgensen,
Bahtiyar Can Karatas, Hjörtur Jóhannsson
Department of Electrical Engineering
Technical University of Denmark
Kgs. Lyngby, Denmark

Stefan Sommer
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

***Abstract*—This paper introduces a binary search algorithm using second order polynomial fitting to efficiently determine the maximum power transfer to a non-controlled load when accounting for variations in Thévenin voltage magnitude due to non-linearity. This is used for voltage stability boundary monitoring of a power system in real time. The binary search with polynomial fitting (BSPF) is compared to a reference algorithm, which sweeps over different load levels, and a binary search and is shown to improve both runtime and accuracy of results. The assessment method can take advantage of parallelization, which together with the BSPF algorithm makes it possible to determine a margin for each of the 2.000 non-controlled loads in a 3.000 bus test system in less than 6 seconds. This enables early detection of voltage instability in highly dynamic future smart grid based power systems.**

***Index Terms*—Power system analysis computing, Power system stability, Real-time assessment, Thévenin equivalent**

I. Introduction

Smart grid solutions such as real-time stabillity assessment methods are important in future power systems with a high share of renewable energy sources. A range of real-time stability assessment methods focuses on the use of Thévenin equivalents. In [1] an algebraically derived equation is used to determine a generators distance to the boundary of aperiodic small signal rotor angle stability. This can be computed in milliseconds [2] by fast computations of Thévenin equivalents.

Thévenin equivalents have been suggested in voltage stability assessment [3]–[5], which uses an impedance match criterion, where the maximum power transfer occurs when the load impedance equals the Thévenin impedance. This holds under the assumption that the Thévenin voltage is constant as the load changes. The complexity of such methods is almost linear however, the impedance matching will not be able to detect instability issues in certain scenarios [6]. This is investigated further in [7], which suggest an alternative method that accounts for changes in the Thévenin voltage magnitude arising from non-linearity as the load impedance changes.

High performance algorithms are important to ensure real-time feasiblity and has been a focus of recent research [8], [9]. The Thévenin equivalent for the generators needs to be determined repeatedly [7] making the computational burden for determining the maximum power transfer high. A Schur complement needs to be computed when determining the coefficients [10], which can be computationally inefficient due to the density [2], [11]. It is therefore important to lower the number of computations to make the stability assessment method feasible for real-time computations.

978-1-5386-8218-0/19/$31.00 ©2019 IEEE

This paper contributes with a binary search algorithm to efficiently compute the maximum power transfer to a load accounting for non-linearity of Thévenin voltage and includes fitting to a second order polynomial for better performance. The algorithm is compared to a reference algorithm, which sweeps over different load levels to find the maximum, and is shown to have significantly higher accuracy. Furthermore, the effect of parallelization on runtime will be tested.

Section II describes the Thévenin equivalent computations and the method for computing the maximum power transfer to a load as in [7]. The different algorithms are introduced and described in Section III. Implementation and tests with respect to runtime and accuracy are done in Section IV. Furthermore, some tests with parallelization are carried out. Section V discusses the results, while Section VI concludes the paper.

II. Background

A. Thévenin equivalents

Thévenin equivalents consist of a Thévenin impedance $\overline{Z}_{th}$ and Thévenin voltage $\overline{V}_{th}$. The Thévenin equivalent seen from node i satisfies

$$\overline{V}_{th,i} = \overline{V}_i - \overline{Z}_{th,i}\overline{I}_i, \tag{1}$$

where $\overline{V}_i$ is node voltage and $\overline{I}_i$ is current injected at node i.

The network can be split in to 2 sets of buses - voltage-controlled buses (vc) and non-controlled buses (nc) with no source. The vc buses will be the terminal of a generator with automatic voltage regulators or the internal voltage seen behind the synchronous reactance jX_d, when the machine is manually excited or the over-excitation limiter (OXL) is activated. By representing loads as impedances the current injected at nc buses will be 0. The admittance matrix will then be block-wise partitioned as

$$\begin{bmatrix} 0 \\ I_{vc} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{nc} & \mathbf{Y}_{v\to n} \\ \mathbf{Y}_{n\to v} & \mathbf{Y}_{vc} \end{bmatrix} \begin{bmatrix} V_{nc} \\ V_{vc} \end{bmatrix} \tag{2}$$

