Equivalent Circuit Formulation based Framework for Probabilistic Power System Analysis

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Martin Rupert Wagner

B.S., Elektrotechnik, Vienna University of Technology
M.S., Mikroelektronik, Vienna University of Technology

Carnegie Mellon University
Pittsburgh, PA

December 2019
© Martin R. Wagner 2019
All rights reserved
ABSTRACT

The Equivalent Circuit Formulation (ECF) was found to enable robust Power System analysis by formulating Power System problems in terms of their true state variables and further applying circuit simulation methods for their robust solution. This thesis describes the theoretical background to formulate equivalent circuit problems for three different Power System analyses: AC Power Flow analysis, an optimization algorithm to identify Power Flow feasibility, and an optimization-based linear State Estimation (SE) algorithm.

We further discuss the design and implementation of a prototype ECF based Power System simulator SUGAR (Simulation with Unified Grid Analyses and Renewables) in C++ that is able to solve the aforementioned analyses effectively. Furthermore, we utilize this implementation to build a framework for probabilistic Power System analyses using a Monte Carlo-based algorithm. We propose a continuation method that effectively and robustly solves Monte Carlo samples given a reference solution, which enables probabilistic Power Flow analysis on models up to continental interconnection-sized systems. In addition, we implement variable correlations within and between models, and propose a probabilistic generation control algorithm.

After comparing our linear State Estimation algorithm that incorporates linear models for PMU and RTU measurements to a traditional WLS estimator, we propose a probabilistic approach to State Estimation utilizing this algorithm. We further demonstrate the feasibility of such an approach and present probabilistic SE studies including network uncertainties. Finally, we propose an approach to identify “true”-grid states by a Monte Carlo-based stochastic optimization.
ACKNOWLEDGEMENTS

To my advisor Larry Pileggi, for his leadership, clear vision, and profound curiosity that sparked this research and made this work possible.

To Franz Franchetti for always having an open ear for me. This work would not have been possible without his continued support and mentorship in all aspects of life.

To my committee members Gabriela Hug and Soumya Kar for the fruitful discussions and insightful questions.

To my group members: Amrit, Marko, Aayushya, and David who are outstanding researchers and were always ready to discuss research, offer help, or just philosophize about life.

To all other PhD students who shared this journey with me: especially, Joe, Meric, Dimitrios, and Onur.

To Javad Mohammadi for being one of the kindest souls on campus.

To Aayushya, Amrit, and the Lunch Bunch for keeping me fit, but also grounded in my basketball aspirations.

To Carolyn Dupont for proof-reading this thesis without spilling coffee on it.

To Julianne Norman for being my better half and always high-spirited life companion.

To my family.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under award no. FA8750-17-1-0059 for the RADICS program, and the National Science Foundation (NSF) under contract no. ECCS-1800812.
Table of Contents

ABSTRACT .................................................................................................................................................. III

ACKNOWLEDGEMENTS .............................................................................................................................. IV

LIST OF ACRONYMS ................................................................................................................................. VII

LIST OF FIGURES ....................................................................................................................................... VIII

LIST OF TABLES .......................................................................................................................................... XII

1 INTRODUCTION ..................................................................................................................................... 1

1.1 MOTIVATION ........................................................................................................................................ 1

1.2 POWER SYSTEMS ............................................................................................................................... 4

1.3 POWER SYSTEM ANALYSES ............................................................................................................. 5

1.3.1 Steady-state Modeling .................................................................................................................... 6

1.3.2 Power Flow Analysis ...................................................................................................................... 7

1.3.3 State Estimation Analysis ............................................................................................................ 9

1.4 CIRCUIT SIMULATION METHODS (SPICE) .................................................................................... 12

1.5 PROBABILISTIC ANALYSIS (MONTE CARLO) ............................................................................... 14

1.6 OVERVIEW OF THIS THESIS ........................................................................................................... 16

2 BACKGROUND: EQUIVALENT CIRCUIT FORMULATION .................................................................. 17

2.1 EQUIVALENT CIRCUIT FORMULATION ....................................................................................... 17

2.1.1 Power Flow ................................................................................................................................... 18

2.1.2 Feasibility ...................................................................................................................................... 29

2.1.3 State Estimation ............................................................................................................................ 35

2.1.4 Power System Control Modeling ................................................................................................... 39

2.2 CIRCUIT SIMULATION METHODS .................................................................................................. 40

2.2.1 Modified Nodal Analysis .............................................................................................................. 40

2.2.2 Solving Linear sparse matrices .................................................................................................... 42

2.2.3 Limiting ......................................................................................................................................... 43

2.2.4 Homotopy Methods .................................................................................................................... 44

2.3 PROBABILISTIC ALGORITHMS ........................................................................................................ 48

2.3.1 Analytical Methods ...................................................................................................................... 48

2.3.2 Numerical Methods ....................................................................................................................... 49

3 SUGAR C++ IMPLEMENTATION .......................................................................................................... 52

3.1 SUGAR C++ (AN ECF BASED SOLVER) ......................................................................................... 52
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGC</td>
<td>Automatic Generation Control</td>
</tr>
<tr>
<td>BLAS</td>
<td>Basic Linear Algebra Subprograms</td>
</tr>
<tr>
<td>ECF</td>
<td>Equivalent Circuit Formulation</td>
</tr>
<tr>
<td>EI</td>
<td>Eastern Interconnection</td>
</tr>
<tr>
<td>KCL</td>
<td>Kirchhoff’s Current Law</td>
</tr>
<tr>
<td>KKT</td>
<td>Karush-Kuhn-Tucker</td>
</tr>
<tr>
<td>LCG</td>
<td>Linear Congruential Generator</td>
</tr>
<tr>
<td>LF</td>
<td>Load Flow</td>
</tr>
<tr>
<td>LFSR</td>
<td>Linear Feedback Shift Register</td>
</tr>
<tr>
<td>MC</td>
<td>Monte Carlo</td>
</tr>
<tr>
<td>MMWG</td>
<td>Multi-Regional Modeling Working Group</td>
</tr>
<tr>
<td>MNA</td>
<td>Modified Nodal Analysis</td>
</tr>
<tr>
<td>MRG</td>
<td>Multiple Recursive Generator</td>
</tr>
<tr>
<td>OPF</td>
<td>Optimal Power Flow</td>
</tr>
<tr>
<td>PCG</td>
<td>Permuted Congruential Generator</td>
</tr>
<tr>
<td>PF</td>
<td>Power Flow</td>
</tr>
<tr>
<td>PLF</td>
<td>Probabilistic Load Flow</td>
</tr>
<tr>
<td>PMU</td>
<td>Phasor Measurement Unit</td>
</tr>
<tr>
<td>PPF</td>
<td>Probabilistic Power Flow</td>
</tr>
<tr>
<td>PRNG</td>
<td>Pseudo Random Number Generator</td>
</tr>
<tr>
<td>PS</td>
<td>Power System</td>
</tr>
<tr>
<td>RNG</td>
<td>Random Number Generator</td>
</tr>
<tr>
<td>RTU</td>
<td>Remote Terminal Unit</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control and Data Acquisition</td>
</tr>
<tr>
<td>SE</td>
<td>State Estimation</td>
</tr>
<tr>
<td>SLF</td>
<td>Stochastic Load Flow</td>
</tr>
<tr>
<td>SRS</td>
<td>Simple Random Sampling</td>
</tr>
<tr>
<td>SUGAR</td>
<td>Simulation with Unified Grid Analyses and Renewables</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>WLS</td>
<td>Weighted Least Squares</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Fig. 1 Structure of a Power System [8]................................................................................................................. 4
Fig. 2 Diagram of positive sequence Power Flow model of a 9-bus system [12]......................................................... 7
Fig. 3 PV plant power generation measured with RTUs [20].................................................................................... 10
Fig. 4 PV plant power generation measured with PMUs [20].................................................................................... 11
Fig. 5 Diagram of linear (green), nonlinear (red), and omitted (blue) system elements within the “PQV” formulation. ........................................................................................................................................... 18
Fig. 6 Diagram of linear (green), and nonlinear (red) system elements within a KCL-based PF formulation......... 19
Fig. 7 Transmission line pi-model with series impedance and shunt elements ......................................................... 20
Fig. 8 Real and imaginary parts of a transmission line ECF model......................................................................... 21
Fig. 9 Diagram of a 2-phase transformer model and its parameters......................................................................... 21
Fig. 10 ECF transformer model including phase shifts and series losses.................................................................. 22
Fig. 11 Shunt from a bus to ground .......................................................................................................................... 23
Fig. 12 Real and imaginary equivalent circuit models of a shunt. ........................................................................ 24
Fig. 13 Linearized real and imaginary ECF models of a constant PQ load............................................................. 25
Fig. 14 Linearized equivalent circuit for one NR iteration of a generator power injection ..................................... 27
Fig. 15 Equivalent Circuit Model of a Slack bus...................................................................................................... 28
Fig. 16 Feasibility sources placed on each bus of the IEEE 14-bus system............................................................. 30
Fig. 17 Adjoint equivalent circuit of a transmission line pi-model. ...................................................................... 33
Fig. 18 adjoint equivalent circuit models of a transformer’s primary and secondary sides............................... 34
Fig. 19 Equivalent Circuit model of a PMU............................................................................................................. 35
Fig. 20 Linear equivalent circuit model of an RTU................................................................................................ 36
Fig. 21 Adjoint RTU equivalent circuit model.......................................................................................................... 38
Fig. 22 Adjoint PMU equivalent circuit model with \( \kappa = 2\text{PMU} \)2, additionally indicating how primary element sources are neutralized in the PMU’s adjoint circuits models ................................................................................................................. 39
Fig. 23 Convergence improvement by applying limiting methods to improve convergence on an 2838 bus system [62]. ........................................................................................................................................... 43
Fig. 24 Five bus example to illustrate the Tx-stepping method............................................................................ 45
Fig. 25 Transmission line series element changes as a function of Tx-stepping method system relaxation factor \( \lambda \). 45
Fig. 26 Adapted transformer model incorporating Tx-stepping adjustments .......................................................... 46
Fig. 27 Auxiliary transmission line that is added during Tx-stepping for a generator (bus 1) remotely controlling the voltage magnitude at bus 4........................................................................................................... 46
Fig. 28 Concepts hierarchy of an ECF based solver. ............................................................... 53
Fig. 29 Global design overview of SUGAR C++ solver. .......................................................... 55
Fig. 30 The two SparseMatrix classes share a common interface within SUGAR C++. .......... 56
Fig. 31 Sparse matrix conversion during Newton-Raphson loop. ........................................... 57
Fig. 32 Activity diagram of the NRSolver’s inner loop implementation .................................. 58
Fig. 33 UML class diagram of the SugarSolver ........................................................................ 60
Fig. 34 Activity diagram of the outer control loop implemented by the SugarSolver. ............. 61
Fig. 35 Class diagram of the ContinuationMethod control interface. ....................................... 63
Fig. 36 Core algorithm of the adaptive Tx-stepping implementation. ....................................... 64
Fig. 37 UML class diagram of the QLimiting class that implements the ContinuationMethod interface. ................................................................. 66
Fig. 38 Control algorithms implemented by QLimiting. .......................................................... 67
Fig. 39 UML class diagram of the DeviceControl class. .......................................................... 69
Fig. 40 Relationships between model data structures of the SUGAR C++ implementation. .... 70
Fig. 41 UML diagram of the Parser ....................................................................................... 70
Fig. 42 Algorithm of the SUGAR C++ parser that applies a factory pattern to instantiate the correct algorithm. ................................................................. 71
Fig. 43 Class diagram of the .raw file parser. ........................................................................... 72
Fig. 44 UML class diagram of the PSModes class .................................................................. 73
Fig. 45 UML diagram of PF, feasibility PF, and SE models with their common interface Model::Base. ................................................................. 75
Fig. 46 UML class diagram of a Plant model. .......................................................................... 76
Fig. 47 UML class diagram of the SugarAlgorithms class. ....................................................... 79
Fig. 48 Flow diagram of the runPowerFlow() function of the SugarAlgorithms class. ........... 80
Fig. 49 UML diagram of structures and classes that represent the SystemGraph library implementation ...................................................................................... 81
Fig. 50 UML class diagram of input parameter data structures. .............................................. 83
Fig. 51 Per iteration solution times of the SUGAR C++ implementation for openly available Power Flow cases using KLU. ................................................................. 86
Fig. 52 Initial Power Flow matrix creation time of selected power system cases. This is a one-time cost. ................................................................. 86
Fig. 53 Performance of the SUGAR C++ solver for a real Eastern Interconnection case and good initial conditions. 87
Fig. 54 SUGAR C++ profile of a real eastern interconnect PF simulation using Tx-stepping. ................. 88
Fig. 55 Solution times of synthetic State Estimation samples that are crated from openly available power flow data. ................................................................. 89
Fig. 56 Valgrind massif memory profiler output for the 9241-bus PEGASE case.......................... 89
Fig. 57 A double-logarithmic plot of requirement for different sized systems for total required memory, KLU memory (symbolic and numeric factorizations), and system matrix size. ................................................................. 90
Fig. 58 The Monte Carlo algorithm. ....................................................................................... 92
Fig. 59 UML class diagram of the parallel MonteCarlo solver and related data structures. ........ 94
Fig. 60 UML class diagrams of the Probabilistic model interface and the class interface for defining and sampling a variable’s distribution................................................................. 95
Fig. 61 Example of json-based probabilistic input ................................................................. 95
Fig. 62 Branch model that defines a localized correlation between its parameters “Gser” and “Bser” .............. 97
Fig. 63 Defines wind power generation correlations between generators in a system.............................. 97
Fig. 64 Class diagram of the correlated random number generator .................................................. 99
Fig. 65 Memory scaling properties of the MC framework for three different size system with additional parallelism. ........................................................................................................... 99
Fig. 66 Depiction of the probabilistic sample continuation algorithm for two probabilistic variables \(x_k\) and \(x_j\). .. 102
Fig. 67 Flow diagrams of the probabilistic sample continuation’s ContinuationMethod visitor function implementations. ........................................................................................................... 102
Fig. 68 Comparing probabilistic results of the 13659-bus PEGASE testcase using SUGAR C++ with sample continuation and a Matpower based MC engine .................................................................................. 103
Fig. 69 Slack bus real power injections into the system for the 2 biggest islands. Negative values mean power generation ......................................................................................................................... 106
Fig. 70 Slack bus reactive power contributions to the system for the two biggest islands ......................... 106
Fig. 71 Extreme voltage magnitudes and angles of the EI test case study ............................................. 107
Fig. 72 Tie-line real power flow on line between PJM and AP grid operators ........................................... 107
Fig. 73 Tie-line reactive power flow between PJM and AP grid operators ............................................. 108
Fig. 74 Left: Histogram of required number of Q-limiting step for most samples. Right: Boxplot of Q-limiting steps for all samples ........................................................................................................ 109
Fig. 75 Left: Histogram of sample runtimes for common samples. Right: Boxplot of runtimes of all samples...... 109
Fig. 76 Definition of intra-model load correlations in the SUGAR C++ probabilistic input .................. 111
Fig. 77 Effects of a positive correlation (factor 0.8) between real and reactive powers on all loads ............ 111
Fig. 78 Definition of inter-model correlations ...................................................................................... 112
Fig. 79 The effects of wind power correlations ...................................................................................... 112
Fig. 80 Definition of probabilistic generator control .............................................................................. 114
Fig. 81 Effects of probabilistic re-dispatching on the real slack bus power ............................................. 114
Fig. 82 Effects of probabilistic re-dispatching on the reactive slack bus power .................................... 114
Fig. 83 Extreme voltage magnitudes and angles of probabilistic studies including wind correlations and both wind correlations and the global probabilistic re-dispatching algorithm. Zoomed in versions are shown in yellow boxes. ........................................................................................................... 115
Fig. 84 Selected probabilistic density functions (PDFs) of our SE algorithm for the 1888-bus RTE case, with (blue) and without (green) including network uncertainties, and the true state of the system ................. 121
Fig. 85 Voltage magnitudes of four select buses of a synthetic State Estimation case on an 82k bus system ....... 123
Fig. 86 Selected voltage angles of the 82k bus synthetic State Estimation case. ................................................................. 123

Fig. 87 Real and imaginary voltages of the IEEE 14-bus case identifying “true” grid states. ......................................................... 126
List of Tables

Table 1 Mapping for linear equivalent circuit elements from the power flow circuit to the adjoint domain [51]. ... 32
Table 2 Comparison of Tx-stepping to standard tool for PF cases. ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・�
1 INTRODUCTION

1.1 MOTIVATION

Today’s societies highly depend on a reliable supply of electricity. In almost every private household around the world, electricity is used for lighting, refrigerating food, and communicating with the outside world. Additionally, many households rely on it for cooking, heating, or cooling. All of these applications have improved the standard of living for today’s citizens and are part of their everyday lives. In addition, reliable electrical power is perpetually saving lives by powering modern diagnostic devices, life supporting machinery, and surgery rooms in hospitals. While hospitals and other vitally important facilities are customarily equipped with backup measures in case of a short-term power loss, a long-term loss of electric power in a country is an extremely threatening scenario. Moreover, such a blackout has potentially devastating effects on the economy of the affected region.

For these reasons, guaranteeing reliable operation of a country’s power system is of vital importance to every nation, assigning electrical power engineers a key role in society and with it a great responsibility. A recent example of how integral an operating power system is to a region’s wellbeing is the prolonged Power System breakdown after the devastating 2017 hurricane Maria in Puerto Rico. Another recent example is the recurring blackouts during the political crisis in Venezuela that cost at least 26 lives [1]. The dreadful economic impact of a blackout is visualized by the cost of the 2003 blackout in the east of North America. It lasted two days and affected more than 50 million customers, incurring an estimated cost somewhere between 7 and 10 billion dollars [2].

While it is imperative to prevent and mitigate the effects of these local disturbances, the key role of power engineers expands to fighting a global crisis and one of humanity’s biggest challenges today. In the fight against Global Warming, one of the mayor challenges is to adapt today’s power systems to remain highly reliably and resilient while being mostly powered by renewable energy sources such as Photovoltaics, wind-power, or hydro-power generation.
A major effect of high shares of renewables in a Power System is a system’s increased stochasticity. Importantly, stochasticity has always been a defining feature of a power system. Traditional sources of stochasticity are changes in demand, changing environmental conditions, and random contingency events. Historically, Power System (PS) analyses for both operation and planning have almost exclusively been deterministic. This discrepancy prevents an effective use of resources and results in high security margins. Furthermore, it is one of the limiting factors for the effective future use of high shares of renewable energy generation.

Policy makers and industry people are aware of this challenge. Efforts to tackle it include the US department of Energy funded Pacific Northwest National Laboratory 2005 report “Stochastic Operations and Planning” [3] that highlights several benefits of probabilistic methods, such as:

- improved reliability
- robust decision making
- optimal capital investments
- more economical system operation
- optimized reserves
- better preparation for extreme events outside of the mainstream operational range

More importantly, the report warns that not having a transition to probabilistic methods would: “Increase the risk of system failures”, “more congestion problems in real time”, lead to “over-conservative operational limits” and “underutilization of transmission assets”, which would result in less “economical system operation while addressing unexpected situations”.

More recently, a report from the North American Electric Reliability Council (NERC) on Probabilistic Methods in 2014 [4] aims to “inform industry of the variety of probabilistic methods currently being researched, to encourage further probabilistic research development”. In Europe, the European Network of Transmission System Operators for Electricity (ENTSOE), recognizes probabilistic methods as a key technology in multiple focus areas of their Research, Development & Innovation Roadmap 2017-2026 [5].
In academia this problem is widely recognized as well. An active academic community around probabilistic methods organizes a biannual conference on Probabilistic Methods Applied to Power Systems (PMAPS) and organizes tutorials, such as the tutorial on “Probabilistic methods for power system management: state of the art, challenges and perspectives” at the PowerTech conference 2019 in Milan. Noteworthily, the understanding of the importance of a probabilistic approach has been around for a long time. The first probabilistic Power Flow (PF) formulation was proposed in 1974 [6]. Examples of the abundance of potential Power System problems to be solved probabilistically are shown in [7].

However, except for the design and reliability considerations discussed in [7], probabilistic approaches have seen slow adoption in the Power System industry, partly because the tradeoff between accuracy and computational complexity is difficult to strike for algorithm designers. As we will see, a lack of robustness in prevalent Power System analyses exacerbates this challenge. Importantly, non-technical issues also prevent the rightful success of probabilistic methods. One key challenge here is the more complex interpretation, visualization, and presentation of probabilistic results to decision makers.

The goal of this thesis is to overcome these challenges and enable probabilistic studies for power systems of the size of continental interconnections without simplifying assumptions in the analyses. We aim to achieve that by first establishing robust algorithms that efficiently solve these problems for two key steady-state analyses (Power Flow, and State Estimation). Further, we implement these algorithms with a focus on performance. Finally, we aim to employ a numerical, Monte Carlo based simulation approach to establish a robust toolbox for probabilistic Power System analyses.
1.2 Power Systems

As already established, a power system is a crucial building block of a nation’s infrastructure. It can be categorized into four components: Generation, Transmission, Distribution and Demand. The challenge of operating a power system is to perpetually balance Generation with Demand while considering constraints of the transmission and distribution networks.

Fig. 1 shows this basic structure of a power system. The majority of the electrical energy is produced in centralized generation facilities. These facilities are customarily placed at a distance from population centers. Transmission systems then transport the energy closer to the consumers. Transmission systems are designed to be resilient to single element failures and commonly feature a meshed network of transmission lines to provide a certain degree of redundancy. These networks ordinarily operate in a balanced state, where each phase transports the same amount of power.

Consumers receive electric energy via distribution systems that are connected to a transmission system at a substation. Distribution systems traditionally have radial structure, resulting in less redundancy. Noteworthily, legacy distribution systems are designed for electricity flows in a single direction. Adapting these systems for bi-directional flow is crucial for the continued integration of (distributed) renewable energy generation.

Power Systems are meticulously planned. In order to guarantee continued reliability of a system, the effects of any topology change or added component must be very well understood. Additionally, high investment costs limit system adaptations, rendering Power Systems highly static.
Planning procedures traditionally employ deterministic models that represent extreme scenarios of the current and future grid. For example, extreme scenarios for the Eastern Interconnection of the United States are created by the Multiregional Modelling Working Group and include grid models up to 10 years ahead [9], including peak loading, light loading and shoulder scenarios for Summer, Winter and Spring.

Power System operations is concerned with a much shorter time scale that commonly ranges from days or weeks ahead to the current state of the system. Still, operating a Power System requires substantial planning as well. The major challenge in operational planning is: *Creating generation schedules that match the forecasted demand and are resilient to contingencies.*

### 1.3 Power System Analyses

Analyses to perform planning and operational tasks are commonly concerned with finding an (optimal) steady-state of the system. This is the case because the necessary assumption that transient behavior of a system between steady-states does not cause oscillations or instabilities holds for any stable PS model. To verify this, time dependent grid analyses exist. However, they require detailed control modeling including difficult to obtain data such as generator control parameters.

Some important steady-state optimization algorithms include: Unit Commitment that establishes which generation facilities need to be online at which point in time, and Optimal Power Flow that finds optimal generation schedules for the allocated generators. Both analyses are generally hard to solve optimization problems for real world systems. Hence, operators commonly fall back to simplified algorithms.

During operation, the state of a system can be estimated by an algorithm tool-chain that is part of a commercial Energy Management Systems. This tool-chain includes Topology Estimation, Bad Data Detection and State Estimation. An estimated system state is further used to analyze if the system is capable of sustaining a required set of contingencies. These contingencies are defined based on a grid operators’ policy restrictions and commonly include N-1 rules, meaning that a given grid is required to sustain a failure of any single component in the grid.
Both the planning and the operational toolchain lead to steady-state grid models that describe a system in certain time intervals. The Power System steady-state analysis, commonly referred to as Power Flow (PF) or Load Flow (LF), is utilized to analyze this steady state. In an operational context, this commonly means to analyze if a given state meets the contingency requirements. For planning purposes, system perturbations such as the effects of a planned new generator or transmission line are studied with this analysis.

It follows that a robust and efficient PF solver is a key asset for any power system planner or operator. However, modern commercial PF solvers suffer from a lack of robustness for bigger, real-world systems, for sizeable perturbations of system states, or for numerically poorly conditioned problems [10], [11]. Notably, the system frequency is not part of the PF analysis restricting efficient modelling of generator and other important control responses to non-physical macro-models.