Combining (1) and (2) the Thévenin voltage can be determined for the nc buses and vc buses respectively as

$$\overline{V}_{th,nc} = -\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}V_{vc} \tag{3}$$

$$\overline{V}_{th,vc} = \left(\mathcal{I} - \mathcal{D}\left(\overline{Z}_{th,vc}\right)\mathbf{Y}_{eq}\right)V_{vc} \tag{4}$$

where $\mathcal{I}$ is the identity matrix and $\mathcal{D}\left(\overline{Z}_{th,vc}\right)$ is the diagonalization of $\overline{Z}_{th,vc}$ in to a diagonal matrix and $\mathbf{Y}_{eq} = \mathbf{Y}_{vc} - \mathbf{Y}_{n\to v}\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}$ is the Schur complement [12]. $\mathbf{Y}_{nc}^{-1}$ is determined efficiently using an LU factorization [2].

The coefficients will be defined as $\mathbf{K}_{nc} = -\mathbf{Y}_{nc}^{-1}\mathbf{Y}_{v\to n}$ and $\mathbf{K}_{vc} = \mathcal{I} - \mathcal{D}\left(\overline{Z}_{th,vc}\right)\mathbf{Y}_{eq}$.

Fig. 1 show the Thévenin equivalent for an nc load $Z_{LD} = R_{LD} + jX_{LD}$. The equivalent is independent of the load of the node and therefore the admittance of load i is removed from $\mathbf{Y}_{nc}$, when determining row i of $\mathbf{K}_{nc}$. Fig. 2 shows the Thévenin equivalent for a generator.

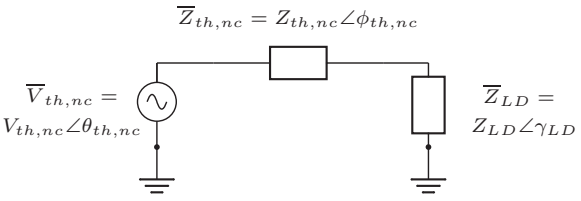


Fig. 1. Thévenin equivalent seen from a load

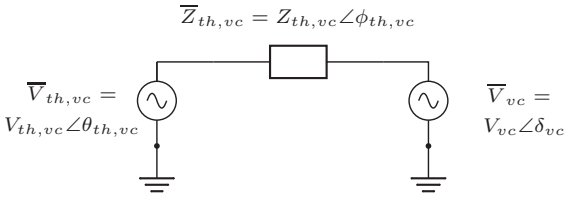


Fig. 2. Thévenin equivalent seen from a generator

The Thévenin impedance is determined as [10]

$$\overline{Z}_{th,i} = \begin{cases} \mathbf{Z}_{nc}(i,i) & i \in nc \\ \mathbf{Y}_{eq}(i,i)^{-1} & i \in vc \end{cases} \tag{5}$$

where $\mathbf{Z}_{nc} = \mathbf{Y}_{nc}^{-1}$. The Thévenin impedance of nc loads is independent of the load of the node and this therefore needs to be accounted for in the computations.

B. Maximum power transfer to a load

The maximum power transfer $P_{LD,max}$ to a given load can be determined more accurately by accounting for changes in the Thévenin voltage magnitude seen from the load [7]. The power transfer to the load P_{LD} for a given change in Z_{LD} can be determined by updating $\mathbf{Y}_{nc}$ with the new load and computing the Thévenin equivalent seen from the generators. The new rotor angle of the generators δ_{vc} can be computed as

$$\delta_{vc} = \arccos\left(\frac{V_{vc}\cos\phi_{th,nc} - P_{inj}Z_{th,vc}}{V_{th,vc}V_{vc}}\right) + \theta_{th,vc} - \phi_{th,vc}, \tag{6}$$

where P_{inj} is the power injected by the generator, which is assumed to be constant as the load change. The remaining quantities can be seen in Fig. 2. $\overline{V}_{th,nc}$ is then recomputed and the power transfer to the load P_{LD} is determined as

$$P_{LD} = \left|\frac{\overline{V}_{th,nc}}{\overline{Z}_{th,nc} + \overline{Z}_{LD}}\right|^2 R_{LD} \tag{7}$$

This is done for different values of Z_{LD} to find the maximum power transfer to the load $P_{LD,max}$. This can be used to determine a margin to the boundary of voltage stability

$$\%\Delta P_{LD} = \frac{P_{LD,max} - P_{LD}}{P_{LD,max}} \cdot 100\% \tag{8}$$

In the computations the coefficients $\mathbf{K}_{nc}$ needs to be computed only once, since each row is independent of the load,while the coefficients $\mathbf{K}_{vc}$ needs to be computed for each change in Z_{LD}. The procedure is shown in Algorithm 1.

To decrease runtime only necessary computations are included. This means that $\mathbf{K}_{nc}$ is only determined for nc loads. Furthermore, the Thévenin equivalent and the new rotor angle is only determined for the generators contributing to a given load. Generators contributing to a load is determined as elements having non-zero entries in $\mathbf{K}_{nc}$ as these are the only elements contributing in the calculations.

Algorithm 1 Find maximum deliverable power to nc loads

```
Compute K_nc and Z_th,nc
Determine contributing generators to each nc load
for each nc load do
  for each change in Z_LD do
    Update Y_nc with new value of Z_LD
    Compute Z̄_th,vc and K_vc for contributing generators
    Compute V̄_th,vc for contributing generators
    Compute δ_vc for contributing generators
    Compute V̄_th,nc for the load using K_nc
    Compute P_LD
  end for
  Determine P_LD,max
end for
```

III. Iteration algorithms

The number of steps needed to determine $P_{LD,max}$ for each load should be as low as possible to improve the runtime of the method. Each step is expensive since $\mathbf{K}_{vc}$ has to be recomputed for every change in Z_{LD}. The maximum power transfer to the load, should therefore be determined fast but also accurately. The following 3 algorithms will be considered

1) Reference algorithm
2) Binary search
3) Binary search with polynomial fitting (BSPF)

A. Reference algorithm

The reference algorithm splits the interval between $Z_{LD,0}$ (the start value) and Z_{th} in to n evenly spaced points and computes P_{LD} for each of these points starting from $Z_{LD,0}$. The remaining steps are skipped if P_{LD} starts to decline or if a contributing generator's rotor angle δ_{vc} becomes imaginary.

P_{LD} will increase with the load up to $P_{LD,max}$ and then start to decline or a generator will get a imaginary rotor angle. It is therefore unnecessary to compute P_{LD} for larger loads as soon as the computed values of P_{LD} start declining. An imaginary rotor angle indicates that the generator will start to lose synchronism [1], since the required amount of power

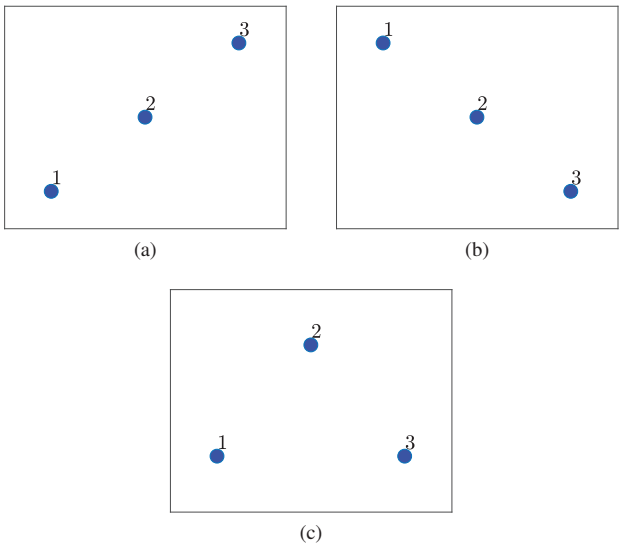


Fig. 3. Possible scenarios for 3 feasible (Y_{LD}, P_{LD}) points: (a) Scenario 1: increasing - (b) Scenario 2: decreasing - (c) Scenario 3: midpoint largest

can not be delivered. This solution is therefore not considered to be a stable operating point for the system and any further increase in load will only worsen the system conditions.

B. Binary search

The reference algorithm have some limitations, since the number of points n needs to be large to get a step size that is small enough to ensure that an accurate $P_{LD,max}$ is found. As an alternative the steps are chosen using a binary search.

The current load $Z_{LD,0}$ with the power transfer $P_{LD,0}$ is used as the first feasible point. The first step $Z_{LD,1}$ is defined as 15% from Z_{th}, based on previous experience [6], [7]. It was seen in experiment to improve runtime compared to using the midpoint between $Z_{LD,0}$ and Z_{th}.