With decreasing stored energy (commonly referred to as “system inertia”) in today’s systems, it is paramount to improve understanding of dynamic system behavior by improved, more detailed models and improved simulation capabilities. Currently, time-dependent system behavior is studied as part of extreme event analysis or to verify transient behaviors of new, crucial grid components. Improved time-dependent studies could include two types of analyses: 1) Dynamic simulations to evaluate system short-time-scale transients, and 2) quasi-steady-state approaches with lower computational requirements that enable efficient studies of comparatively slow effects.

1.3.1 Steady-state Modeling

Steady-state behavior of power systems is simpler to model and to study. Steady-state models are comprised of a network model and macro-models for loads, generators and other grid components (e.g. FACTS devices, etc.). The most common way to describe a PS on the transmission level is the positive sequence PF model. It assumes a balanced operation of the system, enabling the reduction of the multi-phased system to a single-phase model. The bus-branch version of a positive sequence PS model consists of the network graph, defined by vertices (buses) and edges (branches) that represent transmission lines or transformers. This graph,
together with modeled shunts, is considered the network model. In addition, load and generator models are connected to the buses of the system.

Fig. 2 shows a bus-branch positive sequence transmission Power Flow model of a 9-bus system, where loads are shown as arrows, generators by circular symbols, buses by thick lines, and transmission lines and transformers connect buses with each other. Another common transmission systems model is the more detailed node-breaker model that additionally models connectivity of substations. This is necessary when faults in substations need to be studied. Due to the possible imbalance between phases in distribution systems, distribution system models define all phases, including neutral and ground, which further increases the model complexity.

1.3.2 **Power Flow Analysis**

A formal definition of the steady-state analysis “Power Flow” can be given as finding the voltage phasors for each bus (node) in the frequency domain at the nominal system frequency, given the network model, loads and generation. Wherein, a slack bus provides the required reference angle and additionally supplies the necessary power to balance generation with the loads and losses of the system.
Formulations to solve this problem can be categorized by the choice of the underlying coordinate system and by the choice of the applied balance equations. Historically, the first computer implementation of the Load Flow problem was based on current and voltage state variables in rectangular coordinates and Kirchhoff’s Circuit Law (KCL) equations [13]. However, the de-facto standard formulation is the “PQV”-formulation that is based on polar coordinates and power balance equations [14].

A 1974 paper reviews early techniques used to solve this problem [15]. Due to the lack of computational power and memory, simple algorithms based on stationary iterative methods (Gauss-Seidel, etc.) were commonly preferred over Newton-Raphson (NR) based methods. Nevertheless, the superior convergence properties of Newton-Raphson based methods were already well understood. Modern methods almost exclusively rely on NR to provide efficiency in convergence.

1.3.2.1 THE PQV FORMULATION FOR POWER FLOW

The traditional way to solve the PF problem is the “PQV”-formulation introduced by Tinney and Hart in 1967 [14]. It formulates power balance equations in polar coordinates for each bus $i$:

$$P_{G,i} - P_{L,i} = |V_i| \sum_{k=1}^{n_B} |V_k|(G_{ik} \cos(\phi_{ik}) + B_{ik} \sin(\phi_{ik}))$$

(1)

$$Q_{G,i} - Q_{L,i} = |V_i| \sum_{k=1}^{n_B} |V_k|(G_{ik} \sin(\phi_{ik}) - B_{ik} \cos(\phi_{ik}))$$

(2)

where $P_{G,i}$ and $Q_{G,i}$ is the generated real and reactive power, $P_{L,i}$ and $Q_{L,i}$ describe the real and reactive power demand, and $|V_i|$ is the voltage magnitude on bus $i$. Furthermore, for each of the $n_B$ branches adjacent to bus $i$, a term corresponding to the branch flow is summed on the right-hand side. Herein, $|V_k|$ is the voltage magnitude on the adjacent bus $k$ and $\phi_{ik}$ is the voltage angle between buses $i$ and $k$. Finally, $G_{ik}$ and $B_{ik}$ are the branch admittance and susceptance. Eqs. (1) and (2) are further solved by formulating a Jacobian matrix and using the Newton-Raphson numerical algorithm. Despite being the de-facto standard, this formulation introduces difficult to handle trigonometric nonlinearities stemming from the network equations.
Commonly, this results in a lack of robustness of the numerical algorithm for more complex cases, or cases where no good initial guess can be provided.

1.3.2.2 The Current Injection Method (CIM)

Alternatively, the CIM [16] formulates the problem in rectangular coordinates and KCL equations. Its roots can be traced back to a 1970 paper [17] that, among other formulations, describes a current equation-based formulation to solve PF. Furthermore, [17] proposes to split real and imaginary parts of the equations to enable numerical solution via Newton-Raphson. Similarly, the CIM formulates the current mismatch \( \Delta l_k^2 \) on a single phase of bus k is given as:

\[
\Delta l_k^2 = \frac{P_k^{sp} + j Q_k^{sp}}{V_k^*} - \sum_{i \in \Omega_k} Y_{ki}^e V_i^t
\]  

(3)

where \( P_k^{sp} \) and \( Q_k^{sp} \) are the specified real and imaginary power injections, \( V_k \) is the complex voltage on bus k with the complex conjugate operator ‘\(^*\)’, \( Y_{ki}^e \) is the admittance matrix element between buses \( k \) and \( i \), and the summation is over all \( \Omega_k \) buses connected to bus k. Just as described in [17], this equation is further split into real and imaginary parts and solved using the Newton-Raphson algorithm.

Formulating the problem using KCL and rectangular coordinates relaxes the trigonometric network nonlinearities that compromise the robustness of the “PQV”-formulation. Instead, nonlinearities within load and generator models are introduced. Enforcing generator voltage magnitude constraints proves to be especially challenging in this formulation. An improved model for enforcing these constraints was introduced in 2004 [18]. Still, due to this difficulty, the CIM is mostly used for distribution system analysis, where generators controlling voltage magnitudes are very few or nonexistent.

1.3.3 State Estimation Analysis

The second algorithm this work touches on is Power System State Estimation (SE). It uses measurement data and a power grid topology model to estimate the most likely state of a system. It is important to note that the topology model is the result of a different algorithm: Topology Estimation, that uses binary grid measurements to establish the grid’s most likely topology.
The Power Systems industry utilizes proprietary Energy Management Systems to observe and control a system. EMSs include a whole toolchain of interconnected algorithms (Topology Estimation, Bad Data Detection, and SE) to best find the most likely state of the system. For an observer, each individual algorithm is not distinguishable. Academic research, on the other hand, clearly separates these algorithms. Without losing the focus on their interdependences, we will use the academic definition of SE that is traditionally an algorithm concerned with real-valued measurement data only.

The most common measurement device in Power Systems is the Remote Terminal Unit (RTU). RTUs have been operating for decades as part of Power Systems’ Supervisory Control and Data Acquisition (SCADA) systems. These units are ubiquitous. They measure voltage and current magnitudes as well as the power-factor angle, which is defined as the angle between the voltage and current. RTU current measurements are either delivering data on node injections, or branch flows on transmission lines, or transformers. However, these current and voltage measurements are further converted into real and reactive power data to simplify inclusion into the prevalent power-based SE algorithms [19]. Interestingly, the time resolution of SCADA data is in the order of seconds [20].

![Fig. 3 PV plant power generation measured with RTUs [20].](image)

Traditional formulations for the State Estimation problem are exclusively concerned with this kind of measurement data. The earliest SE formulation, introduced by Schweppe [21] in 1970, is still considered the standard today. It formulates the problem as a nonlinear weighted least
squares optimization algorithm, wherein the power system state is defined as a set of complex bus voltages in polar coordinates. However, this leads to a highly nonlinear formulation that is known to be challenging to solve. In fact, even modern state estimators suffer from convergence issues in real-world scenarios [22].

A formal definition of Schweppe’s weighted least squares (WLS) state estimator that is also referred to as AC-state estimator can be given as an algorithm with the following objective:

$$\min_x \left( z - h(x) \right)^T W \left( z - h(x) \right)$$

where $z$ is the vector of RTU measurements that includes real and reactive power injection at nodes, voltage magnitude at the bus nodes and real and reactive power flows across the branches; $x$ is a vector of voltage state variables; $h$ is a vector valued measurement function describing the nonlinear relationship between estimated state and measurements; and $W$ is a diagonal weight matrix, where each term corresponds to the inverse of the variance of a given measurement ($1/\sigma_i^2$). Notably, simplified decoupled linear versions of this algorithm exist [23].

Importantly, there are alternative approaches to Schweppe’s SE formulation. A few notable ones are: recently proposed direct noniterative approaches considering SCADA data [24],[25], a cartesian coordinate formulation that can lead to improved convergence [26], and an SE formulation based on the CIM for PF [27].

![Fig. 4 PV plant power generation measured with PMUs](image-url)
More precise measurement devices are the Phasor Measurement Units (PMUs). PMUs measure synchronized voltage and current phasors with high precision and low time resolution (in the order of milliseconds) using geolocation systems. Unfortunately, deployment of PMUs has been slow. In 2015, only 1800 PMUs were installed in North America (US and Canada) [20]. One can see that PMUs currently only observe a small fraction of the system, when comparing this number to a system model of the Eastern United states that has more than 75k nodes. Nevertheless, due to the linearity of SE algorithms that exclusively consider PMUs [28], [29] SE algorithms using their data have been deployed to PS operators [28].

Current industry practice, to have a nonlinear SCADA data-based State Estimator and operate a linear PMU-based estimator in parallel (as a backup), suggests that hybrid approaches that make use of both types of measurement data [30], [31] are still immature. However, only leveraging one kind of measurement data in an algorithm clearly limits the potential output quality of a State Estimator and calls for improvement.

**Research question #1:**

Can we find an approach for State Estimation that alleviates convergence problems and includes both types of measurement devices (RTUs and PMUs) within an efficient algorithm?

### 1.4 Circuit Simulation Methods (SPICE)

Early research in Computer Aided Circuit Design and Circuit Analysis started in the 1950s. A well written overview of early approaches in this field can be found in [32]. It describes multiple generations of simulators that were developed in an environment of competitive simulator designs. This sparked innovation that was additionally driven by the inherent need from industry and the government (more precisely the military) for high quality circuit simulators. The widespread success of these simulators started around 1969-1970 with the development of the simulation program CANCER [33] and the release of its successor SPICE [34] into the public domain, enabling extensive testing and further development of algorithms. This release accelerated the maturation of the software and lead to widespread adoption of these types of circuit simulators.
By the time of the release of SPICE, general purpose simulator were commonly designed using Modified Nodal Analysis (MNA) to describe the circuits, the Newton-Raphson algorithm to numerically find the solution to nonlinear problems, and sparse LU-factorization to iteratively solve the linearized sub-problems [32]. This successful approach remains the standard in general purpose circuit simulators until today.

Development in the field of circuit simulation continued with the need for simulating systems that were exponentially growing in size (for large scale integration “LSI”, and later very large-scale integration “VSLI” of circuits), and the need to handle steep nonlinearities in increasingly complex device models. To address these challenges, techniques to aid convergence, such as the limiting techniques included in CANCER, were not enough, and more complex techniques were developed. Among the most successful were homotopy methods to find the DC-operating point of a circuit, such as Gmin-stepping, source-stepping, or pseudo-transients [35]. These techniques enabled a simulator to robustly find any circuit’s DC operating point. Modern, high performance oriented circuit simulators are able to robustly handle millions of transistors and billions of parasitics in a single simulation [36].

Formulations for simulating power systems started around the same time of the advent of circuit simulation. The 1976 meeting “Workshop on the Application of Circuit Theory to Power System Problems” hints that there were efforts to interact between the two fields. However, history suggests that these efforts did not prove to be fruitful. Innovation in the power system simulation field remained slow and the ‘PQV’-formulation continued to be the most adopted formulation in the field. Even today, convergence problems in power system simulators are commonly addressed by experienced engineers. One explanation for this is the static nature of a power grid, where additional components are only added after years of planning and a redesign of a power system, something that is common practice in circuit design, is unimaginable.

Recent changes in the power systems field, especially the increased share of distributed renewable generation and the rising concern of the effects of blackouts due to environmental or other factors renew the awareness for the need for robust simulation software. This enables a new push towards developing robust solution techniques for power systems. For this push,
inspiration is readily available by the algorithms that were previously developed for robust circuit simulation in many years of extensive research.

1.5 Probabilistic Analysis (Monte Carlo)

The aforementioned recent changes in the power system field have the potential to spark innovation in the way PS analyses are conducted in general. Probabilistic approaches to power system analysis were proposed soon after their deterministic counterparts [6]. They pose clear advantages over deterministic algorithms and will result in more economical and more secure grid operation.

Probabilistic methods are commonly used in other fields that require rigorous study of their underlying processes such as: finance, physics, statistics, and engineering. Therein, the Monte Carlo method has proven to be one of the most effective numerical tools. The term Monte Carlo was coined in the 1940s when the method was used for physical experiments that lead to the atomic bomb. The method, however, is much older than that. It was previously called ‘model sampling’ in statistics [37]. The Monte Carlo method was created by the simple idea to study a system with uncertainties by sampling from it [37]. Its performance depends on statistically good random sampling and sufficient computational resources to study many samples. Importantly, despite its straightforwardness, it is a very powerful method that is general. This often makes it the only viable option for probabilistic analysis [37].

Using the law of large numbers and the Central-Limit theorem we can quantify its convergence rate to be $\frac{1}{\sqrt{n}}$ for n samples. This is much slower than we are used to from many other numerical algorithms. It means that for a one digit increase in the result’s accuracy requires 100 times more samples. To strike a tradeoff between number of samples and accuracy, it is important to periodically estimate the accuracy of the result during a Monte Carlo simulation.

Due to this comparatively slow convergence, different convergence improving techniques for Monte Carlo have been proposed. They include variance reduction techniques such as antithetic sampling, stratified sampling, or importance sampling. The term Quasi Monte Carlo was established to describe techniques that grow out of these variance reduction techniques. They commonly use a deterministic approach to select samples in order to cover the sample space in
a certain way [37]. However, the improved convergence rate of these methods comes at the cost of generality. Commonly, employing these methods requires assumptions about the ‘effective’ dimensionality of a problem [38], [39].

Our aim is to design a general probabilistic solver for Power System applications. Hence, we plan to use Monte Carlo in its most general way. However, we do this while acknowledging the potential of variance reduction techniques or Quasi Monte Carlo to lower the computational complexity of a problem in special situations.

**Research question #2:**

Can we develop a general robust probabilistic engine for probabilistic power system analysis that operates within operational time constraints for SE and within engineering cycles for PF on system models spanning continental interconnections?
1.6 Overview of this Thesis

In the remainder of this thesis we present research that addresses the posed research questions. Chapter 3 first discusses details of the ECF for Power System problems, including a formulation for Power Flow, an optimization algorithm to analyze the feasibility of a PF problem, and a linear optimization algorithm for AC Power System State Estimation. Chapter 3 further explores the theoretical background and the necessary tools to solve these problems robustly and efficiently. It touches Modified Nodal Analysis, limiting methods, homotopy methods, linear sparse matrix solvers, and probabilistic methods.

Chapter 4 documents the implementation of a prototype probabilistic solver of ECF problems in C++. It includes a description of implemented modules; their interfaces; the implementation of generic control flows, as well as a description of input and output file structures. Description of the external libraries that were utilized are also presented.

Chapter 5 presents algorithms and displays results. Specifically, algorithms designed to robustly and efficiently solve Power Flow problems are presented. Results include data of Probabilistic Power Flow and State Estimation experiments on system models with up to the size of continental-interconnections. Furthermore, the effects of correlations between the random variables are studied. Finally, probabilistic algorithms to model control are presented. Chapter 6 summarizes the conclusions of this thesis and discusses future directions of this work.
2 BACKGROUND: EQUIVALENT CIRCUIT FORMULATION

This chapter introduces the Equivalent Circuit Formulation (ECF) for power system problems and discusses the required background to develop a probabilistic ECF-based solver. First, we formulate the AC Power Flow problem as an ECF problem and derive ECF models for all traditional PF components. Subsequently, we define optimization algorithms to study PF feasibility and AC Power System State Estimation. Furthermore, we present how these optimization algorithms are fully solved within the ECF. Next, we discuss algorithms that are utilized to solve ECF problems in more depth. We touch on Modified Nodal Analysis, sparse matrix solution, limiting, and homotopy methods. This chapter closes with an overview of probabilistic algorithms.

2.1 EQUIVALENT CIRCUIT FORMULATION

The Equivalent Circuit Formulation applies a “circuit formalism” to Power System problems. Applying a circuit formalism means taking a sequence of steps to derive an equivalent circuit of a problem and solving this equivalent problem using circuit simulation techniques. The steps of this formalism in the context of Power System simulation are:

1. formulating element models (lines, loads, generators, ...) in terms of current and voltage state variables
2. linearizing nonlinear element models using a first order Taylor expansion
3. formulating difference equations of the linearized models between two NR steps
4. mapping the linear and linearized nonlinear equations into equivalent circuit models
5. employing Modified Nodal Analysis to create the linearized system-circuit of the problem
6. applying Newton-Raphson with updated linearized problems until convergence

The circuit formalism enables physical understanding of the problem in terms of the equivalent circuit. More importantly, it enables the application of circuit simulation techniques, such as limiting and homotopy methods, to improve simulator robustness.
2.1.1 Power Flow

The historical context and a short overview of traditional PF formulations have already been discussed in the Introduction. We have established that the de-facto standard, the “PQV”-formulation, lacks robustness partly due to its non-physical (purely mathematical) representation of the problem. In fact, the employed power-balance equations can be viewed as adapted KCL equations that are reinterpreted for complex powers:

\[
\bar{I} = \bar{V}\bar{Y}
\]  \hspace{1cm} (5)

\[
\bar{S} = \bar{V}\bar{I}^* = \bar{V}(\bar{Y}\bar{V})^*
\]  \hspace{1cm} (6)

\[
\bar{S}_i = \bar{V}_i \sum_{n=1}^{k} \bar{Y}_{ik}^* \bar{V}_k^*
\]  \hspace{1cm} (7)

Here, (5) represents a KCL governed complex linear system with the voltage vector \( \bar{V} \), the current vector \( \bar{I}_i \), and the linear admittance matrix \( \bar{Y} \). Substituting (5) into the definition of the complex apparent power vector \( \bar{S} \) of (6) results in an equation that is equivalent to the power balance equations employed in the “PQV”-formulation. This becomes apparent when looking at (7) that formulates (6) for a selected bus \( i \) and comparing (7) to the “PQV” power balance equations (1),(2).

Fig. 5 Diagram of linear (green), nonlinear (red), and omitted (blue) system elements within the “PQV” formulation.

Fig. 5 depicts the resulting nonlinearities of the “PQV” formulation. Herein, the power-based macro models are linear, whereas the network models (transformers and transmission lines) are nonlinear. Slack buses are omitted in the “PQV” formulation. Importantly, the nonlinear network equations introduce trigonometric functions that are known to be difficult to handle with NR.
Applying a KCL based method in rectangular coordinates (such as the CIM [16], early KCL based methods [17], or the ECF) leads to a different set of nonlinearities. Now, the network and slack-bus models are linear, whereas power-based macro models lead to quadratic nonlinearities. Fig. 6 identifies the nonlinearities for the current-voltage based formulations.

One of the major challenges for KCL based rectangular coordinate formulations is integrating voltage magnitude constraints of PV-generators. A key insight to solve this challenge was proposed in [18], which linearizes the voltage magnitude circle-equation and identifies the generator’s reactive power Q as an additional state variable. The ECF approach is equivalent to [18] in terms of its formulation. However, the key to simulation robustness does not entirely depend on the chosen formulation. Rather, a good physical understanding of the problem, as enabled by the circuit formalism, allows for the application of well-understood convergence-improving techniques that are difficult to design without this understanding.

Mathematically, the ECF for PF aims to solve the following nonlinear problem:

$$\tilde{I}(V) = \tilde{Y}(V)\tilde{V}$$

(8)

where $\tilde{V}$ is the vector of complex state variables, $\tilde{Y}(V)$ is a complex nonlinear vector-valued function that can be split into a linear part (i.e. the system admittance matrix) and a nonlinear part that includes terms that stem from nonlinear models, and $\tilde{I}(V)$ is a vector of current injections that depends on the state vector $\tilde{V}$ due to nonlinear model terms.

Choosing rectangular coordinates and splitting (8) into coupled real and imaginary parts to fulfill the analyticity requirement of NR for constant power constraints, enables the application of the Newton-Raphson algorithm to solve this nonlinear problem. To find the linearized NR sub-problems of (8), the ECF applies the aforementioned circuit formalism. The linearized versions of $\tilde{Y}(V)$ and $\tilde{I}(V)$ for each NR iteration are hierarchically built using equivalent circuit models of the PF problem which we derive next.
2.1.1.1 Power Flow ECF Models

We start by deriving the linear ECF models for transmission lines and transformers. To find the ECF model of a component we express its governing equations in terms of currents and voltages and further map them into linear equivalent circuit elements.

Transmission Line Pi-Model

Transmission lines are commonly modeled as pi-models that have a line impedance and a shunt element on each connecting bus. A transmission line pi-model is depicted in Fig. 7. To find its equivalent circuit model we formulate Ohm’s law for its shunt and series elements. Expressing the shunt current $I_{sh,ik}$ in terms of the terminal voltages $\tilde{V}_{l,k}$ leads to

$$I_{sh,ik} = \frac{B_{sh}}{2} \tilde{V}_{l,k} \quad (9)$$

Furthermore, the current through the series element $I_s$ can be expressed as

$$I_s = \frac{\tilde{V}_{lk}}{(R_L + jX_L)} \quad (10)$$

Next, we split the complex expressions in (9) and (10) into their real and imaginary parts, where splitting (9) leads to

$$I_{sh,ik}^R = \frac{B_{sh}}{2} V_{l,k}^I \quad (11)$$

$$I_{sh,ik}^I = \frac{B_{sh}}{2} V_{l,k}^R \quad (12)$$

and splitting (10) leads to

$$I_s^R = V_{ik}^R \frac{R_L}{R_L^2 + X_L^2} + V_{ik}^I \frac{X_L}{R_L^2 + X_L^2} \quad (13)$$
\[ I_z^i = V_{ik}^i \frac{R_L}{R_L^2 + X_L^2} - V_{ik}^R \frac{X_L}{R_L^2 + X_L^2} \] (14)

We can now construct the ECF transmission line model by mapping (11)-(14) into linear equivalent circuit elements. This is done by simple analogies. A term of (11)-(14) connecting a voltage with its own current corresponds to a resistor, whereas a term connecting a voltage with a different current corresponds to a voltage-controlled current source.

Fig. 8 Real and imaginary parts of a transmission line ECF model.

Fig. 8 shows the real and imaginary parts of an ECF transmission line pi-model. Importantly, here the current controlled current sources couple the real and imaginary circuits.

**Transformer Model**

ECF models of transformers, including phase shifting transformers, are linear within the ECF as well.

Fig. 9 Diagram of a 2-phase transformer model and its parameters.

Fig. 9 shows a diagram of a transformer model and its components. We can express the relationship between the primary- and the secondary voltages (\( \tilde{V}_1 \) and \( \tilde{V}_2 \)) and currents (\( \tilde{I}_1 \) and \( \tilde{I}_2 \)) of a transformer before losses as

\[ \tilde{V}_1 = \tilde{V}_2 \ t_r \ e^{j\theta_{shift}} \] (15)

\[ \tilde{I}_2 = -\tilde{I}_1 \ t_r \ e^{-j\theta_{shift}} \] (16)
We further split (15) and (16) into real and imaginary parts

\[ V_1^R = t_r (V_2^R \cos(\theta_{shift}) - V_2^I \sin(\theta_{shift})) \]  
(17)

\[ V_1^I = t_r (V_2^R \sin(\theta_{shift}) + V_2^I \cos(\theta_{shift})) \]  
(18)

\[ I_2^R = -t_r (I_1^R \cos(\theta_{shift}) + I_1^I \sin(\theta_{shift})) \]  
(19)

\[ I_2^I = -t_r (-I_1^R \sin(\theta_{shift}) + I_1^I \cos(\theta_{shift})) \]  
(20)

Now, equations (17)-(20) can be mapped into primary and secondary-side ECF models of a transformer. For the primary side, (17) and (18) are mapped into series connected voltage controlled voltage sources defining the real and imaginary primary voltages \( V_1^R \) and \( V_1^I \). Further, the secondary currents \( I_2^R \) and \( I_2^I \), can be expressed as primary current-controlled current sources. Finally, the series loss elements, defined by \( Y_{loss} = G_{loss} + jB_{loss} \) in Fig. 9 are considered on the secondary side. Their derivation is equivalent to the derivation of the series elements in the transmission line model (Eqns. (13) and (14)).