If $Z_{LD,1}$ gives an imaginary rotor angle δ_{vc} for atleast one generator the point is considered infeasible and the interval is restricted using $Z_{LD,1}$ as the new end point $Z_{LD,end}$. If $Z_{LD,1}$ is a feasible point $P_{LD,1}$ is computed, and it is determined if the end point $Z_{LD,end} = Z_{th}$ is feasible.

The search is done iteratively as

1) Compute $Z_{LD,i}$ as midpoint between $Z_{LD,i-1}$ (if this was infeasible use $Z_{LD,i-2}$) and $Z_{LD,end}$
2) If $Z_{LD,i}$ is infeasible $Z_{LD,end} = Z_{LD,i}$ else compute $P_{LD,i}$
3) If 3 feasible points are given sort by the load (see below)

Whenever 3 feasible points are given, they are sorted by the load. Fig. 3 shows the 3 different scenarios, where the power transfer P_{LD} is plotted against the load ($Y_{LD} = 1/Z_{LD}$).

In scenario 1 P_{LD} is increasing with the load. $P_{LD,max}$ will be between point 2 and $Z_{LD,end}$ making point 1 obsolete. If Z_{th} is feasible $Z_{LD,end} = Z_{th}$ will be point 3.

In scenario 2 P_{LD} is decreasing with the load and $Z_{LD,end}$ can be set to point 2. Point 1 will be the current value of the system $P_{LD,0}$, since this is the first feasible point. If the system has passed the boundary of voltage stability $P_{LD,0}$ will be set as $P_{LD,max}$ and the margin to the boundary will be 0.

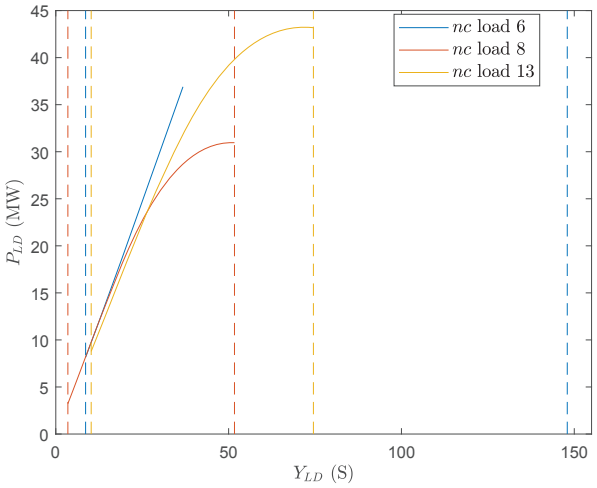


Fig. 4. $(Y_{LD}, P_{LD}) = (1/Z_{LD}, P_{LD})$ curves for 3 different nc loads for the test system Nordic32, Table I. The vertical dashed lines show $Y_{LD,0} = 1/Z_{LD,0}$ and $Y_{th} = 1/Z_{th}$ for each load.

In scenario 3 $P_{LD,max}$ will be between point 1 and 3. Iteratively the midpoint between 1 and 2 and between 2 and 3 is checked. At every iteration the point that is obsolete will be removed so there is always 3 points.

The stopping criteria in all scenarios is the threshold τ. When the relative difference between the 2 relevant points in scenario 1 or 2 is less than τ the search is stopped. It is ensured that both the difference in P_{LD} and Z_{LD} is below τ to avoid scenarios where the difference in P_{LD} is low but the points are on either side of the maximum. For scenario 3 the search is stopped when the relative difference between point 2 and the other points is less than τ.

C. Binary search with polynomial fitting (BSPF)

Fig. 4 shows different shapes of the (Y_{LD}, P_{LD})-curve for 3 nc loads in the Nordic32 test system, Table I. The dashed vertical lines show the interval between $Y_{LD,0} = Z_{LD,0}^{-1}$ and $Y_{th} = Z_{th}^{-1}$.

nc load 8 and 13 have feasible points in the entire interval between $Z_{LD,0}$ and Z_{th}. nc load 8 has $P_{LD,max}$ at Z_{th}, while it occurs just before Z_{th} for nc load 13. The shape of the curve reminds of a second order polynomial, therefore the idea is to fit any 3 feasible points to a second order polynomial and use the maximum of the curve as the next step.

The curve for nc load 6 is almost linear and only have feasible points in part of the interval. This stems from a generator losing synchronism before the nc load reaches it's maximum potential for power transfer. For this load fitting to a second order polynomial would not give a feasible solution and therefore the algorithm is combined with the binary search to be able to still predict the maximum for this type of load.

Whenever 3 feasible points are given these are fitted to a second order polynomial $f(x) = ax^2 + bx + c$ with the maximum $m = -\frac{b}{2a}$. m is infeasible if a is positive i.e. the point is a minimum, if m is outside the given interval or if m is close to $Z_{LD,end}$ and this is an infeasible point. If m is feasible P_{LD} is computed and the now 4 points are reduced

TABLE I
TEST SYSTEMS

Case	no. of buses	no. of nc nodes	no. of nc loads
Nordic32	46	26	18
Pegase89	89	77	35
Pegase1354	1354	1094	673
PTI-WECC-1648	1648	1335	1004
Polish-Winter99	2383	2056	1504
Polish-Winter03	2746	2372	1661
Pegase2869	2869	2359	1491
Polish-Winter07	3012	2665	1939

TABLE II
RUNTIME FOR EACH ALGORITHM

Case	Runtime (s) reference	Runtime (s) binary search	Runtime (s) BSPF
Nordic32	0.113	0.029	0.042
Pegase89	0.577	0.120	0.089
Pegase1354	165.93	36.80	19.26
PTI-WECC-1648	553.10	116.94	57.80
Polish-Winter99	731.55	169.44	97.91
Polish-Winter03	711.06	165.35	95.28
Pegase2869	726.86	164.80	84.88
Polish-Winter07	777.23	181.19	101.56

to 3 points and the fitting is repeated. If m is infeasible the 3 points are reduced to 2 points as in scenario 1 and 2 and a new point is computed like in the binary search.

The search is stopped if the relative difference in P_{LD} is below τ or if m is feasible and close to the previously computed m. To ensure an accurate m a criteria is added that ensures a relative difference in P_{LD} for the feasible points that is less than 50%. The curve is not an exact 2nd order polynomial so this keeps outlying points from skewing the result and atleast one extra point is computed. Furthermore, if the difference in Y_{LD} of the 3 feasible points is below τ, while the difference in P_{LD} is high the fitting is skipped and the binary search is used instead, since this only happens if the fitting does not work like for nc load 6, Fig. 4.

IV. IMPLEMENTATION AND TEST

The algorithms are implemented in MATLAB and evaluated with respect to runtime and accuracy. The runtime is tested on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz.

The test systems are given in Table I. The PTI system is included in the PSS®E 33.0 examples, Nordic32 is found in [13] and the remaining systems are from MATPOWER [14].

The runtimes can be seen in Table II. Line 1 and 2 in Algorithm 1 is the same for all tests and therefore not included in the runtime. The number of points for the reference algorithm is chosen to $n = 50$, since this gives reasonable runtime and $\tau = 10^{-3}$. Furthermore, if the relative difference between $Z_{LD,i}$ and another feasible point is below $\tau_2 = 10^{-4}$ the midpoint between the feasible points is investigated instead.

The runtime is plotted against the number of nc loads in Fig. 5. The decrease in runtime for Polish-Winter07 between

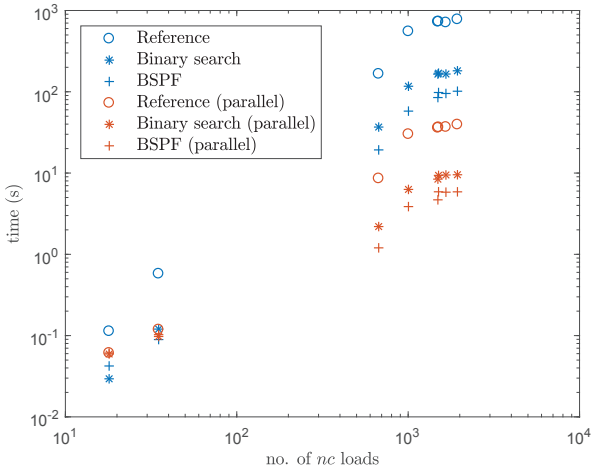


Fig. 5. The runtime for each iteration algorithm (Reference, Binary search and BSPF) depending on the number of nc loads and the runtime, when run in parallel on 24 cores. The plot is logarithmic