![Fig. 10 ECF transformer model including phase shifts and series losses.](image)

Fig. 10 shows the full ECF model of a potentially phase shifting transformer including series losses. This model is straight forward to derive from the governing equations of the PF transformer model. However, a drawback of this model is that it requires additional intermediary nodes on primary and secondary ECF model-sides and auxiliary equations that are required for the controlled voltage sources. As we will see when discussing Modified Nodal Analysis, this can dramatically increase the system matrix for systems with many transformers.

An alternative ECF transformer model that does not require additional nodes or auxiliary equations can be derived by including the series losses of Fig. 9 into the derivation. We do this
by relating the primary voltage $\tilde{V}_1$ with the secondary voltage after losses $\tilde{V}_{2,l}$. For this, we split the losses $\tilde{V}_{\text{loss}} = G_{\text{loss}} + j B_{\text{loss}}$ into real and imaginary components, and apply Ohm’s law between $\tilde{V}_{2,l}$ and $\tilde{V}_2$:

$$\tilde{I}_2 = (G_{\text{loss}} + j B_{\text{loss}}) (\tilde{V}_{2,l} - \tilde{V}_2) \quad (21)$$

Further, we substitute the expressions for $\tilde{V}_2$ and $\tilde{I}_2$ from Eqs. (15) and (16) into Eq. (21) and arrive at an expression for the primary current $\tilde{I}_1$ in terms of the voltages $\tilde{V}_1$ and $\tilde{V}_{2,l}$:

$$\tilde{I}_1 = \frac{(G_{\text{loss}} + j B_{\text{loss}})}{t_r^2} (\tilde{V}_1 - \tilde{V}_{2,l} e^{j \theta_{\text{shift}}}) \quad (22)$$

Finally, by only substituting $\tilde{V}_2$ from Eq. (15) into Eq. (21) we find an expression for the secondary current $\tilde{I}_2$ in terms of both terminal voltages

$$\tilde{I}_2 = (G_{\text{loss}} + j B_{\text{loss}}) (\tilde{V}_{2,l} - \tilde{V}_1 e^{-j \theta_{\text{shift}}} t_r) \quad (23)$$

Following the same formalism as in the previous models, i.e., splitting (22) and (23) into their real and imaginary parts and further mapping them into equivalent circuits, we arrive at a compact transformer model that does not require an extension of the system matrix.

**SHUNT AND CONSTANT IMPEDANCE LOAD MODEL**

Shunts are linear elements that model system losses such as the capacitive losses of transmission lines. Constant impedance loads, which are commonly part of ZIP-load models, are modeled equivalently. A shunt or constant impedance load is defined as an admittance from a bus to ground.

![Fig. 11 Shunt from a bus to ground.](image)

A circuit diagram of a shunt can be seen in Fig. 11, while Fig. 12 shows the corresponding equivalent circuit model that can be found formulating Ohm’s law by expressing the shunt...
current $I_{sh}$ in terms of its voltage $\bar{V}_{sh}$ and further splitting the complex equation into real and imaginary parts.

![Real and imaginary equivalent circuit models of a shunt.](image)

**LOAD MODELS**

The PQ load model models a constant real and reactive power load. It is the most common load model today. However, it has been recognized that this model does not capture the full behavior of many real-world (aggregated) loads [40]. Especially its fixed voltage sensitivity raises concern. The ZIP-load model was designed as an improvement over the PQ-load model. It adds constant current and constant impedance load parameters.

**PQ-LOAD MODEL**

The ECF PQ-load model is derived by expressing the constant apparent power $\bar{S}_L$ by its real and imaginary components $P_L$ and $Q_L$ (the PQ-load parameters)

$$\bar{S}_L = P_L + jQ_L = \bar{V} I^*$$  \[24\]

and further rewritten it in terms of (complex) current and voltage state variables

$$I = \frac{P_L - jQ_L}{\bar{V}}$$  \[25\]

with the complex conjugate operator ($^*$) on the voltage vector $\bar{V}$. Equation (25) is further split into its real and imaginary parts, which is crucial for this model due to the non-analyticity of the complex conjugate operator. The split equations are

$$I^R_L = \frac{V^R P_L + V^I Q_L}{\bar{V}^2 + \bar{V}_r^2}$$  \[26\]

$$I^I_L = \frac{P_L V_I - Q_L V_R}{\bar{V}^2 + \bar{V}_r^2}$$  \[27\]

Eqns. (26) are (27) nonlinear equations in terms of the current and voltage state variables. These nonlinearities force us to apply an iterative numerical solution method (Newton-Raphson). To
prepare for this algorithm, the first order Taylor-expansion is applied to linearize (26) and (27) between two fictitious NR steps $k$ and $(k + 1)$

$$I_{L}^{R,k+1} = \frac{\partial I_{L}^{R}}{\partial V^{R}}\bigg|_{k} (V^{R,k+1} - V^{R,k}) + \frac{\partial I_{L}^{R}}{\partial V^{I}}\bigg|_{k} (V^{I,k+1} - V^{I,k}) + I_{L}^{R,k}$$  \hspace{1cm} (28)

$$I_{L}^{I,k+1} = \frac{\partial I_{L}^{I}}{\partial V^{R}}\bigg|_{k} (V^{R,k+1} - V^{R,k}) + \frac{\partial I_{L}^{I}}{\partial V^{I}}\bigg|_{k} (V^{I,k+1} - V^{I,k}) + I_{L}^{I,k}$$ \hspace{1cm} (29)

where the partial derivatives are evaluated at the NR iteration $k$, at which all terms are known. The linearized Eqns. (28) and (29) are now mapped into linear equivalent circuits that are updated at each NR iteration. Interestingly, to do this efficiently circuit simulators evaluate symbolic expressions of these partial derivatives that are stored within each model.

Fig. 13 shows the linearized and split PQ load model. Importantly, in Fig. 13 the terms of (28) and (29) are further simplified by lumping all known terms (the terms of iteration $k$) together into the independent current sources $I_{eq}^{R,k}$ and $I_{eq}^{I,k}$.

**ZIP-LOAD MODEL**

The ZIP-load model can be split into three components: the previously derived shunt, PQ-load model, and a constant current load model that we discuss next. The ZIP load model defines the constant current phasor $(I_{p} + jI_{q})$ in relation to the unknown local bus voltage angle. To derive a constant current model with this constraint, we start with reformulating the current and voltage terms in polar coordinates:

$$I_{p} + jI_{q} = I_{m}e^{j\theta_{I}}$$  \hspace{1cm} (30)

$$V^{R} + jV^{I} = V_{m}e^{j\theta_{V}}$$  \hspace{1cm} (31)

Now, the ECF model’s current angle is set in reference to its voltage angle $\theta_{V}$

$$I_{m}e^{j\theta_{I} - \theta_{V}}$$  \hspace{1cm} (32)
and, coming back to rectangular coordinates, we split Eq. (32) into real and imaginary parts

\[ I_m^R = I_m \cos(\theta_l - \theta_v) \quad \text{(33)} \]

\[ I_m^I = -I_m \sin(\theta_l - \theta_v) \quad \text{(34)} \]

Furthermore, following the circuit formalism, we linearize the nonlinear equations and formulate them in terms of difference equations between the NR iterations \( k \) and \( k+1 \).

\[ I_m^{R,k+1} = \frac{\partial I_m^R}{\partial V^R} \Big|_k (V^{R,k+1} - V^{R,k}) + \frac{\partial I_m^R}{\partial I^I} \Big|_k (V^{I,k+1} - V^{I,k}) + I_m^{R,k} \quad \text{(35)} \]

\[ I_m^{I,k+1} = \frac{\partial I_m^I}{\partial V^R} \Big|_k (V^{R,k+1} - V^{R,k}) + \frac{\partial I_m^I}{\partial I^I} \Big|_k (V^{I,k+1} - V^{I,k}) + I_m^{I,k} \quad \text{(36)} \]

Finally, Eqns. (35) and (36) can be mapped into equivalent linearized circuits that are updated at each NR step.

It is important to note that defining the constant current angle in reference to its own bus voltage angle results in partial derivatives that include highly nonlinear trigonometric functions. This raises concern of potential robustness issues for system models that contain a large number of constant current (ZIP) load models. An alternative modeling approach that includes constant currents that results in a linear model within the ECF was proposed in [40].

**Generator model**

A generator is traditionally modeled in terms of a real power injection \( P_G \) into the system and a voltage magnitude set-point \( V_G \) at the generator’s bus that is controlled by a variable reactive power contribution of the generator \( (Q_G) \). Similar to the PQ-load model, we can start with (24) to formulate the power constraint into current and voltage state variables and reformulate it in terms of currents that are further split into real and imaginary parts

\[ I_{G,R} = \frac{V^R P_G + V^I Q_G}{V^{R^2} + V^{I^2}} \quad \text{(37)} \]

\[ I_{G,J} = \frac{P_G V^I - Q_G V^R}{V^{R^2} + V^{I^2}} \quad \text{(38)} \]

However, now, the generated reactive power is a controlled variable.

\[ I_{L,R}^{k+1} = \left. \frac{\partial I_{L,R}^R}{\partial V^R} \right|_k (V^{R,k+1} - V^{R,k}) + \left. \frac{\partial I_{L,R}^R}{\partial I^I} \right|_k (V^{I,k+1} - V^{I,k}) + \left. \frac{\partial I_{L,R}^R}{\partial Q_G} \right|_k (Q_G^{k+1} - Q_G^k) + I_{L,R}^{R,k} \quad \text{(39)} \]
\[ I_{L}^{k+1} = \frac{\partial I_{L}^{k}}{\partial V^{R}} \left( V_{R}^{k+1} - V_{R}^{k} \right) + \frac{\partial I_{L}^{k}}{\partial V^{I}} \left( V_{I}^{k+1} - V_{I}^{k} \right) + \frac{\partial I_{L}^{k}}{\partial Q_{G}} \left( Q_{G}^{k+1} - Q_{G}^{k} \right) + I_{L}^{k} \] (40)

Further (39) and (40) are interpreted as linearized equivalent circuits, by mapping terms at iteration \((k + 1)\) into equivalent circuit elements and lumping the known terms of iteration \(k\) into independent current sources.

Fig. 14 depicts the resulting equivalent circuit model that corresponds to the power injection part of the model.

Additionally, a generator’s voltage magnitude constraint has to be enforced. In rectangular coordinates, this constraint can be written as

\[ V_{G}^{2} = V^{R^{2}} + V^{I^{2}} \] (41)

Equation (41) is a nonlinear circle equation. In order to enforce it within the NR algorithm it is reformulated as a function \(F_V(V^R, V^I)\)

\[ F_V(V^R, V^I) := V_{G}^{2} - V^{R^{2}} - V^{I^{2}} \] (42)

and further linearize it, using a Taylor expansion. Finally, (42) is expressed in terms of a difference equation between two NR iterations

\[ F_{V}^{k+1} = F_{V}^{k} - \frac{\partial F_{V}}{\partial V^{R}} \bigg|_{k} V_{R}^{k+1} - V_{R}^{k} + \frac{\partial F_{V}}{\partial V^{I}} \bigg|_{k} V_{I}^{k+1} - V_{I}^{k} \] (43)

which is again mapped into an equivalent linearized circuit that is updated with each NR iteration to enforce the constraint.

Importantly, the voltage magnitude constraint acts as implicit control for the generated reactive power \(Q_{G}\). For more complex power system problems, multiple generators are modeled to enforce a single voltage constraint on a bus that is different from their own. Their share of
generated reactive power is modeled as fractions of the total reactive power necessary to enforce this constraint (participation factors). This can easily be modeled within the ECF by stamping only one voltage constraint for the controlled bus. The necessary reactive power is then supplied by a mix of the participating generators. Mathematically this is done by including the respective generator reactive power participation factors into their power injection equations ((37) and (38)). Section 3.1.4.3 describes how this is implemented as a “plant model” within our solver.

Finally, it is important to mention that this “PV” model of a generator is a non-physical macro model. This becomes apparent when a generator reaches the maximum or minimum reactive power it can generate. In this case, it is no longer able to control the voltage magnitude. To model this situation, current simulation software is forced to switch the PV-generator into a constant PQ-model, injecting its defined real power and, depending on which state was reached, its minimum or maximum reactive power. This non-continuous switching is problematic. It can result in switching loops or deliver nonphysical solutions. An early ECF-based approach to solve this problem was proposed in [41].

**Slack Bus Model**

In Power Flow, the slack bus defines the system’s phase reference angle and delivers the required power to balance generation with load and system losses. Due to this, a generator bus is usually selected as a slack bus. A slack bus model is defined by a constant magnitude $V_m$ at a constant angle $\theta_s$. Equation (44) rewrites the slack bus constraint in rectangular coordinates

$$\tilde{V}_{\text{slack}} = V_m e^{j \theta_s} = V^R_S + j V^I_S \quad (44)$$

Here, the mapping to an ECF model is trivial. A slack bus model consists of two independent voltage sources representing the real and imaginary slack bus voltage, $V^R_S$ and $V^I_S$, respectively. Fig. 15 depicts this model.

---

Fig. 15 Equivalent Circuit Model of a Slack bus.
It is important to note that the validity of the slack bus model is limited to system states that are close enough to the state the system model was created for. Alterations in the system model potentially require a slack bus to deliver an infinite amount of additional power. It is obvious that PF solutions with high slack bus powers due to system model changes that were not matched with changes in generator setpoints have limited validity. We will return to this point when we discuss our probabilistic PF approach.

2.1.2 Feasibility

In extreme cases, the slack bus assumption can result in a PF problem that does not have a numerical solution (infeasible problem). However, this is only one potential cause of a Power Flow infeasibility. Importantly, when no solution to a PF problem is found using a numerical algorithm, it is unclear if there is no solution, or if the numerical algorithm was not able to recover an existing solution due to a lack of robustness. Robust solution algorithms like Tx-stepping (see Section 2.2.4.1) are a big step towards reliably recovering PF solutions. Still, when no solution is found, some uncertainty about a system’s infeasibility remains without a formal global convergence proof of the robust algorithm.

An early discussion on infeasible PF regions can be found in [42]. In [43], a predictor corrector method was applied to identify boundary points and further trace the solution spaces of small power flow cases. In order to identify possible infeasibilities, [44] provides a network-graph based method on finding PF solution bounds, whereas [45] utilizes a semidefinite programming-based method to provide sufficient conditions for infeasible cases. Previous approaches to tackle this problem and find (all) possible solutions are often based on numerical continuations, such predictor-corrector based methods. For example [46] aims to identify boundaries using the continuation power flow method [47]. Alternatively, optimization approaches extending the original PF problem are applied to identify infeasible regions. A mixed complimentary approach focusing mostly on PF robustness can be found in [48]. A least squares optimization based algorithm to identify infeasible cases, provide a measure of their infeasibility, and find an optimal direction to counter the infeasibility was proposed in [49]. A similar approach focusing on the subproblem of load adjustments can be found in [50].
In the ECF, PF infeasibility translates to a single condition: In the case of infeasibility, at least one of the nodal KCL equations cannot be satisfied. With this in mind, the authors of [51] apply an optimization algorithm that recasts the PF problem to an optimization problem with a guaranteed existing solution. This is achieved by adding current injections to each, or a select set of system nodes, and further formulating an optimization problem to minimize these injections.

![Diagram of IEEE 14-bus system with feasibility sources](https://immersive.erc.monash.edu/stac/)

Fig. 16 Feasibility sources placed on each bus of the IEEE 14-bus system.

This algorithm has the following properties: If a PF solution exists, the “feasibility current sources” are minimized to zero and the original PF solution is recovered. If no solution exists, the minimal current sources are nonzero. In fact, the nonzero current sources locate and quantify the system’s infeasibilities [51]. Most importantly, this optimization problem can fully be formulated and solved in terms of equivalent circuits.

Mathematically, this problem is defined as

\[
\min_{I_F} \frac{1}{2} \| I_F \|_2^2 \tag{45}
\]

s.t. \( Y_{GB} V + I(V) = I_F \tag{46} \)

where equation (45) represents its objective function, and equation (46) is an equality constraint to satisfy KCL at every system node, i.e. the ECF constraint for Power Flow including the additional sources \( I_F \). Importantly, due to the nonlinearities in the PF constraint, this is a nonlinear, non-

* figure partly created at [https://immersive.erc.monash.edu/stac/](https://immersive.erc.monash.edu/stac/)
convex optimization problem. This class of problems is difficult to solve without detailed system knowledge [52]. Nevertheless, one can attempt to find a solution by formulating the Lagrangian

\[ \mathcal{L}(V, I_F, \lambda) = \frac{1}{2} \| I_F \|_2^2 + \lambda^T (Y_{GB} V + I(V) - I_F) \]  

(47)

and differentiating it to obtain the necessary (KKT) optimality conditions [52]

\[ \frac{\partial \mathcal{L}}{\partial \lambda} \rightarrow Y_{GB} V + I(V) - I_F = 0 \]  

(48)

\[ \frac{\partial \mathcal{L}}{\partial V} \rightarrow [Y_{GB}^T + J^T(V)] \lambda = 0 \]  

(49)

\[ \frac{\partial \mathcal{L}}{\partial I_F} \rightarrow I_F = \lambda \]  

(50)

Now, one can find an optimal solution by solving (48)-(50). Importantly, (48) and (49) have a form that can be interpreted as equivalent circuits. Here, Equation (48) represents the equivalent circuit PF problem including feasibility current injections \( I_F \), and equation (49) is its adjoint problem with a first order sensitivity matrix \( J(V) \). Finally, eq. (50) defines the values of the feasibility sources \( I_F \) to be equal to \( \lambda \).

In the adjoint domain, corresponding to (49), \( \lambda \) is interpreted as an adjoint voltage state vector of a system of nonlinear equivalent circuit equations with a linear system matrix \( Y_{GB}^T \) and a nonlinear contribution provided by \( J(V) \). Interestingly, the adjoint system matrix \( Y_{GB}^T \) is the transposed (Hermitian in the complex case) of the primal (Power Flow) problem’s system matrix.

This relationship between primal and adjoint system matrix is a known result and was previously exploited by circuit optimization algorithms. It emerges from the adjoint network theory based on Tellegen’s theorem. The adjoint network theory for circuit optimizations is a well-studied method that was introduced in [53], [54] and was applied to optimization based algorithms for circuit design. Notably, the Power Systems field has given this concept some attention as well [55], [56].

2.1.2.1 ANALOGY TO THE ADJOINT NETWORK THEORY

Adjoint networks can be derived in the following way: Starting with Tellegen’s Theorem stating that for (complex) branch current and voltage vectors \( (\bar{I}, \bar{V}) \) and a primal network \( \mathcal{N} \), its adjoint branch current and voltage vectors \( \bar{\xi}, \bar{\lambda} \) in a topologically equivalent adjoint network \( \mathcal{N}^\ast \) the following holds
\begin{align*}
I^T \tilde{V} &= \tilde{Z}^T \tilde{\lambda} = \tilde{Z}^T \tilde{V} = I^T \tilde{\lambda} = 0 \\
I^T \tilde{\lambda} - \tilde{Z}^T \tilde{V} &= 0
\end{align*}

Hence, (52) is true for \( N \) and \( \tilde{N} \) as well. Looking at a linear circuit with a complex admittance matrix \( \tilde{Y} \) and substituting it into (52) leads to
\begin{align*}
\tilde{I} &= \tilde{Y} \tilde{V} \\
\tilde{V}^T (\tilde{Y}^H \tilde{\lambda}) &= \tilde{V}^T \tilde{\xi} \\
\tilde{\xi} &= \tilde{Y}^H \tilde{\lambda}
\end{align*}

Eq. (55) defines the adjoint admittance matrix to be the Hermitian of the admittance matrix in (53). Indeed, this is the same correspondence that was found by formulating the necessary KKT conditions of our optimization problem (in (48) and (49)). However, with this result coming from adjoint circuit theory, one can now follow the findings of this theory and map the linear PF ECF models directly to their adjoint counterparts. Table 1 defines this mapping

<table>
<thead>
<tr>
<th>power flow circuit</th>
<th>( \rightarrow )</th>
<th>adjoint circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent voltage source</td>
<td>( \rightarrow )</td>
<td>short</td>
</tr>
<tr>
<td>independent current source</td>
<td>( \rightarrow )</td>
<td>open</td>
</tr>
<tr>
<td>capacitor</td>
<td>( \rightarrow )</td>
<td>inductor</td>
</tr>
<tr>
<td>resistance</td>
<td>( \rightarrow )</td>
<td>resistance</td>
</tr>
<tr>
<td>inductor</td>
<td>( \rightarrow )</td>
<td>capacitor</td>
</tr>
</tbody>
</table>

For nonlinear models, however, this analogy does not directly hold. To solve this problem, in [51] they derive an extension to the adjoint network theory. Therein, the nonlinear first order sensitivity matrix \( J^T (V) \) of (49) includes additional terms that are derived individually for each nonlinear model.

2.1.2.2 Linear Adjoint Models

The adjoint split circuits of linear power flow models can be derived using the analogies presented in Table 1. Alternatively, these analogies can be found by taking the Hermitian of an element’s complex governing equations in matrix form (compare (55)).
**ADJOINT TRANSMISSION LINE MODEL**

A transmission line pi-model is defined by its series impedance \( Z = R + jX \) and its shunt capacitances \( \frac{B_{sh}}{2} \) at both terminals \((i, k)\). Formulating Ohm’s law and following one of the two described ways to find the adjoint circuit elements, one finds that its adjoint circuit is governed by

\[
\bar{x}_s = (G - jB)(\bar{\lambda}_i - \bar{\lambda}_k) \\
\bar{x}_{i,k} = \frac{jB_{sh}}{2} \bar{\lambda}_{i,k}
\]

where \( \bar{x}_s \) is the current through the adjoint series element and \( \bar{x}_{i,k} \) are the adjoint shunt currents at buses \( i \) and \( k \). Splitting (56) and (57) into their real and imaginary components and following the circuit formalism, one arrives at the adjoint circuit of a pi model transmission line.

![Fig. 17 Adjoint equivalent circuit of a transmission line pi-model.](image)

Fig. 17 shows the adjoint ECF pi-model of a transmission line, with the complex adjoint current \( \bar{x}_{i,k} = x^R_{i,k} + jx^I_{i,k} \) and the complex adjoint voltages \( \bar{\lambda}_{i,k} = \lambda^R_{i,k} + j\lambda^I_{i,k} \) split into their real and imaginary components.

**ADJOINT TRANSFORMER MODEL**

Starting from the transformer model without auxiliary variables that is governed by (22) and (23), one can find its adjoint governing equations by the relationship between primary and adjoint system defined in (55) to be

\[
\bar{x}_1 = \frac{(G_{toss} - jB_{toss})}{t_r^2} (\bar{\lambda}_1 - \bar{\lambda}_{2,l} t_r e^{-j\theta_{shift}})
\]

(58)
\[
\tilde{x}_2 = (G_{\text{loss}} - jB_{\text{loss}}) \left( \bar{\lambda}_{2,1} - \bar{\lambda}_1 e^{j\theta_{\text{shift}}} \right)
\] (59)

which can be directly mapped into a transformer’s adjoint equivalent circuit by splitting (58) and (59) into their real and imaginary parts. Fig. 18 depicts this adjoint model.

Fig. 18 adjoint equivalent circuit models of a transformer’s primary and secondary sides.

Importantly, these adjoint models remain unchanged for any ECF based optimization algorithm. We presented the derivation of the linear models, since they are required for further use in our linear State Estimation algorithm. For the derivation of nonlinear models of the PF feasibility algorithm, the reader is referred to [51].

2.1.2.3 Excitation of Adjoint Feasibility Circuits – Equivalent Circuit Programming

Interestingly, following the analogies in Table 1, one finds that the adjoint equivalent circuit of a slack bus model corresponds to a short on both the real and imaginary adjoint circuit. This implies that the adjoint equivalent circuit of this optimization problem is purely excited by the coupling terms between the primal (PF) circuit and its adjoint counterparts. This is expected, because an excited adjoint circuit corresponds to infeasibility (compare (50)).

Notably, the feasibility optimization problem is only one possible realization of optimization problems that can be solved as equivalent circuits (“equivalent circuit programming” (ECP) problems). Other notable examples are Optimal Power Flow [57], or State Estimation that will be discussed next. In these examples the excitation of adjoint circuits is defined by the negative gradient of the optimization problem’s objective function (compare [57]). In the special case of the feasibility problem this results in no additional excitation other than through coupling terms.
2.1.2.4 A Note on Convergence and Optimality

The PF feasibility optimization’s problem class is hard to solve without good knowledge of the system. However, by formulating the problem in terms of equivalent circuits one possesses this system knowledge and can apply the full toolbox of circuit simulation methods to find a solution to this coupled nonlinear equivalent circuit problem. In fact, [51] applies limiting techniques and extend the Tx-stepping method that is presented in Section 2.2.4.1 to adjoint circuits to robustly find solutions to this problem. An attempt for a mathematical proof of the optimality of the numerical solutions of this algorithm can also be found in [51].