TABLE III
$\Delta P\%$ BETWEEN REFERENCE ALGORITHM AND BINARY SEARCH

Case	Maximum difference (%)	Average difference (%)	Minimum difference (%)
Nordic32	16.19	1.56	0
Pegase89	46.38	1.33	0
Pegase1354	$1.19 \cdot 10^4$	66.06	-0.407
PTI-WECC-1648	$5.47 \cdot 10^3$	32.76	0
Polish-Winter99	$2.78 \cdot 10^4$	93.59	0
Polish-Winter03	$1.06 \cdot 10^5$	449.58	0
Pegase2869	$5.08 \cdot 10^9$	$3.42 \cdot 10^6$	-0.405
Polish-Winter07	$2.36 \cdot 10^5$	566.28	0

the reference algorithm and the binary search is more than a factor of 4. BSPF lowers runtime by an extra 40% resulting in a factor of 7.5 in total. The plot shows close to quadratic complexity for all algorithms, giving similar scaling for all.

Another important aspect is the accuracy, therefore the computed values of $P_{LD,max}$ are investigated. Table III shows the percentage difference between the reference algorithm and the binary search computed as ($P_{LD,max,search}$ is largest for $\Delta P\% > 0$ and the opposite for $\Delta P\% < 0$)

$$\Delta P\% = \frac{P_{LD,max,search} - P_{LD,max,ref}}{P_{LD,max,ref}} \cdot 100\% \quad (9)$$

The binary search makes it possible to find the maximum faster and more accurately. On average the $\Delta P\%$ is largely in favour of the binary search and the biggest difference in favour of the reference is less than 0.41%. For a larger n it would be possible to get better results for the reference algorithm, however this would result in higher runtimes. Tests were conducted using $n = 250$, which gave 4 times larger runtimes, however $\Delta P\%$ was still large and some systems even ended up with a larger average difference.

BSPF is compared to the binary search as

$$\Delta P\% = \frac{P_{LD,max,BSPF} - P_{LD,max,search}}{P_{LD,max,search}} \cdot 100\% \quad (10)$$

$P_{LD,max,BSPF}$ is largest for $\Delta P\% > 0$ and the opposite for $\Delta P\% < 0$. $\Delta P\%$ for these two algorithm lies in the interval of $[-0.71, 5.52]\,\%$ with an average difference for all test systems between 0 and 0.01%. Overall we get a better accuracy using BSPF on top of a better runtime. Furthermore, when comparing BSPF to the reference algorithm the minimum difference was at machine accuracy i.e. BSPF was better for all loads.

A. Parallelization test

The voltage stability boundary monitoring is meant to operate in real time, but with current implementations this is not possible. However, runtimes can be further improved by introducing parallelization. The computations for the loads can be distributed on to several cores by parallelizing the other loop i.e. line 3 in Algorithm 1. The number of steps for each load is different and therefore the work will be split dynamically.

The methods are parallelized on a machine with 2 CPUs of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz with 12 cores each i.e. 24 cores in total. The lowest runtime is using 24 cores, however the last couple of cores only give a small improvement. The results form parallelization are shown in Fig. 5. The parallelization of the reference algorithm give runtimes below 40s for all systems, while the binary search have runtimes below 10s and BSPF below 6s. These improved runtimes makes the method feasible also for larger systems.

V. Discussion

The binary search with polynomial fitting is able to determine the maximum power transfer to loads more accurately than the reference algorithm with a lower runtime. Comparing BSPF to using binary search shows that the polynomial fitting improves runtimes considerably. The smallest system has faster runtime using the binary search, whereas the remaining systems are fastest using BSPF. The polynomial fitting is only an improvement for some buses as seen in the examples in Fig. 4. For the buses, where the power transfer is limited by a generator starting to lose synchronism, the polynomial fitting is a waste of time, since the shape is closer to linear.

It is possible to get better accuracy by lowering the threshold if this is needed in computations at the cost of runtime. The application of the reference algorithm is limited due to the high runtime and low accuracy. Future work could include a deeper investigation in to the relationship between the threshold, runtime and resulting accuracy.

The algorithms in this paper lowers the number of steps for each load and therefore future work would be to improve the runtime of each step either by optimizing the computations or completely changing the computations.

The tests with parallelization showed that it is possible to determine a margin for all nc nodes in the system in less than 6s. This is still a considerable amount of time however a margin is also computed for almost 2.000 loads, which would correspond to doing 2.000 continuation power flows. Runtimes below 6s would still make it possible to determine any potential problems before the critical boundary is reached.

All the test systems are far from the stability boundary and it would therefore be interesting to investigate performance as the systems are nearing instability. This can be done in a real-time setting, where the methods are tested on a range of snapshots generated from a time-domain simulation.

VI. Conclusion

This paper describes a binary search to efficiently determine the maximum power transfer to all non-controlled loads in a power system, when accounting for variations in the Thévenin voltage. The binary search includes fitting of feasible points to a second order polynomial to more efficiently determine the point of maximum power transfer. BSPF is shown to be significantly more accurate compared to the reference algorithm, while also having lower runtime.

Parallelization makes it possible to get a margin to the boundary of voltage stability for all 2.000 non-controlled loads in a 3.000 bus system in under 6s. This enables early detection of voltage instability.

References

[1] H. Jóhannsson, A. H. Nielsen, and J. Østergaard, "Wide-Area Assessment of Aperiodic Small Signal Rotor Angle Stability in Real-Time," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4545–4557, 2013.

[2] S. Sommer, A. Aabrandt, and H. Jóhannsson, "Reduce-Factor-Solve for Fast Thevenin Impedance Computation and Network," *IET Gener., Transmiss. Distrib.*, vol. 13, no. 2, pp. 288–295, jan 2019.

[3] I. Šmon, G. Verbič, and F. Gubina, "Local Voltage-Stability Index Using Tellegen's Theorem," *IEEE Trans. Power Syst.*, vol. 21, no. 3, pp. 1267–1275, 2006.

[4] S. Corsi and G. Taranto, "A Real-Time Voltage Instability Identification Algorithm Based on Local Phasor Measurements," *IEEE Trans. Power Syst.*, vol. 23, no. 3, pp. 1271–1279, aug 2008.

[5] M. Glavic and T. Van Cutsem, "A short survey of methods for voltage instability detection," in *Proc. IEEE Power and Energy Soc. Gen. Meeting*, jul 2011.

[6] A. Perez, H. Jóhannsson, P. Vancraeyveld, and J. Østergaard, "Suitability of voltage stability study methods for real-time assessment," in *Proc. 4th IEEE PES Innovative Smart Grid Technol. Conf. Europe*, oct 2013.

[7] B. C. Karatas, H. Jóhannsson, and A. H. Nielsen, "Improved Voltage Stability Boundary Monitoring by Accounting for Variations in Thevenin Voltage Magnitude," in *Proc. 8th IEEE PES Innovative Smart Grid Technol. Conf. Europe*, Sarajevo, Bosnia and Herzegovina, 2018.

[8] C. Hildebrandt, B. C. Karatas, J. G. Møller, and H. Jóhannsson, "Evaluation of Factorization Methods for Thévenin Equivalent Computations in Real-Time Stability Assessment," in *Proc. 20th Power Syst. Comput. Conf.*, Dublin, Ireland, 2018.

[9] C. H. L. Jørgensen, J. G. Møller, S. Sommer, and H. Jóhannsson, "A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment," *IEEE Trans. Power Syst.*, 2019.

[10] J. G. Møller, H. Jóhannsson, and J. Østergaard, "Super-Positioning of Voltage Sources for Fast Assessment of Wide-Area Thévenin Equivalents," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1488–1493, 2017.

[11] L. Giraud, A. Haidar, and Y. Saad, "Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D," *Numer. Math.-Theory Methods and Appl.*, vol. 3, no. 3, pp. 276–294, 2010.

[12] F. Dörfler and F. Bullo, "Kron reduction of graphs with applications to electrical networks," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 60, no. 1, pp. 150–163, 2013.

[13] CIGRÉ TF38.02.08, "Long Term Dynamics Phase II, Final Report," Tech. Rep., 1995.

[14] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, 2011.

Department of Electrical Engineering
Center for Electric Power and Energy (CEE)
Technical University of Denmark
Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark

www.elektro.dtu.dk/cee
Tel: (+45) 45 25 35 00
Fax: (+45) 45 88 61 11
E-mail: cee@elektro.dtu.dk