2.1.3 State Estimation

AC Power Systems State Estimation (SE) can be formulated as a similar optimization problem that, with linear ECF measurement device models, results in a linear equivalent circuit problem. To derive the SE algorithm, we first present the equivalent circuit models of the two measurement devices prevalent in today’s power grids: the models for RTU (Remote Terminal Unit) and PMU (Phasor Measurement Unit). Subsequently, the optimization problem is defined. Importantly, the presented measurement models were first introduced in [58], and further refined in [59], which proposed a linear equivalent circuit model for RTU measurements.

2.1.3.1 PMU Measurement Model

A PMU measures voltage and current phasors. In the ECF, a voltage or current phasor corresponds to a pair of real and imaginary independent voltage or current sources, respectively. A source conductance $G_{PMU}$ is added to the PMU model to combine both types of measurements. Fig. 19 shows the real and imaginary equivalent circuits of the PMU ECF model for multiple current measurements. Without loss of generality the SE algorithm is further derived for a PMU with a single current measurement.

![Fig. 19 Equivalent Circuit model of a PMU.](image-url)
Interestingly, the current through the source conductances of this model is a measure of the PMU’s measurement errors. This becomes clear with the following thought experiment: Imagine a perfect PMU measurement. Injecting a measured current that corresponds to the exact injection of the real system into a perfect system model results in a response of this system that sets the PMU’s terminal voltage to precisely the measured voltage phasor. This scenario implies that the current through $G_{PMU}$ is zero. Now, any deviation from this perfect scenario results in a proportional non-zero current through $G_{PMU}$. This fact is utilized to formulate the optimization problem.

2.1.3.2 RTU Measurement Model

An RTU measures voltage $|V|$ and current magnitudes $|I|$ on a bus, as well as a power factor angle $\phi_{pf}$, which is the angle between the voltage and the current. Due to the lack of a global phase reference, these measurements are difficult to incorporate into this analysis. However, one can define an equivalent admittance from the three quantities that recovers the original system state in case of perfect measurements [60]. This equivalent admittance is defined by the measured quantities as

$$G_{RTU} + jB_{RTU} = \frac{|I|}{|V|} e^{i\phi_{pf}} = \frac{P_m + jQ_m}{|V|^2}$$  \hspace{1cm} (60)

However, measurements are never perfect. To account for measurement error, one utilizes the concept of feasibility sources that are further minimized to find a state estimate with minimal error. Fig. 20 shows the real and imaginary ECF circuits of an RTU including the additional feasibility sources.

2.1.3.3 AC State Estimation as an Optimization Problem

A state estimation problem is comprised of a set of RTU and PMU measurements from a Power System, which is modeled as a network of transmission lines, transformers, and shunts. Note that
both PMU and RTU measurement models, as well as all network constraints, are described by linear models within the ECF.

One can now define an optimization algorithm that minimizes these measurement errors

\[
\min_{\chi} F_e = \left\| \mathbf{I}_{\text{GPMU}}^{RJ} \right\|_2^2 + \left\| Y_{RTU} \mathbf{I}_{\text{RTU}}^{RJ} \right\|_2^2.
\]

s.t. \( Y_{GB} V = I_{\text{inj}} \)

where \( Y_{RTU} \) is a factor weighting an individual RTU model, \( V \) is the voltage state vector, \( Y_{GB} \) is the split bus admittance matrix that includes all terms defined by the linear PF network models, and \( I_{\text{inj}} \) is a vector of current injections. Its value is defined by the kind of measurement device on a bus. For an RTU measured bus, its contribution is defined as

\[
I_{\text{inj,RTU}}^R = G_{RTU} V^R - B_{RTU} V^I + I_{RTU}^R.
\]

\[
I_{\text{inj,RTU}}^I = G_{RTU} V^I + B_{RTU} V^R + I_{RTU}^I.
\]

Alternatively, the injected current for a PMU measured bus is defined as

\[
I_{\text{inj,PMU}}^{R,I} = I_{\text{PMU}}^{R,I} + I_{\text{GPMU}}^{R,I}.
\]

Importantly, \( I_{\text{GPMU}}^{R,I} \), the current through the PMU’s source conductance \( G_{PMU} \), is a dependent variable and can be expressed by formulating Ohm’s law

\[
I_{\text{GPMU}}^{R,I} = G_{PMU} (V_{\text{PMU}}^{R,I} - V_{\text{PMU}}^{R,I}).
\]

The objective function (61) together with the linear equality constraint defined by (62)-(66) is a convex optimization problem that can be solved using any optimization toolbox.

2.1.3.4 Adjoint Circuits of The Measurement Models

Alternatively, the adjoint circuit approach can be used to solve this problem as an equivalent circuit problem. Importantly, adjoint models of the PF feasibility optimization algorithm do not change for this algorithm. However, additional adjoint models of the RTU and PMU equivalent circuits are required. Therein, an RTUs adjoint circuit is trivially obtained by following the analogies of Table 1. It is depicted in Fig. 21.
As mentioned earlier, the excitation of the adjoint circuits in an ECP problem depends on the gradient information of its objective function [57]. To find the local excitation terms of the adjoint PMU circuit, one has to revert to optimization theory and find that a solution to the full problem (61)-(66) can be found by formulating its Lagrangian

$$\mathcal{L}(V, I_{RTU}^R, \lambda) = \|I_{PMU}^R\|^2_2 + \|Y_{RTU} I_{RTU}^R\|^2_2 + \lambda^T (Y_{GB} V - I_{inj})$$

(67)

Importantly, the current $I_{PMU}^R$ is not a parameter of the Lagrangian, since it is a dependent variable that is fully defined by the system’s state. To find an optimal solution to this convex problem one solves

$$\nabla \mathcal{L}(V, I_{RTU}^R, \lambda) = 0$$

(68)

Note that for an RTU measured bus solving (68) leads to the same result that was found for the feasibility algorithm. For a PMU measured bus, however, the gradient information of the objective function is found as additional terms that excite the adjoint PMU model. One identifies these terms by looking at the partial of the Lagrangian (68) to the voltage state vector $V$ on a PMU measured bus $i$

$$\frac{\partial \mathcal{L}(V_i, I_{RTU}^R, \lambda_i)}{\partial V_i^{R,i}} = 2 G_{PMU}^2 (V_i^{R,i} - V_{PMU}^{R,i}) + \lambda_i^{R,i} Y_{GB,i}^{R,i} = 0$$

(69)

where the term $2 G_{PMU}^2 (V_i^{R,i} - V_{PMU}^{R,i})$ defines the excitation of the adjoint PMU equivalent circuit. Mapped into equivalent circuit elements, this term corresponds to independent current sources governed by the known measurement voltages $V_{PMU}^{R,i}$, and voltage controlled current sources that are controlled by the primary bus voltages $V_i^{R,i}$, for both the real and imaginary adjoint circuits. In addition to the excitation terms, the term including the local bus admittance $Y_{GB,i}^{R,i}$ adds a conductance $G_{PMU}$ to the adjoint model. Notably, this part is equivalent to what one finds by deriving the PMU’s adjoint model from Table 1 by setting both independent sources to
zero, and translating the primary bus conductance $G_{PMU}$ to an adjoint bus admittance. The full adjoint PMU model including excitation and coupling terms can be seen in Fig. 22.

![Adjoint PMU equivalent circuit model](image)

Fig. 22 Adjoint PMU equivalent circuit model with $\kappa = 2G_{PMU}^2$, additionally indicating how primary element sources are neutralized in the PMU’s adjoint circuits models.

These models, together with the PF network models and their adjoint counterparts, now comprise a linear equivalent circuit problem including four coupled equivalent circuits. Importantly, this algorithm is an optimization algorithm for Power System State Estimation including PMU and RTU measurements that is fully linear.

This implies a guaranteed solution to this algorithm under any set of measurement data. In Section 4.3.1 we study how this algorithm compares to a traditional SE approach. Furthermore, we will leverage the comparatively low computational cost of this algorithm to propose a probabilistic approach to SE.

2.1.4 Power System Control Modeling

Power System operators employ multiple levels of controls to a Power System. The primary control is designed to automatically respond to imbalances in a system within a few seconds. This control (droop-control) is a fully proportional frequency control that stabilizes the system frequency in case of a disturbance but does not recover a system to its nominal frequency. A secondary control, the automatic generation control (AGC), recovers system frequency by adapting generation of participating generators. These generators receive AGC control signals in a period of around a minute. An electricity market or an otherwise created generation schedule is commonly referred to as tertiary control.

As primary and secondary control schemes are frequency dependent, they are not trivially included into the PF analysis that is agnostic of the system frequency. However, modeling these controls is important to improve on the slack-bus model. Different approaches to include these
controls into PF studies have been proposed. We discuss an approach for a crude AGC model for probabilistic samples in Section 4.2.2.3.

In parallel to these controls, a power system’s voltage profile is stabilized by tap changing and phase shifting transformers, switched shunts, and modern devices such as FACTS. These controls are commonly included in PF simulation software including the SUGAR C++ implementation. However, they are modeled as outer loop controls that adjust control variables only after a full NR solution is obtained. Since the controlled variable generally responds in a nonlinear fashion to the control signal, modeling controls this way is known to lead to numerical instabilities. An alternative approach that models these controls as continuous functions can be found in [41].

2.2 Circuit Simulation Methods

We have rederived ECF models for the PF, SE, and PF feasibility algorithms. Now, we discuss methods and techniques that are required to solve these equivalent circuits. We start with Modified Nodal Analysis that enables casting an ECF problem into a sparse matrix problem. Then we discuss sparse matrix solution algorithms. Finally, we introduce convergence aiding algorithms such as limiting techniques and homotopy methods.

2.2.1 Modified Nodal Analysis

Modified Nodal Analysis (MNA) enables one to cast equivalent circuit problems into sparse matrix problems that can further be solved using sparse matrix solution techniques [61]. It is an extension of Nodal Analysis that includes independent voltage sources and controlled sources. Nodal Analysis formulates a linear circuit problem in terms of KCL equations on each node

\[ Y V = J \]  \hspace{2cm} (70)

with a nodal admittance matrix \( Y \), nodal independent current injections \( J \), and the nodal voltage vector \( V \), which is unknown. Now, each node is assigned a unique ascending number for identification. This number corresponds to a row in the system of KCL equations, which enables us to build the system matrix effectively in a computer program.

Furthermore, one realizes that there is specific set of entries with the same shape that correspond to each circuit element’s contribution to the system (we call this a “stamp”).
Importantly all element stamps together sum to the full legal system matrix. Hence, in an efficient automized fashion one can stamp each element successively into a system matrix (and the current injection vector \( J \)) creating a full system of equations.

For example, a stamp of a conductance \( G \) between nodes \( i \) and \( j \) can be found by formulating Ohm’s law

\[
G(V_i - V_j) = 0
\]

and adding its nodal branch current contributions into the matrix \( Y \)

\[
\begin{bmatrix}
\cdots & G & \cdots \\
\cdots & -G & \cdots \\
\end{bmatrix}
\begin{bmatrix}
V_i \\
V_j \\
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
\end{bmatrix}
\]

Stamping a single element, or a subset of elements that do not comprise a circuit does not lead to legal equations. However, after summing all stamps of each circuit element, a fully legal problem is obtained. Importantly, stamping each node including the ground one arrives at a linear dependent set of equations (*indefinite equations*). This is because no reference potential has yet been chosen. Commonly, the ground node is used as this reference; its contribution is omitted in the system of equations (*definite eq.*).

In Nodal Analysis, difficulty arises when independent voltage sources are added to the circuits. MNA solves this problem. It introduces an auxiliary node (a state variable) that represents the current through the voltage source. The voltage constraint is further stamped as the obligatory additional equation \([61]\).

A final set of circuit elements, imperative to model equivalent circuit problems, are controlled sources. With the exception of voltage controlled current sources that fit naturally into the nodal equations, controlled sources need the handling of voltage sources as a prerequisite. These controlled sources are: current controlled current sources that use a zero-valued voltage source to measure the controlling current, current controlled voltage sources, and voltage-controlled voltage sources.

With these elements and their accompanying stamps, we can now build the linear system of equations that is further solved for each NR step in our nonlinear problem solver. The linearization of the nonlinear elements is explained as part of the derivation of the nonlinear models that are presented for PF in Section 2.1.1.1 and for SE in Section 2.1.3.
2.2.2 SOLVING LINEAR SPARSE MATRICES

After creating a sparse system matrix that corresponds to a linearized ECF problem using MNA, we use sparse linear solvers to find the voltage state vector of one NR iteration of the problem. While we will focus on direct solution methods for linear sparse matrices, it is worth mentioning that iterative solution algorithms exist. Iterative methods in connection with good preconditioners can be a good alternative for special cases where matrices structures (and conditioning) are well known. This is especially true for extremely big systems of equations (due to the reduced memory requirements of iterative solvers) or for parallelized algorithms. Direct solvers, on the other hand, are very robust and guarantee a precise solution for a big class of matrix structures.

The most well-known direct solution algorithm is Gaussian elimination, which is the classical algorithm to solve matrices. The most used in circuit simulators, however, is the similar method of LU factorization. LU decomposition factorizes a linear system matrix

\[ Y = LU \]  \hspace{1cm} (73)

into a lower L and upper U triangular matrix, which enables solving a linear system in two steps

\[ x = UV \]  \hspace{1cm} (74)
\[ Lx = J \]  \hspace{1cm} (75)

LU decomposition has the same complexity of Gaussian elimination but delivers two triangular matrices that (if necessary) can be re-used to solve a single system with a set of different right-hand side vectors. Circuit Simulators leverage this property extensively [61].

In the special case of a Hermitian positive-definite matrix, the Cholesky factorization can be employed. It factorizes a system matrix A

\[ A = LL^* \]  \hspace{1cm} (76)

into a lower triangular matrix L that, together with L* (the conjugate transpose of L), are used to solve this special system of equations. Within our approach Cholesky-decomposition is employed in the Monte Carlo framework for creating correlated random samples.
2.2.3 Limiting

Solving nonlinear systems of equations using the Newton-Raphson algorithm commonly requires convergence aiding techniques in addition to a good sparse numerical solver. Convergence aiding methods range from very simple but surprisingly powerful approaches to sophisticated algorithms that guarantee finding a solution, if one exists. The class of limiting methods fall into the first category.

In contrast to simple damping of the solution vector during NR-iterations, which has been widely studied to improve convergence for PF, limiting methods only adapt a select set of variables between iterations. Voltage limiting does that by setting an upper bound on the possible change of a variable between two iterations. If a variable exceeds that limit, it is limited to only move the maximal allowable distance. Variables that do not exceed this limit remain unchanged. This technique improves convergence of PF dramatically.

Fig. 23 shows how limiting methods improve the convergence of a 2383-bus testcase starting from different Q initial guesses. We see that the solver does not reach convergence until the 100th iteration for some initial guesses without applying limiting. When limiting is applied, the solver reached convergences after 5-6 iterations for all initials.
2.2.4 Homotopy Methods

Limiting methods are not always sufficient to find a nonlinear PF problem’s solution. When this is the case, homotopy methods can be employed. A homotopy, which is a function that smoothly alters the original problem to a problem that is simpler to solve, is a powerful approach for solving complex nonlinear problems. After obtaining a solution to this system, they robustly guide a nonlinear solver back to the original problem’s solution. Mathematically one can define a homotopy $\mathcal{H}(x, \lambda)$ by a weighted sum of two nonlinear functions

$$
\mathcal{H}(x, \lambda) = (1 - \lambda)F(x) + \lambda G(x)
$$

(77)

where the original nonlinear problem is defined by $F$. This problem is initially altered to a problem $G(x)$ that is easy to solve, the homotopy factor $\lambda$ defines a smooth transition between the original and the altered problem. In (77) $\lambda$ ranges from 0 to 1.

2.2.4.1 Tx Stepping

The Tx-stepping method is a homotopy method that was specifically created to robustly solve PF problems [63]. It uses a set of a PF problem’s defining features to create a homotopy method that robustly recovers a PF problem’s solution without the knowledge of a good initial guess.

One of these defining features is the slack bus assumption that sets the system’s reference angle and has a defined voltage magnitude setpoint. The PF solution in the vicinity of a slack bus will very likely be close to this slack bus angle and magnitude. Hence, when reducing the electrical distance of buses that are far away from a slack bus, the complex voltage solution tends towards the complex voltage defined by the slack bus.

Utilizing this feature, the Tx-stepping method first reduces the electrical distances in a system to bring the system’s state closer to the slack bus state. Then it aims to find a solution to this relaxed system by using the complex slack bus voltage as initial guess for all buses.
Fig. 24 Five bus example to illustrate the Tx-stepping method.

Fig. 24 illustrates the Tx-stepping method’s concept on a five-bus system. In order to reduce the electrical distances in the system we add auxiliary admittances at every bus. After finding an initial solution, this admittance is iteratively relaxed until the original problem is reached.

We further discuss the auxiliary elements defined in [63] that reduce the electrical distance during Tx-stepping. The auxiliary elements of a transmission line are trivially obtained by including an auxiliary admittance, which is a function of the homotopy factor $\lambda$, in parallel to the series element of the original transmission line model. The adapted model can be seen in Fig. 25.

![Diagram of transmission line series element changes as a function of Tx-stepping method system relaxation factor $\lambda$.]

Defining the changes for transformers is a bit more involved. In addition to series losses that need to be reduced, a transformer additionally defines a tap ratio and possibly a phase shift. Both parameters also need to be relaxed for the Tx-stepping method. The goal for the initial solution is to be as close as possible to the slack bus voltages, hence the initial tap ratio of any transformer should be one. Similarly, all transformer phase shifts should be set to zero when the Tx-stepping method is initialized.
The transformer model adapted for Tx-stepping can be seen in Fig. 26. In addition to the auxiliary admittance, it also defines smooth functions to adapt the transformers tap ratio \( \hat{t}_r \) and phase angle \( \hat{\theta} \) dependencies on the homotopy factor \( \lambda \)

\[
\hat{t}_r = t_r + \lambda (1 - t_r) \\
\hat{\theta} = \theta - \lambda \theta
\]

where the tap ratio \( t_r \) and the phase shift angle \( \theta \) are the transformer’s original parameters.

The final element of Tx-stepping is required when a PS model includes generator models that control voltage magnitudes of buses different from their own, which worsens convergence difficulties. These difficulties stem from hard to find required reactive powers \( Q_o \) that control the remote voltage magnitudes. To counter these difficulties, additional artificial transmission lines are added during Tx-stepping. These lines connect a generator’s bus directly to the bus it controls, which greatly simplifies the problem. During the Tx-stepping algorithm, this auxiliary transmission line is relaxed away to find the original problem’s solution when Tx-stepping finalizes. This scenario is depicted for a four-bus system in Fig. 27.
Additional pitfalls with remote controlled generator voltages exist. For example, not every bus can be controlled by any generator. A clear control path that does not interfere with other controls is needed. As we will see, our approach tackles this problem by a pre-simulation check for legal control paths using a system graph.

It has been shown that the Tx-stepping algorithm is able to robustly solve ill-conditioned as well as huge real-world transmission systems including their contingencies. Additionally, it has been successfully applied to distribution systems [64].

Table 2 Comparison of Tx-stepping to standard tool for PF cases.

<table>
<thead>
<tr>
<th>Case Name</th>
<th># Nodes</th>
<th>Standard Tool</th>
<th>SUGAR Tx-stepping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># Converge</td>
<td># Diverge</td>
</tr>
<tr>
<td>2017SLL</td>
<td>80778</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Eastern1</td>
<td>76228</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>2021SUM</td>
<td>81904</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 2 shows convergence properties of SUGAR using Tx-stepping for three different Eastern Interconnection planning cases and 15 different initial conditions using voltage angles in the range of $[-40^\circ,40^\circ]$ and voltage magnitudes of [.9,1.1]. Since Tx-stepping is agnostic to initial conditions, the authors of [64] were able to robustly converge all 15 samples. Importantly, the standard tool was not able to converge complex cases like these from any of these initial conditions.

Table 3 Comparison of Tx-stepping with standard tool for contingency cases.

<table>
<thead>
<tr>
<th>Case Name</th>
<th># Nodes</th>
<th>Contingency</th>
<th>Standard Tool</th>
<th>SUGAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>76228</td>
<td>2 Generators</td>
<td>Diverge</td>
<td>Converge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Gen. + 2 Branches</td>
<td>Diverge</td>
<td>Converge</td>
</tr>
<tr>
<td>Case 2</td>
<td>78201</td>
<td>2 Generators</td>
<td>Diverge</td>
<td>Converge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Gen. + 2 Branches</td>
<td>Diverge</td>
<td>Converge</td>
</tr>
</tbody>
</table>

Table 3 compares the Tx-stepping algorithm with a standard tool for contingencies of real Eastern Interconnection planning cases. Here, SUGAR using Tx-stepping was able to converge for all 4 studied cases, whereas the standard tool did not find the solution to any of these cases.
2.3 Probabilistic Algorithms


Our focus is on only Load Flow or PF algorithms. Additionally, this work touches on a probabilistic State Estimation, which is not mentioned in [7] and has generally not been widely studied. The probabilistic PF algorithms discussed in [7] are exclusively analytical. The author distinguishes between a Stochastic Load Flow (SLF) and Probabilistic Load Flow (PLF) approaches. Interestingly, the SLF algorithm builds upon the assumption of Gaussian random variables and formulates a load flow problem as a weighted least squares algorithm. This is similar to the classical State Estimation algorithm introduced by Schweppe [21]. The introduced PLF approaches are linearized analytical approaches, as first proposed in [6]. They include linear approximations around the solution point of the PF equations, and linearizations of the PF equations (DC Load Flow and decoupled Load Flow).

2.3.1 Analytical Methods

Analytical methods to calculate a probabilistic PF in addition to the ones that were discussed in [7] can be categorized into algorithms that use a Cumulant Method [65]–[68], which can mathematically be made equivalent to the Fast Fourier Transform [69]. These algorithms assume that input probability density functions can be described by a finite series of their moments to circumvent an otherwise required convolution. Importantly, convolution based formulations exist as well [70].

Other commonly applied analytical methods are based on Point Estimates. These methods concentrate on the first few moments of a problem’s probabilistic inputs to infer information about probabilistic output variables [71]–[75]. Less commonly applied methods utilize fuzzy logic [76], Nataf transformation (e.g. [77], [78]) that is only applicable for multivariate Gaussian distribution [79], and interval arithmetic [80] that can be viewed as a probabilistic method.
Analytic algorithms have in common that they require a linear formulation of the base problem or a linearization around a certain point [81]. Additionally, restricting assumptions on correlations between variables to total independence or linear correlations apply. Another important drawback in the context of probabilistic PF is that analytical approaches are by their design not able to include network uncertainties.

2.3.2 Numerical Methods

Numerical methods for probabilistic algorithms have higher computational cost compared to their analytical counterparts. However, they have the big advantage of generality. Due to the recent exponential increase in computing power and the development of more efficient and robust PF solvers, numerical methods are increasingly likely to be a good and viable option for probabilistic studies.

Numerical methods can be categorized into Simple Random Sampling (SRS) Monte Carlo (MC) methods, that are the most general, but rely on good pseudo random number generators and have comparatively slow convergence. However, many times they are the only viable option to deliver reliable results due to their generality [82]. Examples for SRS MC methods applied to power flow studies can be found in [83]–[85].

Variance Reduction techniques or Quasi-MC methods that build on them are attempts to improve MC convergence under a set of assumptions [38]. While they are considered to be probabilistic methods, their sampling strategy is based on deterministic algorithms. Commonly applied to PS are Latin Hypercube Sampling [86]–[90] and Latin Supercube Sampling techniques [91]. Also applicable is the stratified sampling [92], or importance sampling technique [93]. The latter is especially useful in the case of heightened interest in a certain probability region (e.g. rare events).

2.3.2.1 Pseudo Random Number Generators

The foundation of good Monte Carlo results is a good random number generator (RNG). While approaches for physical RNGs that produce truly random numbers exist, Pseudo Random Number Generators (PRNGs), which are deterministic algorithms producing a stream of numbers that appear to be random, are preferred for our purpose, if only to be able to reproduce results
when necessary. Other properties that speak against the use of physical RNGs are their comparatively slow speed, possible biases, and their availability [94].

The random “appearance" of the numbers a PRNG produces is sufficient for MC simulations, if the randomness is “good" enough for the results to be “similar enough" to what would have been obtained if the numbers were truly random [94]. How can the goodness of a PRNG be measured in more concrete terms, such that one can rely on it during a simulation? The goodness of a PRNG can measured by different statistical test-suites that attempt to break the randomness of a generator[94], [95]. It is important to use generators that have been extensively tested and that have an established theory behind their random number generation scheme, because the number of potential pitfalls designing such generators is high. A prominent example is the RANDU generator that had severe defects due to abnormal correlations [94].

To pick a good PRNG for our purpose, we need to define a few more important parameters. A good PRNG should have a period that is long enough to produce high quality pseudo random numbers during a simulation, its output should be distributed as closely to a uniform-distribution as possible, it should be fast enough that it does not become the bottleneck of simulation, and ideally it should be able to produce parallel random streams that are extremely useful when evaluating MC samples in parallel. Other PRNG properties that we are not as concerned about are a PRNGs code-size, seek-ability (ability to skip forward), or its cryptographical qualities [95].

Many successful classes of PRNGs use linear recurrences over a cyclic state space (a ring \(\mathbb{Z}_m\)). If \(m > 2\), these algorithms are called Linear Congruential Generators (LCGs). Their internal state follows a state transition function

\[
\tau(x) = (ax + c) \mod m
\]

(80)

Where \(x\) is the state of the LCG, \(a \in \mathbb{Z}\setminus\{0\}\) is the “multiplier”, and \(c \in \mathbb{Z}\setminus\{0\}\) is called the increment. The output of an LCG is then

\[
\xi(x) = x/m
\]

(81)

LCGs have a maximal period of \(m\). Hence, they should not be used for MC simulations. However, they can be very useful as a part of combined generators [94].

One way of combining LCGs is by creating a Multiple Recursive Generator (MRG) that defines a transition function over \(\mathbb{Z}_m^k\). The maximal period of MRGs is \((m^k - 1)\). Alternatively, multiple
LCGs can be combined, by running them in parallel and further combining their states to get a single random output. Interestingly, it can be shown that both approaches are equivalent.

A special class of generators are LCGs over \( \mathbb{Z}_2 \), which are called linear feedback shift registers (LFSR), which can be generalized to (Twisted) Generalized Linear Feedback Shift Registers. The most widely used PRNG, the “Mersenne-Twister” is such an algorithm. It has a very long period of \( (2^{19937} - 1) \) [94]. However, it has a big state that requires quite a bit of memory and is comparatively slow compared to more modern approaches [95].

The family of Permuted Congruential Generators (PCGs) is such an approach. It combines speed with long periods and “excellent” statistical properties. Additionally, it provides multiple parallel streams of random numbers, which is another advantage over the “Mersenne-Twister.” The idea behind this group of generators is to combine well understood and fast medium-quality LCGs with permutation functions at their output [95]. This class of generators will be used for our probabilistic algorithm.
3 SUGAR C++ IMPLEMENTATION

The theoretical background of an Equivalent Circuit Formulation based solver is now established. Next, we present details of the implementation of an ECF based solver. We call implementations of equivalent circuit-based solvers SUGAR: *Simulation with Unified Grid Analysis and Renewables* [96].

This implementation was designed with the following goals in mind:

- Cleanly designed and written code for reusability and extendibility
- Efficient, high-performing code with a focus on readability while allowing code optimizations in the inner loops
- Parallelizable code to support the Monte Carlo analyses that form the primary objective of this thesis

3.1 SUGAR C++ (AN ECF BASED SOLVER)

The readability, reusability, and most importantly, efficiency, of a software such as SUGAR depends heavily on its architecture and design. The better a software is broken up into operative sub-components that interact via well-defined interfaces, the better the software will operate. Additionally, continuous future development is only possible when these components and their interfaces are logical, easily understood, and well documented. Furthermore, it is not necessary to reinvent the wheel. As many components as possible should be part of high-performing, widely used, and well tested third-party libraries.

3.1.1 CONCEPTS AND ABSTRACTIONS

Before describing the software design in depth, we discuss abstractions and concepts that comprise an ECF based solver from an algorithm perspective. This enables us to better understand the required software abstractions, their expected performance, and their possible restrictions.
Fig. 28 Concepts hierarchy of an ECF based solver.

Fig. 28 gives an overview of algorithms that are employed to solve a PF problem using the ECF.

The lowest level algorithm within this hierarchy is a linear sparse matrix solution algorithm. As discussed in 2.2.2, linear sparse matrix solvers can be classified into direct and iterative solvers. Direct sparse matrix solvers are a well-developed field [97]. Sparse matrix solution methods include the Cholesky-, QR- and, LU-factorization algorithms. LU factorization is the oldest solution method and represents the default algorithm for general square matrices [98]. LU factorization algorithms can be categorized into left-looking, right-looking, and multi-frontal algorithms. The challenge of any LU factorization algorithm is to effectively determine permutations p and q of a matrix $A_{n \times n}$ such that the factorization $A(p, q)$ is reasonably stable and L and U are close to optimally sparse [97]. This requires an algorithm to find a tradeoff between maximized matrix conditioning and minimal matrix fill-ins.

It is difficult to determine exact time and memory requirements as well as scaling properties of these algorithms as they highly depend on a problem’s properties, such as matrix-sparsity, matrix-structure, etc. However, [98] describes the performance of right-looking LU factorization algorithms to take $O(n + |A| + f)$ time, where $n$ is the matrix size, $|A|$ is the number of non-zeros in $A$ and $f$ is the number of floating point operations. With a few exceptions, this reduces to $O(f)$ [98]. To determine memory complexity of these algorithms, we will take an empirical approach and study memory requirements of the KLU [99] algorithm of different problems in the results section.

Both of the evaluated LU factorization algorithms (UMFPACK [100] and the KLU [99]) are divided into two stages: a symbolic factorization stage that only requires the matrix structure as
input, and a numeric factorization that finds numerical results to a given problem. This enables us to reuse symbolic factorizations during the next-higher algorithm in this hierarchy (the Newton-Raphson algorithm), as NR solves multiple sparse matrix problems that only differ numerically.

The Newton-Raphson algorithm is a numerical solution algorithm for nonlinear problems. It requires an initial guess for a problem’s solution and employs a gradient based method to iteratively converge to a solution of a nonlinear system of equations. Barring some exceptions, a good initial guess leads to quadratic convergence of this algorithm.

To solve a general system of nonlinear equations the NR algorithm requires partial differentials of all dependent system variables to each other (the Jacobian). ECF based solvers hierarchically build a problem’s Jacobian in terms of equivalent circuit analogies using equivalent circuit-based models and MNA. Therein, each model symbolically stores equations to determine its corresponding Jacobian entries for each NR iteration. This enables a modularized and efficient algorithm design.

The highest-level algorithms within an ECF solver are limiting or homotopy methods that aim to improve convergence of the nonlinear solver. Herein, limiting methods require a solution vector data-pass for each NR-iteration. Hence, their implementation is performance critical, making them perfect candidates for early code optimizations. Homotopy methods on the other hand, alter a system after an NR loop finalizes. Their implementation is less critical to inner-loop solver performance, however, designing these algorithms to require minimal steps is critical to the overall performance of an ECF solver.

3.1.2 Module and Interface Design

The SUGAR C++ implementation is split into three core components: A nonlinear equivalent circuit solver infrastructure, a set of solver control algorithms, and a set of data structures. These data structures are the parsed power flow models, their ECF representations, and structures that assemble the element models to models of power systems. Additionally, a set of utility functions and packages fulfill I/O tasks or tasks that are commonly used in many parts of the code.
Fig. 29 gives an overview of the solver’s components. The “ECF solver” component comprises of three objects: the SparseSolver that serves as an interface to sparse matrix solver libraries and is utilized to solve a single linearized equivalent circuit problem; the NRSolver that solves one Newton-Raphson loop at a time and is controlled by the SugarSolver, which in turn enables the integration of control algorithms via a single well-defined interface; the SugarSolver that relays the Power System model (PSModels) to the NRSolver and keeps track of solutions.

The data pipeline classes enable Power System models to be instantiated from and written to different file types. Parsers for PSSE .raw file formats or MatPower files exist. The PowerFlowData class acts as a container for this input data. From a PowerFlowData object a PSModels object is instantiated. The PSModels class encapsulates all ECF models and provides interfaces for solver and control algorithms. All of this is enabled by a set of utilities that serve either I/O purposes, such as a library to log solver states and progress to the console or a file, a command line input library, or a json parser to that enables solver inputs via a json based input file. Other utilities include sparse matrix and vector representation classes and helper functions that predefine MNA stamps.
3.1.3 ECF SOLVER

The “ECF solver” is the core of this implementation. Here, we focus on a clean optimized implementation of the solver classes that define interfaces for control but do not implement controls themselves. The ECF solver consists of three stages.

3.1.3.1 SparseSolver

The innermost stage is the SparseSolver class that defines an interface for sparse matrix solver libraries. Its purpose is straightforward. It receives a sparse matrix problem defined by a sparse system matrix and a right-hand side vector, relays this problem to a sparse matrix solution library, and returns the solution vector to the next stage, the NR Solver. The sparse solver utilizes either the UMFPack [100] or KLU [99] library to solve sparse matrix problems. Both are part of the SuiteSparse package.

3.1.3.2 Matrix and Vector Representations

Sparse matrix solvers require specific matrix formats that commonly compress storage. Many different formats with specific advantages and disadvantages exist. Generally, more complex compression schemes make it more difficult to create and modify a matrix. This is why our implementation splits creating a system matrix into two stages with two different matrix formats.

![Diagram of matrix and vector representations](image)

Fig. 30 The two SparseMatrix classes share a common interface within SUGAR C++.

Fig. 30 shows the common interface of the two available sparse matrix classes of the SUGAR C++ implementation. Herein, the TripletMatrix stores each matrix entry as a triplet of row index, column index, and matrix entry. Its underlying data structure is a C++ map. This enables stamping while the matrix structure is still unknown. The CSSparseMatrix structure stores a sparse matrix in either compressed row or compressed column format. Both formats are used as inputs to the
sparse matrix packages. However, to handle them efficiently, the matrix structure needs to be known in advance. Looking at the inner loop from a data flow perspective, clarifies how both matrix types are used.

Fig. 31 Sparse matrix conversion during Newton-Raphson loop.

Fig. 31 shows the Newton-Raphson loop from a matrix manipulation perspective. First, a matrix is filled with non-changing entries before entering the loop, where the complete structure of the matrix is still unknown. The trick to define the full system matrix structure is to add non-changing entries as zero elements and later fill them during the loop. After the full matrix structure is defined and the slow and storage heavy *TripletMatrix* is converted to a compressed representation. The corresponding *CSSparseMatrix* that is further filled with nonlinear elements and passed to the solver library. The initially converted matrix is stored, so that subsequent iterations are able to reuse only efficiently stored matrices.

It is worth noting that a similar interface exists for vector representations that include a *PairArray*, which is the pendant to the *TripletMatrix*, and a *DenseArray* that is used after conversion. However, while this separation simplifies our algorithm, splitting vector representations is far less critical. Vectors of *SparseMatrix* problems are generally dense and much smaller in size than their corresponding matrices. For simplicity of the following discussion, we will refer to sparse matrices and vectors without mentioning their exact representation.
3.1.3.3 **NRSolver Class**

We now look at the nonlinear solver from an algorithm perspective. The *NRSolver* class implements the inner NR-loop, which solves a nonlinear ECF problem using the Newton-Raphson algorithm. The *NRSolver* implements the innermost loop. Hence, it is crucial in defining the overall performance the solver. We designed this inner loop to be insulated from most control algorithms. This enables an efficient, clean implementation that optimizes performance.

![Activity diagram of the NRSolver’s inner loop implementation.](image)

Fig. 32 Activity diagram of the *NRSolver*’s inner loop implementation.

Fig. 32 shows the activity diagram of the *NRSolver*’s inner loop implementation. The algorithm is started by a function call from the *SugarSolver*. It stamps the full system matrix in three distinct stages. First, the linear components are stamped and stored for reuse. Next, the control
algorithm dependent matrix entries are stamped. Further, both sub matrices are composed to a non-changing matrix that remains unchanged within a NR loop. Hence, this matrix is reused every time the loop reaches its top. After the innermost loop is entered, all nonlinear elements are stamped on top of the composed non-changing matrix, defining the full matrix problem. Note that a control algorithm’s influence on this loop so far is limited to the `stampControl()` method. The full matrix problem is then solved via a function call to the `SparseSolver` that, after a successful solve, returns a solution vector. This solution vector is further postprocessed. Postprocessing algorithms require the current and the previous solution vectors. They include limiting and damping methods and the calculation of the difference between the solutions to evaluate convergence.

The efficiency of the limiting, damping, and difference algorithms is vitally important to the total solver performance as they require looping through data inside the innermost loop. Hence, all algorithms are implemented inside the `NRSolver`. This enables their implementation within a single data pass, which keeps them as high-performing as possible. The postprocessing algorithms are at the bottom of the inner loop. Now, three reasons to exit the NR loop exist:

- The difference between solutions is smaller than the tolerance: the problem converged.
- The maximum number of iterations is reached.
- The system diverged and the outside control algorithm (via a function of the `ContinuationMethod` interface) decides to stop.

Finally, the `NRSolver` returns a success/fail flag and the final number of iterations. In case of convergence, the solution vector is further extracted and stored by the `SugarSolver` class. Importantly, a single NR loop is commonly not sufficient to solve a full PF problem. Usually, control actions that are modeled in outer loops (e.g. Generator Q-limiting) are necessary. To improve the performance of the solver we aim to reuse as much data as possible. Therefore, when the `SugarSolver` invokes the `NRSolver` loop again, the initial linear matrix that was created during the prior loop is reused and the inner loop starts with stamping the control dependent matrix parts.
3.1.3.4 SugarSolver

The SugarSolver defines how control algorithms interact with the NRSolver by implementing an outer control loop structure that includes call-back functions for generic control algorithms. This design enables insulation of potentially complex control implementations from the solver that using this design remains clean, readable, and efficient. Additionally, this design choice defines a rigid interface for control algorithms, which greatly simplifies their implementation.

![UML class diagram of the SugarSolver.](image)

Fig. 33 UML class diagram of the SugarSolver.

Fig. 33 shows a class diagram of the SugarSolver including its most important components. The SugarSolver controls the execution of inner Newton-Raphson loops. It initializes these loops with initial guesses. Hence it stores possible initial guesses as member variables and keeps track of sub-problem solutions that can further be used for future initial guesses of ensuing inner loops.

Additionally, the SugarSolver references a PSModels object that contains all ECF models for stamping. As discussed in Section 2.2.1, prior to stamping, the elements have to be assigned MNA matrix row indices. The SugarSolver initializes MNA node indices within the models via the set_model_node_indices() function and further stores this node assignment in a NodeMap class that maps each PF node to corresponding matrix rows. A final crucial member of the SugarSolver class is a pointer to a control algorithm that is accessed via a ContinuationMethod interface.
After receiving a PSModels structure, setting its MNA node assignments, and receiving a control algorithm, a simulation is started via the SugarSolver’s `run()` function. When the simulation finalizes, the `run()` function returns a structure of statistics of the simulation run including number of total Newton-Raphson iterations, number of control loop iterations, and success or failure of the simulation. Finally, the simulation’s result can be obtained by calling the `get_solution()` function.

Fig. 34 Activity diagram of the outer control loop implemented by the SugarSolver.

Fig. 34 illustrates the structure of the outer control loop that is invoked by the `run()` function call. Control actions take place as part of five call back functions that invoke the:
• initialization of the control algorithm
• preparation for the next inner-loop (NR-loop)
• control action after the NR-loop succeeded
• alternatively, the control action after the NR-loop failed
• control algorithm finalization (success, failure?)

For a clearer understanding of this outer control loop, we walk through one iteration from the control algorithm’s perspective: The controller is first initialized by calling the `initialize()` function. This enables the controller to access data structures that are necessary to determine and fulfill its function. These are the solver state including the previous solutions, and the models that are altered during control actions. After initialization, the control loop is started. A first control action is possible during the `prepare()` call. Subsequently, the inner simulation loop is invoked. After its completion, the controller is notified of the inner-loop’s success or failure. Additionally, the stored solutions are updated by the `SugarSolver`: In the case of a successful inner loop, the new solution is stored as possible initial guess for the future. If the inner loop does not converge, the `SugarSolver` notifies the controller of the failure and restores the last obtained solution as the initial guess for the next inner loop. The outer control loop finalizes when the control algorithm concludes. This is determined by a call to the control algorithm’s `finished()` function.

It is important to highlight that the `SugarSolver` does not rely on the control algorithm to store solutions and use them as future initial guesses. If this was not the case, control algorithms would have to implement this on their own. This would lead to multiple copies of the same code, which is a common source of software bugs. Importantly, this design choice restricts control algorithms to take only a single step back (to the previous solution) in the case of non-convergence, which in practice has proven not to be a limitation.

3.1.3.5 Outer Loop Controls

Next we discuss the implementation of different outer-loop control algorithms. The interface of a control algorithm is defined by the `ContinuationMethod` class.
Fig. 35 Class diagram of the ContinuationMethod control interface.

Fig. 35 shows a class diagram of this interface. Each control algorithm implements a specialization of the ContinuationMethod’s member functions. These functions are then called by the SugarSolver as visitors within the outer control loop of Fig. 34. This structure enables the implementation of two kinds of algorithm: Homotopy or continuation methods, and outer loop controls, such as generator Q-limiting, transformer tap adjustments, or switched shunt adjustments.

3.1.3.5.1 Homotopy Methods implemented as ContinuationMethod Classes

First, we discuss the implementation of homotopy methods. Currently implemented methods within SUGAR C++ are the Tx-stepping method and the probabilistic continuation method that was designed specifically for Monte Carlo Simulations (Section 4.1.2). The Tx-stepping method is implemented to include adaptive step sizes, which greatly speeds up convergence. Its parameters are tuned to solve representations of real Eastern Interconnection PF cases and contingencies thereof as efficiently and robustly as possible, without a good initial guess.
Fig. 36 Core algorithm of the adaptive Tx-stepping implementation.

Fig. 36 shows how the sub-algorithms of the Tx-stepping method are implemented as ContinuationMethod visitor functions. Here, \( \lambda \) is the Tx-stepping homotopy factor (see Section 2.2.4.1). It ranges from 1 to 0, where 1 represents a completely relaxed system and 0 represents the original problem. The adaptive step size implementation of Tx-stepping initializes \( \lambda \) to be 0.5, which represents a 50% relaxed state. Importantly, this is after a solution of the original problem \( (\lambda = 0) \) was attempted.

Next, as defined by the outer control loop algorithm (Fig. 34), a solution attempt with \( \lambda \) at 0.5 is started. In case we find a solution to this 50% relaxed problem, the visitor_nr_loop_succeeded() function is invoked. It first stores the value of \( \lambda \) that now corresponds to a solved system with a solution that can be used as initial guess for the next step. As mentioned before, only the homotopy factor \( \lambda \) needs to be stored. The SugarSolver class automatically stores the solution vector during the outer control loop algorithm. Further, the visitor_nr_loop_succeeded() function increases \( \lambda \)'s step size by a factor that was optimized for solution speed and then increases \( \lambda \) by the updated step size.
In the case of non-convergence of the inner loop, the `visitor_nr_loop_failed()` function is called. Here, a more complex set of possible control actions is implemented. If a prior solution exists, \( \lambda \) is restored to the homotopy factor of this solution. Furthermore, the step size of \( \lambda \) is decreased and \( \lambda \) is updated to take a smaller step in the next solution attempt.

If no prior solution exists, other Tx-stepping parameters are changed in order to find a first solution as quickly as possible. Initially, the relaxation (admittance) of the auxiliary remote-controlled generator lines is adapted. If this fails, \( \lambda \) is scaled towards a more relaxed system with decreasing step size. At each of the steps of \( \lambda \) the auxiliary line relaxation is reset to its original, causing the relaxation to be repeated for each step of \( \lambda \). When \( \lambda \) is 1, the system is completely relaxed in terms of the homotopy factor. Then, the final parameters that can be adapted to find an initial solution are the homotopy admittances of each transmission line and transformer in the system. The number of possible parameter-combinations for initial solution attempts provides a sufficient search space that almost guarantees finding a relaxation that leads to an initial solution. However, for most problems an initial solution is found very quickly without requiring parameter changes other than the changes of \( \lambda \).

Notably, the structure of the outer control loop with its predefined visitor-functions simplifies this implementation of Tx-stepping to simple sequences of logical steps. Similarly, other continuation methods are implemented. One example is the probabilistic sample continuation that is described in Section 4.1.2.

**Device Controls implemented as Continuation Methods**

Outer loop device controls, such as the generator Q-limiting algorithm, controls of transformer taps, and controls of discretely switched shunts, are implemented using the `ContinuationMethod` interface as well. However, it should not go unmentioned that while these outer discrete control loops are a de-facto standard in current commercial PF software, they are problematic. Modeling discrete discontinuities in this way can cause oscillations and lead to non-convergence of a solver.

**CONTROL EXAMPLE: Generator Q-limiting**

We will discuss the generator Q-limiting algorithm as an example of these outer discrete control algorithms. Other algorithms of that kind are implemented similarly enough that explaining one of them in detail will suffice. The generator Q-limiting algorithm, in addition to
implementing the *ContinuationMethod* interface to interact with the *SugarSolver*, has access to the *PSModels* object including all generators of the system.

<table>
<thead>
<tr>
<th>Qlimiting : ContinuationMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>- generator_set : vector<a href="">Model::Generator</a>*</td>
</tr>
<tr>
<td>- gen_back_off : vector<a href="">Model::Generator</a>*</td>
</tr>
<tr>
<td>- set_begin : unsigned int</td>
</tr>
<tr>
<td>- set_end : unsigned int</td>
</tr>
<tr>
<td>- back_off_begin : unsigned int</td>
</tr>
<tr>
<td>- back_off_end : unsigned int</td>
</tr>
<tr>
<td>- check_q_limits(DenseArray const&amp; vector<a href="">Model::Generator</a>* &amp;,...)</td>
</tr>
<tr>
<td>- apply_limits() : void</td>
</tr>
<tr>
<td>- restore_previous() : void</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>+ initialize_continuation_method(...) : void</td>
</tr>
<tr>
<td>+ visitor_prepare_nr_loop() : void</td>
</tr>
<tr>
<td>+ visitor_nr_loop_succeeded() : void</td>
</tr>
<tr>
<td>+ visitor_nr_loop_failed() : void</td>
</tr>
<tr>
<td>+ finished() : int</td>
</tr>
</tbody>
</table>

Fig. 37 UML class diagram of the *QLimiting* class that implements the * ContinuationMethod* interface.

Fig. 37 shows a reduced UML class diagram of the *QLimiting* class. To fully focus on explaining the structure of the algorithm, we omit the sets of other components that hit reactive power limits in this diagram. Other components that are evaluated are continuously modeled switched shunts, FACTS devices, and VSC DC lines.

The core of the Q-limiting algorithm is implemented within three private functions. The *check_q_limits()* function finds all generators that hit their limits given the current solution vector and stores them for future use. The *apply_limits()* function applies limits (converts PV-generators to constant PQ models) to all or a subset of generators that hit their limit. The subset of generators to be limited is defined by the *limit_begin* and *limit_end* indices that are adapted to find subsets of generators for which the simulation converges. In an extreme case, both indices point at a single generator at a time. The *restore_previous()* function restores the previous state in case of a failed limiting attempt. Importantly, every function also applies to generators that back off from their limits.

Fig. 38 shows how the private control functions of the Q-limiting algorithm are utilized by the *ContinuationMethod* interface’s visitor functions. One can see that once the core of the control
algorithm is implemented, mapping the control actions to the interface is straightforward and only requires minor logic.

To explain Fig. 38 that shows the control algorithm's visitor functions in more detail, we walk through one control loop iteration of the Q-limiting algorithm. The algorithm starts with the `visitor_prepare_nr_loop()` function that is called at the top of the control loop. In this implementation, the first call to this function is differentiated from subsequent calls. The first call invokes the `check_q_limits()` function to find an initial set of generators to be limited. Subsequently, it applies these limits. In ensuing calls, only the limits of a defined generator subset (`set_begin`, to `set_end`) are applied.

Next, the altered system is re-solved (as defined by the control algorithm (Fig. 34)). After a successful inner loop, the search for generators that hit their limit as well as generators that back-off from their limit, is re-initiated. If the inner loop diverged, previously limited generators are reset to their prior state and the set of generators to be limited is reduced to a subset of the previous step. Finally, the `finished()` function is called at the bottom of the control loop to
determine if the algorithm has finalized. It returns “false” if there are remaining generators to be limited, and “true” otherwise, which continues the loop.

A note on how a reduced set of generators is established in case of a failed attempt: In case the algorithm is parameterized to limit as many generators at once as possible, the set is cut in half. Dependent on the sorting of the generator vector that is an additional parameter, then either the generators with the highest Q limits, with the smallest Q limits, or generators that exceeded their limits by the highest or lowest amounts are prioritized. In a final stage, when the subset is reduced to a single generator, every generator limit is applied individually.

Finally, it is worth mentioning that this implementation includes an option that is advantageous when composing the Tx-stepping algorithm and the Q-limiting algorithms to a single algorithm. This combined algorithm is applied to aid convergence of very complex systems with many remote generators. In fact, only generators that are controlling remote voltages are considered in this case. Likewise, the “partial limiting” option that only limits generators that were initially found to hit a limit is used for this composed algorithm.

**Device Control Algorithms**

As we have mentioned, it is possible to compose multiple control algorithms together to composite control algorithms. Importantly, the *ContinuationMethod* interface is general enough to allow composite algorithms to be implemented utilizing the same interface as simple control algorithms. One such composite algorithm is the previously mentioned Tx-stepping algorithm for complex systems with many remote-controlled voltage magnitudes, in which generators controlling these magnitudes are limited during the Tx-stepping algorithm to aid convergence. Importantly, implementing atomic control algorithms first and further composing them as composite algorithms that still implement the *ContinuationMethod* interface keeps the solver design clean even with a large number of implemented control algorithms. An alternative scenario that does not define such an interface is prone to become difficult to maintain after a certain number of implemented algorithms. Within such an implementation, the danger of unwanted interactions between the algorithms increases rapidly.
COMPOSED ALGORITHMS

An example of a composite control algorithm, that comprises of an adaptive number of sub-algorithms is the DeviceControl algorithm. It includes possible control of generator Q-limits, transformer taps, discrete switched shunts, and an algorithm to converge PF after DC line parameter changes.

Fig. 39 UML class diagram of the DeviceControl class.

Fig. 39 shows the UML class diagram of the DeviceControl algorithm. One can see that it stores every sub algorithm in a vector of algorithms and keeps track of the currently active algorithm via an index. The DeviceControl algorithm’s visitor functions relay the function call to the currently active sub algorithm. This creates a generic structure in which algorithms can be composed freely by adding them to the controls vector. The sequence in which they are added determines the sequence they are invoked. The DeviceControl control algorithm finalizes when all sub-algorithms finalize.

3.1.4 DATA PIPELINE

We have discussed the “ECF solver” structures and their corresponding control algorithms. Now, we will introduce the data structures that enable simulation of Power System problems. The C++ SUGAR implementation can read two commonly available formats of Power System models: PSSE .raw files, and Matpower .m files. Additionally, an internal file binary file format (.sug) can be used. The data that is defined in these files is parsed into the PowerFlowData class. This raw data is further refined to equivalent circuit models of each simulation component. These components compose the PSModels class that represents the ECF equivalent system model of a problem.
Fig. 40 Relationships between model data structures of the SUGAR C++ implementation.

Fig. 40 gives an overview of the model data structure of SUGAR C++. We will further discuss details of each component, starting with the *Parser*.

3.1.4.1 Parser

The data parser reads and writes PF case data to files. It implements an interface that enables the extension to additional file types without code changes outside of the *Parser* class.

<table>
<thead>
<tr>
<th>Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ parse(string filename) : PowerFlowData</td>
</tr>
<tr>
<td>+ write(string filename, PowerFlowData data): void</td>
</tr>
</tbody>
</table>

As we can see from Fig. 41, its interface is straightforward. It includes a `parse()` function that returns a `PowerFlowData` object, and a write function that writes a `PowerFlowData` object to a file. Importantly, the *Parser* enables the implementation of multiple sub parsers, that all implement the same interface and are instantiated by the *Parser* object, which means it utilizes the factory software design pattern.
Fig. 42 Algorithm of the SUGAR C++ parser that applies a factory pattern to instantiate the correct algorithm.

Fig. 42 shows an activity chart of the `parse()` function’s algorithm. After the `parse()` function is called, a solver object that specializes in parsing a certain file type is instantiated based on the ending of the input file that is parsed. Then the data is parsed and a `PowerFlowData` object is returned. The `write()` function of the `Parser` is implemented similarly.

3.1.4.1.1 PSSE “.raw”-FILE PARSER

As an example of a specialized parser, we discuss details of the PSSE .raw file parser that reliably parses “.raw”-files from file version 30-34. Interestingly, parsing every “.raw”-file version with one parser proves to be a complex endeavor, since the file structure between .raw file versions changes erratically. For example, in the bus definitions of version 30 and 31, shunts are defined immediately after the bus type. These positions shift for other versions. Additionally, the position of the owner definition switches place. In versions after 32 it is defined in front of the solution, while in versions before 32 it comes after the solution. Erratic changes like these continue for many elements between .raw file versions. This is exacerbated by the fact that a version number was only included into files after version 32. When a version number is defined, it is commonly faulty within files in general. This leaves a parser with the only option to infer the
file version from the file structure. Despite these challenges, our parser is heavily tested on a multitude of cases and operates reliably for even the most complex ones.

![RawParser class diagram]

**Fig. 43** Class diagram of the .raw file parser.

Fig. 43 shows the UML class diagram of the *RawParser*. It can be seen that a parse function for each raw file section is implemented. Correspondingly, write functions for each of these sections exist.

The parser works in the following way. It splits the “.raw”-file into its sub-sections and infers the file version from their structure. Next, it parses each section by tokenizing it line by line and assigning the parsed values to a *PowerSystemData* object. Finally, the filled *PowerSystemData* object is returned.

The Matpower file parser operates similarly. It first splits the “.m” case file into sections and further tokenizes the data, before filling the parsed data into a *PowerSystemData* object. Importantly, this is only possible because the data defined in a Matpower case file is a subset of the data that can be defined as part of a “.raw”-file. While Matpower case files exhibit less complexity compared to .raw files, their structure is not as rigidly defined, which slightly increases difficulties in tokenization.
3.1.4.2 PSMODELS – CLASS

The `PSModels` class loads the data from the `PowerFlowData` class to create corresponding ECF models that can further be used for simulation. It serves as a container for ECF models that together comprise a system model. Additionally, it implements functionality such as sanity checks of the data, and interfaces to apply continuation methods, control algorithms, or contingencies.

Fig. 44 shows an UML class diagram of the `PSModels` class. One can see that each class of ECF models is stored in separate vectors. This simplifies access to these models for control algorithms, post simulation analysis, and several other utility functions. Importantly, the functions `get_active_linear_equipment()` and `get_active_nonlinear_equipment()` are utilized by the NRSolver to obtain vectors of linear and nonlinear models to be stamped during a NR loop. Noteworthily, separating models into vectors for each type instead of storing all models together...
in a single vector using an interface speeds up the stamping process dramatically. This is because
the reduced overhead of virtual function call tables.

A *PSModels* class can directly be instantiated using a *PowerFlowData* object. Alternatively, PF
data is loaded into the class via the *create_models()* function. To create ECF models, specialized
“create” functions for each model class are implemented. Some ECF models, like static shunts,
transmission lines, or load models are straight forwardly created from the supplied data. Other
elements involve additional calculations.

Many of these calculations require a graph representation of the system in order to be
implemented efficiently. An example for such an algorithm is the obligatory sanity check that
each system island has an assigned slack bus. The *SystemGraph* object provides this graph
representation and is accessible for the *PSModels* class via a reference. The implementation of
the *SystemGraph* class is discussed in Section 3.1.6.

Finally, the *PSModels* class contains a reference to the *solver_settings* object. Solver settings
that affect models and their creation include a system wide loading factor, a flag that enables
ignoring bus labels, and other variables such as a minimum electrical distance between plant
models. Interestingly, ignoring bus labels is sometimes useful due to an ambiguity that results
when a generator is defined on a bus that is set to be a load-type bus. Usually, this is the case
when a generator hit its reactive power limit and is interpreted as a constant real and reactive
power device when the case-file is written. However, this situation is ambiguous since it could
also indicate that a generator’s status flag is set incorrectly. The user is informed about this
situation via a warning message. To further enable simulation of both possible scenarios, we only
instantiate generators that are located on generator-type busses by default but allow overwriting
this behavior by an “ignore bus labels”-flag.

3.1.4.3 Power System Analysis Models

The PSModels class aggregates all models within the SUGAR C++ implementation to a system
that in general poses a nonlinear equivalent circuit simulation problem. All models implement
the same interface for stamping their contributions to the system matrix and have their MNA
nodes assigned.
Fig 45 provides an overview of all models implemented in SUGAR C++. These are the PF models, additional models for the PF feasibility optimization algorithm, and the State Estimation algorithm. Notably, when an optimization-based simulation is started, each PF model additionally stamps its corresponding adjoint equivalent circuit contributions.

The implemented PF models correspond to all models that are defined in PSSE “.raw”-files. We have validated each model’s correct implementation with versions of the PSSE and the PowerWorld simulators. This validation process can become quite complicated, considering the amount of different input parameter settings that are possible for some models. This is especially true for transformer parameters.

**Transformer Model**

Transformer models of PSSE “.raw”-files have three parameters that establish which quantities are used to define a transformer’s tap ratio, phase angle, and losses. Together, these 3 parameters specify 18 different combinations of parameters that are converted into the transformer’s electrical model parameters. Our implementation has cross validated all 18 scenarios with PSSE and PowerWorld.
THREE WINDING TRANSFORMERS

Three winding transformers of “.raw”-files are defined in delta-configuration using the same set of parameters as two winding transformers. Our implementation converts them to a star configuration and models each star component as a two-winding transformer model. Importantly, each winding has its own state that can be on or off. We create the auxiliary star node as a set of two auxiliary matrix entries and do not assign an artificial bus number to this node. Notably, a three winding transformer only defines one magnetizing loss element that is modeled on its primary winding.

PLANT MODEL

Plant models are composite models that allow multiple generators to control a single voltage magnitude of a bus different from their own. They are the SUGAR C++ implementation’s solution to sets of “.raw”-file generators that control a single remote bus voltage magnitude. Instantiating such a model requires multiple auxiliary algorithms that are now presented.

---

**PlantModel : Model::Base**

- controlled bus : int
- voltage_setpoint: double
- accumulated_q_min : double
- accumulated_q_max: double
- initial_q : double
- limit_status : Q_LIMIT_T
- auxiliary_variable_index: mIdxType

- vector<Model::GeneratorModel"> generators_in_plant
- vector<Model::DynamicShuntModel"> dynamic_shunts_in_plant
- vector<Model::FACTSModel"> facts_in_plant
- vector<pair<int, Model::VSCDCLineModel”>> vsc_dc_line_terminals

```cpp
+ append_gen_to_plant(GeneratorModel & gen) : void
+ append_facts_to_plant(FACTSModel & facts) : void
+ append_vsc_to_plant(VSCDCLineModel & vsc, int terminal) : void
+ append_ss_to_plant(DynamicShuntModel & shunt) : void
+ stamp(SparseMatrix & const Array&, const Array&) : void
+ stampNonChanging(SparseMatrix & const Array&) : void
+ stampContinuation(SparseMatrix & const Array&) : void
+ set_model_node_indices(NodeMap &, ..., bool run_feasibility);
+ is_remote_controlled() : bool
+ check_q_limits(Array const& V, ...) : int
+ void check_q_back_off(Array const& V, ...) : void
+ set_limits(vector<Q_LIMIT_T> &limits) : void
+ get_plant_limited_status() : Q_LIMIT_T
+ is_plant_limited() : bool
```

---

Fig. 46 UML class diagram of a Plant model.
Fig. 46 shows an UML class diagram of a Plant model. The purpose of this model is to control a bus (the `controlled_bus`) to a voltage magnitude (the `voltage_setpoint`). To enable this control, a set of voltage controlling models cooperates. These models include generators, dynamically modeled shunts, FACTS devices, or VSC-DC lines (to be more precise a certain terminal of a VSC-DC line). The plant model stores pointers to every cooperating model in order to have access to these models. Furthermore, functions to append models of each type to a single plant are implemented.

Importantly, since all models control exactly one bus voltage magnitude, only one auxiliary variable is added to the system of equations. This auxiliary variable defines the reactive power that is necessary to control the bus’s voltage magnitude to its setpoint. To split the total reactive power into each model’s contribution, participation factors are defined. The sum of all participation factors is normalized to one, defining fractions of reactive power contributions to each element. Unfortunately, assigning these factors can become arbitrarily complex, especially when prioritizing reactive power delivery of certain models over others. This is possible since, in reality, all controls operate independently from each other. For example, it is valid for an operator to choose to change the setpoint of a switched shunt, or FACTS device up to their limits before generators adapt their contribution. Nevertheless, this macro modeling of reactive power control represents a compromise that works sufficiently given the provided information.

Creating a plant model additionally requires algorithms that check for validity of the created model. In order for a plant model to work properly, a clear control path between each generator and the controlled bus has to exist. A clear path is defined by a connection that does not include other voltage-controlled buses. We check for this, utilizing a shortest path algorithm of the SystemGraph. An additional requirement for a controllable plant is its electrical distance from neighboring plants with different voltage magnitude setpoints. It is easily imagined that two voltage magnitude setpoints that are electrically very close to each other result in extremely high branch currents that are likely unphysical and cause numerical stability problems for the solver. To evade this possibility, plant models must adhere to a minimum electrical distance to each other. A breadth-first search algorithm for electrically close plants is used to avoid such a scenario. If our implementation encounters two plant models that are electrically too close to
each other, it merges both plants to a single one. This is a valid assumption, since electrically close nodes will result in minimal voltage differences.

It is worth mentioning the following data peculiarity that is a common occurrence: Sometimes multiple models are set to control a single magnitude but define different magnitude setpoints. In this case, our algorithm prioritizes generator setpoints. For other discrepancies, the first encountered setpoint during model creation is used. Additionally, a warning message drawing attention to this ambiguity is printed.

Finally, we discuss a plant model’s interfaces. From Fig. 46 we see that a plant model implements the *Model::Base* interface that defines functions for matrix stamping and MNA node distribution. Additionally, interface functions that are utilized by the *QLimiting* control algorithm are implemented. Noteworthily, when a single generator of the plant hit its limits, the distribution factors of the remaining generators have to be reassigned. The plant is only considered to be reaching its limit when each of its elements is limited. When backing off from a limit, a plant assigns every single generator for back-off at once.

**Switched Shunts**

The instantiation algorithm of switched shunt models is notable. Switched shunts can be modeled in multiple ways. When inserted as dynamical switched shunts, they are treated similarly to generator models. In our C++ implementation, they are part of *Plant* models and are limited via the *QLimiting* algorithm. Discrete switched shunts, however, are modeled as static shunts within a Newton-Raphson loop. They require an outer loop control algorithm to be adjusted.

**Adjoint and Coupling Models**

For optimization-based algorithms, each element model additionally stamps its adjoint counterparts. The PF feasibility optimization algorithm requires two additional types of models to be instantiated: A *ReferenceModel* is the adjoint counterpart of a *SlackBus* model that sets the adjoint reference, and an *AdjointCouplingModel* couples primary and adjoint circuits via feasibility sources.
3.1.5 *SugarAlgorithms*

The aforementioned *SugarAlgorithms* class defines a global interface to the available algorithms and sub-algorithms of our SUGAR C++ implementation.

<table>
<thead>
<tr>
<th>SugarAlgorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>runPowerFlow(PSModels*, SugarSolver* ) : bool</td>
</tr>
<tr>
<td>runStateEstimation(PSModels*, SugarSolver* , const state_estimation_settings_s &amp; , const solver_settings_s &amp; ) : bool</td>
</tr>
<tr>
<td>createStateEstimationSample(PowerFlowData* , json const&amp; json_input , bool run, pf , MyUniformNumberGenerator&lt;double&gt;&amp; , vector&lt;int&gt;&amp;active_buses , vector&lt;int&gt;&amp;bus_permutation) : unique_ptr&lt;PowerFlowData&gt;</td>
</tr>
<tr>
<td>calculateStateEstimationMetrics(const DenseArray&amp; solution , PSModels* , vector&lt;int&gt; const&amp;active_buses , const PowerFlowData*pfld) : pair&lt;double,double&gt;</td>
</tr>
<tr>
<td>MC(PSModels* , SugarSolver&lt;double&gt;* , monte_carlo_settings_s const&amp; , spdlog::level::level_enum) : int</td>
</tr>
<tr>
<td>createTxStepping(...) : unique_ptr&lt;ContinuationMethod&gt;</td>
</tr>
<tr>
<td>createTxSteppingQlimting(...) : unique_ptr&lt;ContinuationMethod&gt;</td>
</tr>
<tr>
<td>createDeviceControl(...) : unique_ptr&lt;ContinuationMethod&gt;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>save(string filename , SugarSolver* , PSModels* , PowerFlowData* ) : int</td>
</tr>
<tr>
<td>write_line_loadings(string filename , SugarSolver* , PSModels* ) : void</td>
</tr>
<tr>
<td>write_generator_file(string filename , SugarSolver* , PSModels* ) : void</td>
</tr>
</tbody>
</table>

Fig. 47 UML class diagram of the *SugarAlgorithms* class.

Fig. 47 provides a UML class diagram of the *SugarAlgorithms* class. It defines algorithms for Power Flow, and State Estimation simulations. Furthermore, Monte Carlo simulations based on both simulation types can be started. The Power Flow feasibility optimization uses the same algorithm as a Power Flow simulation with a *run feasibility* flag signals the additional creation of adjoint circuits. Furthermore, *SugarAlgorithms* defines auxiliary algorithms for State Estimation that create synthetic SE cases (*createStateEstimationSample()*)) and calculate performance metrics of SE solutions (*calculateStateEstimationMetrics()*).

Functions to write output files are defined in the *SugarAlgorithms* class. They include a *save()* function that writes power system case files dependent on the file ending of the given file name (".m", ".raw", or ".sug"), and specialized write functions that output ".csv" based files of a system’s line loadings and generator set points.
Finally, sub-algorithms to create outer control loops are defined, standardizing their creation process. These algorithms return a pointer to the control algorithms interface class only, simplifying their handling. Notably, the `createNR()` function defines a simple Newton-Raphson loop that only applies limiting algorithms, if requested, without any outer loop controls.

![Flow diagram of the `runPowerFlow()` function of the `SugarAlgorithms` class.](image)

Fig. 48 shows a diagram of the `runPowerFlow()` algorithm. Visibly, these algorithms become complex quickly, therefore, we use these base-algorithms as building blocks for more evolved algorithms. For example, an algorithm running multiple artificial State Estimation cases calls the the `runPowerFlow()` algorithm to create a base solution and further creates SE cases utilizing the `createStateEstimationSample()` algorithm and subsequently the `runStateEstimation()` and the `calculateStateEstimationMetrics()` function in a loop.

### 3.1.6 Network Graph Library

Several algorithms within the SUGAR C++ implementation require a graph representation of the system to be implemented efficiently. Some of them are system island exploration, search for corresponding slack buses, and plant model minimal electrical distance enforcement.
The decision of implementing a lightweight graph library was made after exploring several third-party library options that did not fit the purpose. In fact, after careful design, the *SystemGraph* library’s performance is close to the performance of optimized graph libraries.

Fig. 49 UML diagram of structures and classes that represent the *SystemGraph* library implementation.

Fig. 49 shows a UML diagram of the *SystemGraph* library’s data structures. We decided to utilize an adjacency list-based data structure that is optimal for sparse graphs like a power system’s network graph. Compared to an adjacency matrix-based implementation, it is much more space efficient without having disadvantages during graph traversal. Our adjacency lists are stored in map containers of the C++ standard library.

The *SystemGraph* stores a map of vertices, that is indexed by corresponding system bus numbers. Furthermore, each vertex stores a map of edges and edge_properties and includes a vertex_properties structure. Herein, edge_properties include a measure of electrical distance (the magnitude of the complex impedance) and a status flag. Furthermore, vertex_properties include an element_counter that defines the number of connected models, the island number of a vertex, and a status flag. Importantly, these are the properties utilized in the graph-based algorithms of the current implementation. These properties are easily extended, since adding additional ones does not require any changes in the graph algorithms.

This graph structure can be traversed using search algorithms. A depth-first as well as a breadth-first algorithm were implemented using non-recursive algorithms with a focus on performance. They allow the implementation of more complex algorithms utilizing visitor function calls. The algorithm for minimal electrical distance utilizes the *dfs_zbr()* function that includes the electrical distance calculation directly within the search, which simplifies its
implementation and improves performance. A note on performance, theoretically both search algorithms scale with $O(|V|+|E|)$ and require $O(|V|)$ memory, where $|V|$ is the number of vertices and $|E|$ is the number of edges.

Importantly, for graph construction, a highly performant implementation is necessary as well. This becomes clear when looking at system sizes of 80k buses and over 100k branches that are partly defined in different data structures. We implement this initialization, with one data pass for each bus and branch type. After initialization we utilize a depth-first search algorithm to explore system islands.

The `island_bases` data structure stores roots for each island, where a root is an island’s slack bus if at least one slack bus exists (signaled by the `slack_bus` flag of the `root_properties` data structure). Utilizing the `island_bases` data structure, our implementation efficiently finds islands that do not have a slack bus assigned and fixes the problem. Additionally, traversing the vertices based on islands provides an efficient way of building the slack bus voltage-based initial guess of the Tx-stepping algorithm.

None of the algorithms implemented in this library cause a noticeable delay for any simulation. In a worst-case scenario of a PF simulation, an Eastern Interconnection planning case that defines multiple islands and requires few iterations to converge due to good available initial guesses, the library’s share in total computational cost is around 2.6%. For a 118-bus case power flow run, this share of total computational cost drops to 0.29%.

3.1.7 Settings and Input File Format

This SUGAR C++ implementation enables the simulation of three major algorithms for Power Systems (PF, PF feasibility, and SE) and implements a probabilistic solution framework. All of these components are configured using multiple parameters. We organize these parameters in a json-based settings file that is further parsed into accompanying data structures for each solver component.
Fig. 50 UML class diagram of input parameter data structures.

Fig. 50 provides an overview of these data structures. They include global settings such as the verbosity of the solver, solver parameters, parameters of control algorithms, as well as parameters for composite algorithms such as State Estimation or probabilistic analysis.

3.1.8 Used Third-Party Libraries

It is infeasible to design a high-performing software package without the efficient use of third-party libraries for tasks that are not unique to the developed software. The SUGAR C++ implementation utilizes libraries to solve sparse matrix problems, log progress to the console or to log-files, read and write data in json-format, and to generate random numbers.

3.1.8.1 Sparse Matrix Solvers

We utilize libraries of the SuiteSparse package to solve sparse matrices. Specifically we implemented SparseSolver interfaces for the UMFPack [100] and the KLU [99] solvers. Both libraries implement LU factorization algorithms.
**KLU**

The KLU library implements a sparse left-looking factorization algorithm and relies on permutations to block diagonal matrix forms. Additionally, it implements algorithms to reduce matrix-fill ins during factorization. Herein, the KLU set of algorithms is designed to specifically perform well when solving “SPICE-like” problems [99]. In the SUGAR C++ implementation, KLU generally outperforms UMFPACK (see Section 3.1.9), which is most likely due to this characteristic.

**UMFPACK**

The UMFPACK library uses an Unsymmetric MultiFrontal method and a direct sparse LU factorization algorithm [100]. Compared to that from KLU, this algorithm is more general. UMFPACK relies on BLAS (Basic Linear Algebra Subprograms). Hence, for optimal performance it is important that it is linked to a BLAS library that is optimized for a system’s architecture. Noteworthily, for probabilistic analysis with multiple threads solving matrices in parallel, it is imperative that the BLAS library is configured to run sequentially. Otherwise, the expected speedup might not materialize.

### 3.1.8.2 JSON-PARSER

To read and write json-input based files we utilize the open source “JSON for modern C++" library [101]. It is high performing and provides a simple-to-use interface with predefined getters for standard data types. Especially convenient is its ability to directly map JSON containers to C++ standard library containers. To read and write more complex data structures, one can implement two interface functions

```cpp
void to_json(json&, const data_structure &)
void from_json(const json &, data_structure &).
```

where *data_structure* is the type of the respective data structure. SUGAR C++ utilizes this library to parse the solver settings as well as to read probabilistic input files and write statistics of the probabilistic results as json-based output files.
3.1.8.3 Logger

Logging the progress of a command line tool is vital, since this is the only way to track a tool’s progress. Hence, the speed of the logger library is performance critical to the whole simulator, especially when the solver is set to verbose mode and logs inner loop information. The SUGAR C++ implementation uses the spdlog library [102] that is a very fast, thread-safe, header-only library for C++. Benchmarks promise that logging more than 5 million strings per second is possible when the library is accessed by a single thread.

3.1.9 Performance Evaluation of SUGAR C++

The total simulation time of nonlinear simulators depends on the problem and its initial conditions. Hence, the total simulator runtime is not a good measure to evaluate the performance of an implementation. This is especially true if homotopy methods are employed to solve a problem due to the unavailability of good initial conditions.

3.1.9.1 Computational Cost

To evaluate performance of the solver, we will first look at how the inner solution loop performs. This is defined by the per-iteration solve time of the simulator, which is a cost incurred for each nonlinear-solver iteration. The time to build the initial nonchanging matrix can be seen as a fixed cost. However, for linear algorithms or algorithms that need few iterations to converge, this cost might be substantial. We will now look at these timing measurements for different sized systems and for different solver settings.
Fig. 51 Per iteration solution times of the SUGAR C++ implementation for openly available Power Flow cases using KLU.

Fig. 51 shows per iteration times of the solver on different size, openly available power flow cases [103], [104]. The time shown includes the total time for nonlinear matrix stamping, sparse matrix solution, solution vector post processing, and final condition evaluation. In Fig. 51, the KLU sparse matrix solver was used. We will take a closer look at the time contributions of each component using a profiler. Importantly, these inner loop times are the time values where small improvements can magnify to big overall performance improvements of the solver.

In contrast to these inner loop times, the initial linear matrix creation time is a one-time cost, since these matrices are reused for each iteration during the analysis.

Fig. 52 Initial Power Flow matrix creation time of selected power system cases. This is a one-time cost.
Fig. 52 shows the initial linear matrix creation times. Compared to the per iteration times these costs are higher. However, since the linear matrix is only created once, the total impact of linear matrix creation on the total solution time is generally negligible. However, these times do give us a good measure for the speed of the stamping algorithm.

### 3.1.9.2 Performance Profile

Next, we discuss the relative impact of different solver components on the total simulation cost. To create performance profiles of our software, we used the “callgrind” tool [105] of the profiling software Valgrind. First, we will look at cases where a good initial guess is available and the convergence is reached within a few iterations. Then we will look at cases where Tx-stepping is needed for them to converge. This increases the amount of total iterations to problem convergence and adds calculations of the outer control loop (Tx-stepping) to the total solver performance.

![Performance Profile Diagram](image_url)

Fig. 53 Performance of the SUGAR C++ solver for a real Eastern Interconnection case and good initial conditions

Fig. 53 shows the solver’s component’s contributions to total performance when running PF on a real Eastern Interconnection case (over 78k buses) with good initial conditions. Herein, generator Q-limits are taken care of within the simulation. For this scenario we see that parsing the “.raw” text file actually takes up 24.3% of the total simulation time. Creating the ECF models takes another 13.51%. The nonlinear solver algorithm runs for 61.77% of the time. Here, 51.18% is spent solving linear sparse matrices, and 2.21% stamping nonlinear elements. Interestingly, for simulations that converge quickly, building the linear matrix still requires 2.8% of the total
simulation time. Interestingly, this is using the KLU sparse matrix solver library that outperforms UMFPACK for this type of matrix. Using the same test set up, we found UMFPACK to incur 79.5% of the total simulation cost. From Fig. 53 we see that sparse matrix solution is the bottleneck of the total simulator performance in our solver. For simulations that take more iterations and outer loops with control algorithms, this situation becomes even more clear.

![Diagram](image)

**Fig. 54 SUGAR C++ profile of a real eastern interconnect PF simulation using Tx-stepping.**

Fig. 54 shows the profiler output for the Eastern Interconnection case without a good initial guess. The solver needs more iterations, including multiple Tx-stepping steps to find a solution. Hence, the inner loop performance becomes more important. This inner loop performance is limited by sparse matrix solving, which takes up 83.93% of total simulation time. Other functions such as the `stampContinuation()` method that stamps the Tx-stepping system changes, the nonlinear stamps, and the voltage postprocessing together incur around 10% of total computational cost.

All previous performance studies show the solver’s performance for PF problems. When solving problems involving optimization algorithms within the ECF, we deal with matrixes that are double in size as compared to PF. However, since our SE algorithm is linear, only a single sparse matrix solution is required.
Fig. 55 shows convergence times for synthetically created state estimation cases that were solved using our linear algorithm. We notice that we solve the biggest SE cases in the order of one second. This computational effectiveness suggests the possibility of probabilistic SE studies. We will present such studies in the Results chapter.

3.1.9.3 Memory Profile

Finally, we study the memory profile of our implementation using the “valgrind” tool for memory profiling “massif”. Within this tool we use the “--stacks=yes” flag to enable profiling of program stack and heap.
The massif-visualizer [106] prepares a graphical representation of a program’s memory-profile. Fig. 56 shows such a profile for a Power Flow study of a 9241-bus test case. We see that the uncompressed system matrix (TripletMatrix) is the single biggest structure during this simulation. However, this data structure is short lived. It is destroyed with the creation of the compressed system matrix (CSSparseMatrix) (2.4 MB). After the uncompressed system matrix, the KLU solver is the second biggest data structure and requires 6.2 MB of memory. The peak memory use of this simulation was found to be 32.4 MB. In addition to the already mentioned structures, this includes input-data, ECF models, additional sparse sub-matrices, the system graph, and stored intermediary solution vectors.

![Fig. 56. A double-logarithmic plot of requirement for different sized systems for total required memory, KLU memory (symbolic and numeric factorizations), and system matrix size.](image)

Fig. 57 shows the memory requirements of SUGAR C++ for different sized Power Flow cases in double logarithmic form. Interestingly, we find a close to linear relationship between system size and required memory for all three studied variables. This unexpected finding is due to increased sparsity of increasingly big test systems. Additionally, we find that for the studied systems the memory requirements of the KLU algorithm are on average 2.87 times the size of the system matrix, and the total memory requirements of the solver are on average 14 times the size of the corresponding system matrix.
3.2 **Monte Carlo Framework**

To create probabilistic results for the implemented ECF algorithms, we implemented a Monte Carlo based numerical probabilistic tool that bases on our ECF solver implementation. Our main goal for this MC framework implementation was to run efficient probabilistic simulations on the available server infrastructure.

3.2.1 **Properties of Monte Carlo Algorithms**

Simple random sampling Monte Carlo algorithms are known to converge with an error decrease of $1/\sqrt{n}$, where $n$ is the number of samples. This means that for a tenfold increase in accuracy, one hundred times more samples have to be run. This is comparatively slow. Hence, algorithms exist to improve on this convergence rate. However, the MC algorithm is the only general algorithm for probabilistic studies and will be used in this implementation. Monte Carlo simulations offer almost complete data parallelism, which implies extremely good weak-scaling properties. These problems are commonly referred to as “embarrassingly parallel” problems.

3.2.2 **Design Considerations**

In order to design a high performing algorithm, multiple options arise. One option is to use bulk parallel programming techniques that efficiently scale across multiple compute nodes: These require post processing of Monte Carlo sample results on a file-system level or communication of results using message-passing libraries such as MPI [107]. Alternatively, shared memory programming offers the efficient utilization of single compute nodes utilizing a thread-level parallel paradigm. To create shared memory parallel programs, multiple libraries exist; e.g. the thread building blocks [108] or openMP [109] libraries. Since C++11, the “thread support library,” which is part of any C++ standard library, offers features to implement memory parallel programs. These features include direct interfaces to system threads, locks and atomic data types for synchronization, or higher-level structures such as asynchronous execution scheduling featuring asynchronous interactions with futures and promises.

In our implementation we aim for efficient use of a single compute node while keeping minimal overhead and maximum control of the programming model. Hence, we decided on an
implementation using C++ thread support library functions. As we will see, this design choice also enables us to fully control data reuse and sharing through the C++ copy and move construction feature. Note that this design choice does not limit potential speed-up across multiple machines. In fact, this is trivially achieved by starting multiple thread parallel MC cores on different machines while utilizing different RNG streams to guarantee independent random samples.

3.2.3 IMPLEMENTATION

The Monte Carlo algorithm design is straight forward. We utilize parallel worker threads for computation, while printing intermittent updates in the main thread. However, careful synchronization is required when simulation results are stored in a shared data structure and when final conditions are determined based on all previous results.

![Flowchart of the Monte Carlo algorithm](image-url)

**Fig. 58** The Monte Carlo algorithm.
Fig. 58 presents the _MonteCarlo_ algorithm in a flow diagram. The algorithm starts parallel threads that have access to a _SugarSolver_ and a _PSModels_ structure. The worker-thread clones the _SugarSolver_ that include an initial guess and shared matrix structures that do not change between samples. It then creates a probabilistic sample by cloning _PSModels_ utilizing the specialized _clone_probabilistic()_ function. Cloning of these data structures enables us to use move and copy to determine which data is shared and which data needs to be recreated for every probabilistic sample.

Next the worker thread starts the requested ECF based algorithm. After the algorithm’s completion, the worker thread stores the solution in a data structure that is shared between all threads. A write lock needs to be acquired to access this data structure. After writing and establishing that this latest result did not meet a final condition of the Monte Carlo algorithm, the worker releases the write lock and returns to the top of the loop to calculate the next sample. When a final condition is met, the thread exits and notifies all other threads to suspend calculations. After all threads have joined the main thread, the results are printed and the Monte Carlo algorithm finalizes. Final conditions are defined by the maximum sample count or when the probabilistic algorithm reaches a confidence threshold on a simulation’s parameter.
Fig. 59 shows a UML class diagram of the MonteCarlo solver and its related data structures. One can see that the MonteCarlo class implements synchronization using mutexes and atomic counters that partly serve as semaphores. Additionally, a condition variable is used to stall and reawake the main thread.

A vector of MonteCarloData objects serves as common solution store. In addition to the raw solution vector, multiple statistics of a simulation are stored. Importantly, performance timing measurements that are stored in the MC_MEASUREMENTS struct introduce substantial overhead and thus can be enabled and disabled during compile time.

Each ECF model that implements probabilistic variables inherits from the Probabilistic interface that defines functions for handling probabilistic model. Furthermore, each probabilistic variable is defined using the pointer to a Distribution_I interface that can include different distributions. Currently, Normal and Uniform distributions are implemented in this framework.
Importantly, each model implements copy and move constructors and a randomize function for probabilistic sample creation.

Fig. 60 UML class diagrams of the Probabilistic model interface and the class interface for defining and sampling a variable's distribution

Fig. 60 gives an overview of these interfaces. Importantly, additional functions of the Distribution_I interface enable the identification of model, element, and location data that is connected to the distribution object's variable. This information is crucial to create inter-model correlations.

3.2.4 Probabilistic Input Data

The SUGAR Monte Carlo framework reads probabilistic input data in json-format. Variable's uncertainties can be defined on a global, zonal, area, or bus level.

```json
"probabilistic": {
  "global": {
    "load": {
        "P": {
            "distribution": "normal",
            "stddev": 0.1
        }
    },
    "zonal": [
        { id: 1,
          "xfmr": { }
        }
    ],
    "area": [],
    "bus": []
  }
}
```

Fig. 61 Example of json-based probabilistic input.

Fig. 61 shows an example of a json-based probabilistic input. It shows the global definition of normally distributed real power loads with a standard deviation of 0.1 of their nominal values. Additionally, it defines network uncertainties for transformers in area 1. Uncertainties can be defined for each variable that has a mapping to a json-element defined.
A model calls helper functions that implement search algorithms for matching uncertainty definitions. These functions instantiate a probabilistic variable and return it to the model. Importantly, in this search more local definitions are prioritized over global ones.

3.2.5 Used Libraries

The only third-party library that was used for the implementation of this Monte Carlo framework was a pseudo random number generator. The importance of a fast, high-quality random number generator cannot be understated. We utilize the PCG RNG [95] library for that purpose. More precisely, we implement an interface to the “pcg32” RNG type.

3.2.6 Correlations within Monte Carlo Simulations

Two ways to include variable correlations into numerical simulations exist. If variable interactions are assumed to be linear, uncorrelated random numbers are transformed into correlated ones by multiplication with a Cholesky factorized correlation matrix. This is our current approach.

Alternatively, nonlinear variable interactions can be defined via Copula functions. However, this increases modeling complexity substantially. Many different copulas with multiple parameters exist. Defining realistic model parameters for these functions becomes increasingly difficult for these functions without a substantial amount of available data. A good overview of copula functions used in power systems modeling can be found in [110].

3.2.6.1 Design Considerations and Implementation

We distinguish between two types of correlations that are defined using slightly different information. Correlations of variables within models can be defined within the uncertainty definitions of the models. Since there are no interactions between models, local correlations are parsed and instantiated in a straightforward manner as part of a model.
Fig. 62 Branch model that defines a localized correlation between its parameters “Gser” and “Bser”.

Fig. 62 shows an intra-model correlation defined for variables of a branch model.

Alternatively, correlations between models can be defined. These correlations define interactions of a single, or two types of variables across the whole system, within a zone, or area.

"inter-model-correlations": [  
  {    
    "from": {    
      "model": "gen:wind",    
      "element": "P",    
    },    
    "to": {    
      "model": "gen:wind",    
      "element": "P",    
    },    
    "value": 0.8,    
    "exp-decay": 0.5,    
    "power-decay": 0.0  
  },
],

Fig. 63 Defines wind power generation correlations between generators in a system.

Fig. 63 defines positive wind power correlations across a system. Importantly, the “to” element can identify a different variable. For example, negative correlations between wind power availability and PV generation could be defined using two different elements. In addition to defining the base value of a correlation, we define parameters for functions that define the weakening of correlations with increased distance between variables. We can define exponential or simple power laws for this decay and approximate the distance between variables by their shortest path distance on the system graph.
This information is sufficient to create a global correlation matrix for a system. Each distribution is assigned a matrix row that is defined by a distribution’s position in a vector of all distributions in a system that is initially created. Now, creating correlation matrix entries for localized correlations between two variables is implemented with a data-pass through all models. We simply use an interface function that returns correlations of probabilistic variables within the models.

Creating correlations between variables requires a different mechanism. To avoid multiple model-data passes for the search for matching variables, we add functions that return identifying information such as a probabilistic variable’s element and parameter name directly into the I_Distribution interface. Now, a search through the vector of all system distributions suffices to create globally correlated variables. In addition, information about a distribution’s local bus is available, which enables us to implement correlation-weakening effects with increasing distance between variables.

A system’s correlation matrix does not change between MC samples and is reused within our framework. However, it is stored in a Cholesky factorized form to enable direct multiplication to transform uncorrelated samples to correlated ones. Importantly, distributions have to be shifted to a zero mean before creating correlated samples and shifted to their original positions afterwards.

While creating uncorrelated samples is possible within a single data pass, creating correlated random samples requires a three-step process. First, random samples for each variable are drawn. These random samples are stored in a vector of system distributions. A reference to this vector is passed to the MyCorrelatedRNG class (Fig. 64). In a second step, correlated samples are created by multiplication of previously drawn uncorrelated samples with the Cholesky factorized correlation matrix. Lastly, the correlated random variables are reassigned to the distributions. Because only reference information was passed to the MyCorrelatedRNG, the PS models now see this information and a correlated system sample is created.
3.2.6.2 Third-Party Libraries

An additional third-party library is required to calculate the Cholesky factorization. We use the CHOLMOD library [111] that is part of SuiteSparse for this purpose.

3.2.7 Memory Profile of the Monte Carlo Framework

In contrast to the detailed memory profile in section 3.1.9.3 we employ a method with less overhead to study memory scaling properties of our MC framework. We investigate how the maximum memory use changes with increasing parallelism for different size systems using the Linux standard tool “time -v”.

![Fig. 65 Memory scaling properties of the MC framework for three different size system with additional parallelism.](image)

Fig. 65 shows how additional parallelism increases memory requirements for a 118-bus system, a 9241-bus system and an 80k-bus system. Looking at the trendlines for each system we find sub-linear scaling with exponents of .32, .70, and .74, respectively.
4 Algorithms and Results

This chapter covers four topics in which advancements were made. First, an improved implementation of the Tx-stepping algorithm is presented. Then we introduce a continuation method that is essential for efficient probabilistic studies of big, complex power-system models such as planning models of the Eastern Interconnection. Furthermore, we present probabilistic PF and SE studies to showcase the capabilities of our probabilistic solver. Then, we present probabilistic studies including correlations within and between models. Finally, modelling improvements of control algorithms within probabilistic PF are proposed.

4.1 Homotopy and Continuation Methods

4.1.1 Adaptive Tx-stepping

The Tx-stepping algorithm was developed to robustly solve complex power flow cases with no available initial conditions. This conservative approach started at an Tx-stepping homotopy factor \( \lambda = 1 \) and an initial step size of 0.1 that was only reduced in case of divergence of sub-samples. This concept led to a robust algorithm for single cases, including contingencies of real eastern interconnection systems [64]. However, this first implementation left room for improvement in efficiency.

We developed an adaptive step-size algorithm to increase efficiency of the Tx-stepping method that, while keeping the robustness of the original algorithm, converges much faster to the solution. For this we take a more aggressive approach when initializing the homotopy factor. Now we aim to find an initial solution at \( \lambda = 0.5 \) and only increase \( \lambda \) in case of divergence. Additionally, we adapt step sizes much more dynamically. In this improved algorithm there are two different factors changing the steps that the Tx-stepping algorithm is taking, one for forward steps towards the solution, and one for steps backwards towards the initial solution.

To optimize this implementation, we run an exhaustive search for an optimal parameter set over a set of realistic parameters and all complex available base cases.
Table 4 Adaptive Tx-stepping convergence times and number of Tx-stepping steps necessary to converge real Eastern Interconnection testcases ranging between 78k-79k buses.

<table>
<thead>
<tr>
<th></th>
<th>17SLL</th>
<th>17SUM</th>
<th>18SLL</th>
<th>18SUM</th>
<th>18WIN</th>
<th>21SLL</th>
<th>21SSH</th>
<th>21SUM</th>
<th>26WIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time [s]</td>
<td>39.5</td>
<td>50.2</td>
<td>36.8</td>
<td>48.0</td>
<td>50.5</td>
<td>32.2</td>
<td>69.4</td>
<td>46.9</td>
<td>60.2</td>
</tr>
<tr>
<td># steps</td>
<td>30</td>
<td>35</td>
<td>27</td>
<td>32</td>
<td>32</td>
<td>20</td>
<td>41</td>
<td>26</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 4 shows the resulting convergence speed and required Tx-stepping steps for nine different real Eastern Interconnection planning cases that range in size between 78 and 79k buses. Importantly, among the available cases, those that are the most difficult to attain convergence include a big share of plant models that control remote voltages. Additionally, they include almost every PS model that the PSSE “.raw”-file standard defines.

4.1.2 Probabilistic Sample Continuation

The Tx-stepping algorithm is sufficiently robust and efficient to solve big planning cases, including their contingencies for engineers to efficiently work with these cases. However, for many extreme cases or a huge number of samples, its efficiency and robustness can further be improved. The probabilistic sample continuation method utilizes domain knowledge of Monte Carlo samples and a reference solution to create a robust solution method for MC samples. As we will see, using this additional information enables us to create an algorithm that is much better suited for probabilistic studies due to its improved robustness and convergence speed. The probabilistic sample continuation method requires a prior solution to a similar system and a known set of system changes that lead from this system to the problem at hand. A probabilistic sample fulfills this requirement, when at least one prior sample has been solved.

Without loss of generality, we assume that a reference solution of the original system (each probabilistic variable at the mean) exists. We can then define a continuation from the reference system to any sample, by tracing each probabilistic variable’s derivation from its mean

\[ H(x, \lambda_{mc}) = \lambda_{mc} F_{sample}(x) + (1 - \lambda_{mc}) F_{mean}(x) \]  

(82)

where \( \lambda_{mc} \) is the probabilistic continuation’s homotopy factor, \( x \) is a vector probabilistic variables, \( F_{mean}(x) \) describes the system at the reference solution, and \( F_{sample}(x) \) is a probabilistic sample.
Fig. 66 Depiction of the probabilistic sample continuation algorithm for two probabilistic variables $x_k$ and $x_j$.

Fig. 66 shows a graphical representation of this algorithm for two probabilistic variables ($x_k$ and $x_j$) in the system. When a MC sample’s solution is not found using NR and limiting methods, the probabilistic continuation is started. It scales each probabilistic variable back towards a value where a known solution exists using the reference solution as initial guess. When a solution to a scaled system is found, this solution is used as new initial guess and a solution attempt of a system closer to the desired sample solution is started. This algorithm is repeated with adaptive step sizes until a solution to the MC sample is recovered.

The probabilistic sample continuation method is implemented using the ContinuationMethod interface. Its visitor functions are shown in Fig. 67. The continuation factor $\lambda_{mc}$ is initialized to

---

![Diagram](image-url)
0.5, since this algorithm is started after a failed solution attempt of a probabilistic sample. This means we initially attempt to solve a system half-way between the sample and reference system. The initial step size of the algorithm is 0.5 as well. This means that, if a solution at $\lambda_{mc}=0.5$ is found, a solution attempt of the sample is attempted immediately. When no solution is found, the step-size is cut in half before starting the next attempt. This means we implement an asymptotic approach to the reference system in this first stage of the algorithm. As soon as this algorithm finds its first solution the step-size is doubled, and the system is scaled towards the sample.

4.1.2.1 COMPARISON TO STATE-OF-THE-ART

We compare a probabilistic solver using the probabilistic sample continuation method to a Monte Carlo solver that was implemented using the openly available MATPOWER package [112].

![Fig. 68 Comparing probabilistic results of the 13659-bus PEGASE testcase using SUGAR C++ with sample continuation and a Matpower based MC engine.](image)

Fig. 68 shows the estimated probability of system collapse of the 13659-bus PEGASE test case with increasing real and reactive power uncertainty. We estimate this probability by the ratio of non-convergent samples to the total sample number. Notably, this is a very crude abstraction for a probability of system collapse, since control algorithms that would keep a strained system
within a normal operating range are not modeled. However, looking at this measure we see that
the MATPOWER-based algorithm overestimates the probability of grid collapse by 19% on
average. This highlights the need for a robust probabilistic solver for high quality probabilistic
results.

4.1.2.2 COMPARISON TO TX-STEPPING

To compare this algorithm to the robust Tx-stepping algorithm, we adapt the probabilistic
solver and force a solution via continuation methods for every sample. This enables us to look at
a more complete picture of the sample continuations performance. For this study, we use a global
normally distributed PQ-load uncertainty with a standard deviation of 2% of every load’s defined
P and Q values and study 1000 identical MC samples using both methods.

Table 5 Probabilistic samples converged with the use of Tx-stepping or the sample continuation method without a prior solution attempt.

<table>
<thead>
<tr>
<th>runtime/sample [s]</th>
<th>2383wp</th>
<th>9241pegase</th>
<th>Synth10k</th>
<th>Synth25k</th>
<th>SUM18</th>
<th>ACTIVSg70k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob. Continuation</td>
<td>0.066</td>
<td>0.266</td>
<td>0.427</td>
<td>0.828</td>
<td>2.09</td>
<td>2.087</td>
</tr>
<tr>
<td>Tx-stepping</td>
<td>0.375</td>
<td>7.200</td>
<td>16.767</td>
<td>10.34</td>
<td>86.4 (977)</td>
<td>184.69 (923)</td>
</tr>
</tbody>
</table>

Table 5 shows average sample convergence times for 1000 Monte Carlo samples of each case.
We see that the probabilistic sample continuation method uses its advantage of additional
knowledge and under these circumstances clearly outperforms Tx-stepping. In fact, efficient
probabilistic studies of big systems are only viable using such an approach. Hence, all probabilistic
results presented in the following are generated using the probabilistic sample continuation
method.

4.2 PROBABILISTIC PF

In this section we present probabilistic studies that were created using our framework. We
begin with a probabilistic study on a real EI planning case. Further, we will model correlations
within the probabilistic solver and propose an algorithm to model generation control based on
probabilistic sample differences from the base case.
4.2.1 Probabilistic Study of EI Case

First, we present a study on a MMWG Eastern Interconnection planning case (SUM17) with 78319 buses. Note that these planning cases are designed to model a two-year extreme, meaning there is a 50% chance that this extreme situation occurs in the modeled year. Hence, these cases are among the most complex to solve. Additional complexities that require robust solution algorithms are 1) the number of generators set to control voltage magnitudes that are not located at their respective buses and 2) the number of different models and data structures that have to be modeled correctly in order to find the correct solution. Only a well-validated solver, such as our C++ SUGAR implementation, can solve these cases.

As a scenario for our probabilistic study, we assume that a system operator is interested in a probabilistic study of tie-line flows. Tie lines are the lines connecting a system operator to its neighbor. Hence, we choose to study all buses that connect the system operator’s network with its neighbors. We choose PJM as system operator and assume that PJM has increased knowledge about its own system and model the non-PJM system with higher uncertainties.

<table>
<thead>
<tr>
<th>uncertainty definitions</th>
<th>Branch uncertainty (R/X) [%]</th>
<th>Transformer uncertainty (R/X) [%]</th>
<th>Load uncertainty (P/Q) [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJM</td>
<td>2.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Others</td>
<td>5.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Table 6 shows the assumed uncertainties. All uncertainty values are in relation to the nominal values of the respective components. We assume normally distributed uncertainty distributions. With these input parameters, we start a MC study of $10^5$ samples that we evaluate on an Intel Xeon server CPUs (E5-2680 @ 2.40GHz) using 45 threads for parallel processing of MC samples. Under these circumstances, the study needed 7h and 8mins to converge. This is sufficient for engineering cycles, where commonly one study a day is required. However, with additional resources, one probabilistic study of such a system in an hour is conceivable.
Fig. 69 Slack bus real power injections into the system for the 2 biggest islands. Negative values mean power generation.

Fig. 69 shows the real power slack bus injections of the two biggest islands in the system. Here the smaller island comprises of 1266 buses and the big island comprises of 76304 buses. Together, they supply angle reference and slack powers for almost the full system. Herein, the spread of power injections of the larger slack bus amounts to more than 800MW. Some of this spread could be reduced using outer control loops, such as transformer tap changes and discontinuous switched shunts. However, it is more realistic to model generation control that in a real system is more likely to control these system variations.

Fig. 70 Slack bus reactive power contributions to the system for the two biggest islands.

Fig. 70 shows the corresponding reactive power slack bus injections of the two bigger islands. Both distributions show a distortion from a normal distribution that favors solutions with higher reactive power injections. These are likely a result of generators reaching their reactive power supply limit for samples that require more reactive power. Hence, the slack bus must supply comparatively more reactive power.
Fig. 71 Extreme voltage magnitudes and angles of the EI test case study.

Fig. 71 shows pdfs for the maximum and minimum voltage magnitudes and angles that were found in the system during this study. Herein, the voltage angle pdfs show clear deviations from a normal distribution. This is easily explained by the fact that they are comprised of values that are acquired from multiple busses, dependent on where a sample’s extreme values are located.

In this experiment we have observed the flows on each of PJM’s bordering tie lines. We will now examine the flows on of these lines as an example.

Fig. 72 Tie-line real power flow on line between PJM and AP grid operators.

Fig. 72 shows the probability of real power tie-line flows on a line between the PJM and the AP grid operators. We can identify the probability of maximum flows for these tie lines and include
conservative bounds based on confidence interval calculations for their flows. For the line in Fig. 72, we find a less than 1% chance of a flow of more than 101.69MW (conservative 95% interval bound).

![Reactive power flow distribution](image)

**Fig. 73** Tie-line reactive power flow between PJM and AP grid operators.

Fig. 73 shows the reactive power flow on this line. A 95% confidence bound finds a 1% chance for a flow of 97.32MVAr.

### 4.2.1.1 ON THE RUNTIMES OF MC SAMPLES INCLUDING GENERATOR LIMITING

The runtime performance of the solver is mostly limited by the number of required steps of the non-continuous generator Q-limiting algorithm. In fact, while an initial solution before this algorithm was found for all $10^5$ samples of this study, the generator Q-limiting algorithm failed to converge for 190 samples. While it is likely that feasible solutions to these samples are found with additional control algorithms, such as transformer tap controls and switched shunt controls, these algorithms are also implemented using non-continuous switching. Hence, they are equivalently prone to convergence problems. An approach to improve on these non-continuous algorithms was proposed in [41].
Fig. 74 shows the distribution of required generator limiting steps of each sample. Interestingly, we encounter a pdf with two defined peaks around 4 and 11 steps for most samples. The boxplot Fig. 74 includes all (converged and failed) samples. Notably, a physical solution for the highest outlier (215 steps) was found.

Fig. 75 shows the distribution of required generator limiting steps of each sample. Interestingly, we encounter a pdf with two defined peaks around 4 and 11 steps for most samples. The boxplot Fig. 74 includes all (converged and failed) samples. Notably, a physical solution for the highest outlier (215 steps) was found.

The runtime of a single sample shown as a histogram in Fig. 75 correlates highly with the number of required generator Q-limiting steps. This explains the split peak pdf for common samples with peaks around 7 and 14 seconds. However, for outliers we see that the algorithm can sometimes spend multiple minutes trying all possible generator Q-limit combinations in an attempt to find a solution.
4.2.2 CORRELATIONS AND PROBABILISTIC CONTROL MODELING

We have established that the SUGAR C++ MC framework scales to interconnection-sized systems. Now, we look at modeling improvements such as the modeling of correlations and a probabilistic generation control algorithm that aims to improve on the slack bus assumption. For these studies, we will use a synthetic 2000-bus case that models a power system on the footprint of Texas. This system is selected because it contains generator fuel type information and defines big share of wind generation.

In the following experiments, we will look at slack bus power injections as a proxy measure for the whole system state under different scenarios. This enables us to give a clear standardized picture of the modeled effects across different scenarios. In reality, an operator would likely be interested in selected bus voltage states or flows through critical infrastructure. Importantly, selecting these variables of interest requires good prior knowledge of a system.

Table 7 Base uncertainties for synthetic Texas case (2000-bus).

<table>
<thead>
<tr>
<th>Branch uncertainty (R/X)</th>
<th>Transformer uncertainty (R/X)</th>
<th>Load uncertainty (P/Q)</th>
<th>Wind power injection (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0%</td>
<td>5.0%</td>
<td>5.0%</td>
<td>10.0%</td>
</tr>
</tbody>
</table>

Table 7 shows the assumed global base uncertainties for all studies using the synthetic 2000-bus case as their underlying power system model. First, we add variable correlations to these definitions.

4.2.2.1 CONSIDERING INTRA-MODEL CORRELATIONS

Two distinct types of correlations can be modeled within our framework. Intra-model correlations affect a pair of variables of a single model, whereas intra-model correlations define interactions across models.
Fig. 76 Definition of intra-model load correlations in the SUGAR C++ probabilistic input.

Fig. 76 shows how intra-model correlations are defined in the SUGAR C++ input file. They are part of a variable’s definition in the model description and identify a correlation by name of the correlated variable and correlation factor.

```json
"load": {
  "P": {
    "distribution": "normal",
    "stddev": 0.05,
    "correlation": {
      "Q": 0.8
    }
  },
  "Q": {
    "distribution": "normal",
    "stddev": 0.05
  }
}
```

Fig. 77 Effects of a positive correlation (factor 0.8) between real and reactive powers on all loads.

Fig. 77 shows the effects of intra-model load correlations on the real power slack bus injections. As expected, correlations between loads induce widening of the resulting pdf. This means that extreme cases are more likely with correlated variables. Intra-model correlations can be defined between any variable of a model including localized definitions for a zone, area, or certain bus. However, they are not able to define variables’ correlations across models.
4.2.2.2 Considering Inter-Model Correlations

To study effects of correlations across models, we define inter-model correlations. Correlated wind power generation is one of the most studied examples for this type of correlation.

```
"inter-model-correlations": [
  {
    "from": {        
      "model": "gen:wind",
      "element": "P",
    },
    "to": {        
      "model": "gen:wind",
      "element": "P",
    },
    "value": 0.8,
    "exp-decay": 0.5,
    "power-decay": 0.0
  }
]
```

Fig. 78 Definition of inter-model correlations.

Fig. 78 defines the wind power generation correlations. Here, a wind generator model is defined using a colon as separator between the model name and its selecting parameter ("wind"). We assume that the correlation (factor 0.8) is reduced exponentially with increasing distance of wind power generators. Since no geographic data is available in this case, we assume that we can measure the distance by the number of bus hops between two generators. Alternatively, we could define the decrease of correlation using a power law.

Fig. 79 The effects of wind power correlations.
Fig. 79 displays the effects of correlated wind power generations on the real power slack bus injections. Similar to the effects of load correlations, we see an increased likelihood of extreme samples.

4.2.2.3 Probabilistic AGC

All previous probabilistic studies utilized a slack bus to supply the resulting load/generation and loss differences of a sample to the base case. However, with increased uncertainty and the resulting increased likelihood of high deviations from reference slack bus injections, this assumption loses its validity. It is necessary to improve on this assumption for probabilistic studies for different time frames.

For time frames that cross market intervals, an abstraction of dispatch laws can be implemented [7]. However, considering recent advances in Optimal Power Flow algorithms [113], a stochastic OPF would be the preferred solution. For shorter time frames, automatic generation control, such as droop or AGC control, should be modeled for more realistic results. Deterministic modeling solutions for these algorithms include distributed slack bus implementations [66] or including frequency into power system models [41].

However, we propose a solution that utilizes sample deviations from the base case. Our algorithm re-dispatches a selected set of generators based on a sample’s load and generator deviations to rebalance the system. Our re-dispatching algorithm follows a linear law and is implemented prior to the sample solution. Hence, it models a combination of short-time (primary and secondary) controls that automatically adapt a system to fast changes. We call this algorithm probabilistic AGC. However, we are aware that this is a simplistic algorithm that models a re-dispatch for short term imbalances. While we do hope to improve results compared to assuming a static slack bus, this algorithm is not an attempt to model the effects of either droop or AGC control as realistically as possible. The algorithm’s advantage lies in its simplicity. Importantly, differences in network losses cannot be modeled using this algorithm.
Fig. 80 Definition of probabilistic generator control.

Fig. 80 defines generation control for generators that fit into the participating fuel type and minimum size criteria. Additionally, the algorithm is implemented to re-dispatch generation based on either global, zonal, or area-based imbalances.

Fig. 81 Effects of probabilistic re-dispatching on the real slack bus power.

Fig. 81 shows the effects of a global probabilistic AGC algorithm that is defined in addition to the prior studied wind power correlations. We see that our algorithm reduces global imbalances to a point where less imbalances than in the originally studied uncorrelated base case are encountered. Notably, to converge difficult samples that include probabilistic AGC re-dispatching, the sample continuation method must be adapted to include a smooth transition from a generator’s original power dispatch to the updated set-point.
Fig. 82 Effects of probabilistic re-dispatching on the reactive slack bus power.

Compared to Fig. 81, Fig. 82 shows similar narrowing of the reactive slack bus powers back towards the baseline case. However, this re-dispatching algorithm that is simply based on real power changes reduces the optimality of the reactive power distributions in the system. This can be seen by a shift in the mean reactive slack bus power towards a higher value.

Fig. 83 Extreme voltage magnitudes and angles of probabilistic studies including wind correlations and both wind correlations and the global probabilistic re-dispatching algorithm. Zoomed in versions are shown in yellow boxes.
Fig. 83 shows extreme voltage magnitudes and angles of the three presented studies. We see very narrow distributions for the maximum voltage magnitudes and angles. To enable a look at the tails of these distributions, we show zoomed in versions of these distributions in yellow sub-boxes. We see a widening in each variable’s respective distribution for wind correlations. The probabilistic re-dispatching algorithm reduces this widening for each variable. Interestingly, we see a slight shift in the voltage angle minimum distributions, which is in agreement to the increased reactive slack bus power mean of Fig. 82.
4.3 State Estimation

Our SUGAR C++ implementation features a State Estimator that is enabled by ECF measurement models for RTUs and PMUs. An SE problem utilizes the same ECF network models as our PF formulation. Network models utilized in the SE algorithm are transmission line models, transformers, and shunts. Further, a measurement is introduced into the problem via an RTU or PMU model on the measured system bus. Now, we utilize our adjoint circuit-based optimization algorithm to minimize measurement errors and find a state estimate. Since this algorithm is linear, no NR algorithm is required. Hence, we can guarantee a solution to every state estimation sample. To evaluate the quality of these solutions, we compare them to a WLS state estimator.

For this evaluation, synthetic measurement data based on a PF base case is created. To create the synthetic measurements, we first solve the PF base case and further interpret the PF solution as the “true” system state. We denote the vector of complex system state-voltages as $\bar{X}$. The algorithm first determines the type of measurement on a bus by randomly assigning RTUs or PMUs to buses. Herein, the share of total RTU and PMU measurements in the system is determined by input statistics. Furthermore, these statistics define the assumed measurement errors that are superimposed on measurements that would result from the “true” state $\bar{X}$. Importantly, if multiple SE examples are created by this algorithm, the RTU and PMU bus assignments remain unchanged; only their synthetically created measurement values change.

The quality of an SE result is examined using the following measures:

$$x_\sigma = (\hat{X} - \bar{X})^T (\hat{X} - \bar{X})$$  \hspace{1cm} (83)

$$x_{max} = \max |\hat{X} - \bar{X}|$$  \hspace{1cm} (84)

$$r_z = (z - h(\hat{X}))^T (z - h(\hat{X}))$$  \hspace{1cm} (85)

Where $x_\sigma$ is the sum of squared errors over the real and imaginary voltage state variables, $x_{max}$ is the maximum voltage vector deviation, and $r_z$ is the sum of squared measurement residuals of the optimization.
4.3.1 **Comparison against Static AC – WLS State Estimator**

We compare the deterministic ECF based SE algorithm with the Matpower implementation of a traditional AC WLS SE algorithm [112] that was adapted to include current injection measurements on each bus. Unfortunately, phasor measurements are generally not part of the WLS formulation. However, our current formulation requires at least one phasor measurement to set the system’s reference angle. To enable a valid comparison between the two algorithms, we select a zero-injection node to set the reference angle of the system. This eliminates the influence of the PMU’s current phasor measurement for this comparison.

A second difference between the two formulations is how the weights are applied to different measurements. The WLS algorithm weighs every single measurement value by the matrix $W$ that scales its influence within the objective (4). Our formulation does not include every measurement in the objective function, but maps them into models that are further used in the optimization. Hence, only weighting of measurement models as a whole is possible. In order to be conservative in this comparison, we weigh individual measurements within the WLS estimator by the inverse of their standard deviations, whereas within the ECF based algorithm we weigh each measurement model equally.

Table 8: ECF SE Performance measures for different systems

<table>
<thead>
<tr>
<th></th>
<th>$x_\alpha \pm c_{1%}$</th>
<th>$x_{max} \pm c_{1%}$</th>
<th>$r_\gamma \pm c_{1%}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>6.50e-3</td>
<td>8.00e-3</td>
<td>1.14e-2</td>
</tr>
<tr>
<td>#2</td>
<td>3.80e-3</td>
<td>4.60e-3</td>
<td>7.30e-3</td>
</tr>
<tr>
<td>#3</td>
<td>2.25e-2</td>
<td>2.76e-2</td>
<td>0.97e-2</td>
</tr>
<tr>
<td>#4</td>
<td>5.92e-2</td>
<td>7.16e-2</td>
<td>1.31e-2</td>
</tr>
<tr>
<td>#5</td>
<td>1.04e-1</td>
<td>1.26e-1</td>
<td>1.27e-2</td>
</tr>
</tbody>
</table>

Table 8 shows performance measures of the ECF based SE algorithm for five openly available power system test cases [103], [104]. Synthetic sets of measurements were created using the methodology described earlier in this section with the following statistics: RTUs have 1% normally distributed real and reactive power injection measurement uncertainty, and 0.4% voltage magnitude measurement uncertainty. The single PMU that sets the reference angle within the

---

1 #1: IEEE-118 bus system; #2: case_ACTIVSg500; #3: 1888-bus RTE model; #4: case_ACTIVSg2000; #5: 6515-bus RTE model.
ECF is assumed to be a perfect measurement. Its source conductance $G_{PMU}$ is set to 10 p.u. All uncertainty definitions are based on a measure’s “true state”.

To perform a valid comparison between the two algorithms, we ran multiple SE samples until we reached an accuracy such that at least one performance measure ((83)-(85)) was within ±5% of its mean with 99% confidence. On average, 544 instances were run for each case.

Table 9 WLS SE performance measures for different systems

<table>
<thead>
<tr>
<th></th>
<th>$x_d \pm c_{99%}$</th>
<th>$x_{max} \pm c_{99%}$</th>
<th>$r_2 \pm c_{99%}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>3.60e-3</td>
<td>7.20e-3</td>
<td>2.20e-3</td>
</tr>
<tr>
<td>#2</td>
<td>2.30e-3</td>
<td>5.40e-3</td>
<td>8.90e-3</td>
</tr>
<tr>
<td>#3</td>
<td>1.07e-2</td>
<td>7.10e-3</td>
<td>3.54e-2</td>
</tr>
<tr>
<td>#4</td>
<td>2.44e-2</td>
<td>7.70e-3</td>
<td>3.58e-2</td>
</tr>
<tr>
<td>#5</td>
<td>6.33e-2</td>
<td>8.40e-3</td>
<td>1.25</td>
</tr>
</tbody>
</table>

From Table 9 we see slightly better results for the nonlinear WLS SE formulation as compared to our linear ECF based State Estimator (Table 8). Notably, this is under conditions that are designed to compare a best-case scenario for the WLS, where weighting can be applied due to good knowledge of uncertainties, and possible divergence is avoided by good initial conditions. Different to the nonlinear WLS approach, our formulation guarantees a solution due to its linearity and is superior to the WLS algorithm in terms of computational complexity. This enables a probabilistic approach to SE, resulting in a more complete (probabilistic) picture of a system’s state.

4.3.2 **Linear Probabilistic State Estimation**

State estimators operate under hard time constraints. However, our linear formulation with its connected low computational complexity suggests a potential for a probabilistic approach to SE. Numerical approaches to probabilistic analysis are computationally intensive. Hence, only a few probabilistic SE approaches for special applications have been proposed [114],[115] so far. However, probabilistic approaches to general PS analysis have many advantages. According to [114], advantages of a probabilistic SE approach are reduced standby resources and better ability to quantify voltage variabilities with intermittent renewable generation. [114] also discusses the possibility of probability-based market products. In addition, the motivation for probabilistic analysis discussed by Makarov [3] leads us to think that a probabilistic SE could play a major part
in comprehensive probabilistic PS analyses. For example, probabilistic SE naturally leads to probabilistic contingency analysis for improving risk-awareness.

Our MC framework enables us to readily evaluate this idea. We study this probabilistic SE algorithm on the 1888-bus RTE system model of the French transmission grid that is openly available [104]. Importantly, this system size is a realistic abstraction of a regional size grid operator in the US and enables us to get a realistic picture of our algorithm on a system of the size of common real-world models. NYPA’s EMS model, for example, comprises approximately 1600 buses [116].

<table>
<thead>
<tr>
<th>Network uncertainties</th>
<th>$\sigma_R$ [% of mean]</th>
<th>$\sigma_x$ [% of mean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission line</td>
<td>5%</td>
<td>0.5%</td>
</tr>
<tr>
<td>Transformer</td>
<td>1%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

We test our algorithm in the following way: We synthetically create a single set of measurement data, where 10% of system buses are randomly selected to be PMU measured. Of these PMUs, 40% are assumed to be perfect measurements. The remaining 60% of PMUs have 0.02% uncertainty for all their measures (i.e. real and imaginary currents and voltages). All other buses are RTU measured with RTU uncertainties identical to the previous experiment. Additionally, we add network model uncertainties that are defined in Table 10. Importantly, only a numerical probabilistic approach to SE is capable of including the evaluation of network uncertainty effects on the estimate of state.
Fig. 84 Selected probabilistic density functions (PDFs) of our SE algorithm for the 1888-bus RTE case, with (blue) and without (green) including network uncertainties, and the true state of the system.

Fig. 84 shows selected results from a probabilistic SE simulation with one million samples. The probabilistic density functions (PDFs) in Fig. 84 depict the voltage magnitude and angle of bus 1337 and the real and reactive branch flow on the branch that connects buses 1337 and 311, which has one of the highest loadings in the system. The (PDFs) without network uncertainties are shown in green, whereas the PDFs with network uncertainties are shown in blue. The true (synthetic) system state $\bar{X}$, shown as a red-dotted line, is within the distributions for all PDFs. We observe low influences of network uncertainties for voltage angle and real power in Fig. 84, however, standard deviations of the voltage magnitude and reactive power distributions were increased by 4.95% and 21.3%, respectively.

Interestingly, the probabilistic algorithm allows us to make statements about probability of states. This is not possible in a deterministic setting. We find, for example, that the probability of an absolute value of real power flow on the presented branch of greater than 15 p.u. is 0.244%. Alternatively, we find that there is less than a 1% chance for the real power flow on this branch.
to be higher than 14.73 p.u. (with a 99% confidence interval of ± 0.0023 p.u.). Statements of this sort are a valuable addition for system awareness and can lead to improved risk awareness, which is especially pertinent in modern grids with ever-increasing uncertainties and reduced margins of error.

The algorithm to estimate confidence intervals of percentiles in MC solutions is based on order statistics and Binomial distributions [37]. Estimated confidence intervals are an important measure to characterize the quality of MC results. These algorithms can either be used to approximate the amount of necessary MC samples prior to running the simulation, or they can be calculated during the simulation to finalize the algorithm when the required accuracy is reached.

4.3.3 State Estimation samples up to EI

We next present a probabilistic SE study on the synthetic USA transmission power system case [103] with 82k buses. As in the previous examples, we assume that 10% of system buses are observed by PMUs, such that 40% are perfect measurements and 60% have 0.02% normally distributed measurement uncertainty \((V_{PMU}^{RJ}, I_{PMU}^{RJ})\). Furthermore, RTUs have 1% normally distributed measurement uncertainty for their real and reactive power injections and 0.4% for their voltage magnitude measurement. With this, a deterministic SE case based on the synthetic USA system [103] is created and further evaluated using the linear probabilistic SE algorithm. For each experiment, \(10^4\) Monte Carlo samples were created by sampling from the base SE case’s uncertainty distributions.
Fig. 85 Voltage magnitudes of four select buses of a synthetic State Estimation case on an 82k bus system.

Fig. 86 Selected voltage angles of the 82k bus synthetic State Estimation case.

Fig. 85 shows selected bus voltage magnitudes and Fig. 86 displays the corresponding voltage angles. Unsurprisingly, the distributions change notably with the distance of the observed bus
from the closest PMU measurement. For Fig. 85 and Fig. 86, the closest PMU-measured buses are: (a) 4 hops away, (b) 5 hops away, (c) one hop away, and (d) at the chosen bus. The difference in the SE comparisons between (c) and (d) in Fig. 85 and Fig. 86, however, is not due to the one hop difference in distance from the closest PMU measurement bus, but due to the difference between imperfect and perfect PMU measurements.

4.3.4 Linear probabilistic SE performance considerations

The linear probabilistic algorithm of our SUGAR C++ MC framework solved the $10^6$ Monte Carlo samples of the 1888-bus case in 7 minutes and 9 seconds on an Intel Xeon server CPUs (E5-2680) running on 2.40GHz using 45 of 56 possible threads. Using the same setup, the $10^4$ samples of the 82k bus case were solved in 13 minutes. Both of those times are close to or within market intervals, suggesting the algorithm’s suitability for real-world application. However, to reach the same quality for both studied cases, clearly more samples are required for the 82k bus system. However, for a centralized probabilistic SE that covers a whole continent, funding for the necessary computing power is likely available. This is true because this algorithm is expected to scale linearly with additional CPUs due to the independence between Monte Carlo samples of this algorithm.

Importantly, for this algorithm to be used in real world interconnection systems, additional challenges concerning data communication, pre- and post-processing, arise. On the input side of this algorithm, the measurements must be reliably communicated across vast distances, and on the output side probabilistic results that are encoded in Gigabytes to Terra-bytes of data have to be evaluated. These are big, but not unsurmountable, challenges. Modern communication systems enable reliable and secure communication of vast amounts of data across the globe, and to process the results the field of big-data science developed tools that are capable of handling comparable payload. For example, an Apache Spark [117] based-algorithm could be utilized to create pdfs of bus states or line flows. Alternatively, similar output statistics could be created in a parallel and distributed manner.
4.4 Stochastic Optimization to Identify “true” System States

Our linear probabilistic approach to SE sparked the idea of a stochastic optimization to identify “true” grid states. We can identify a “true” grid state by a minimal (zero) objective function value of a probabilistic SE sample. Since our objective function is a measure of measurement error, a zero objective function means that no measurement errors were encountered for this sample. This identifies a “true” grid state under the assumption of a perfect topology model.

We can now define a probabilistic algorithm to identify these states. We do that by invoking the probabilistic SE algorithm. Then, samples with minimal objective functions are filtered and stored as candidates for samples that identify a true state of the grid.

We tested this algorithm on the IEEE 14-bus test case. For this, we synthetically created an SE sample by randomly assigning a single PMU to one of the 14 buses. The remaining buses were assumed to be measured by RTUs. We assumed measurement uncertainties for PMUs to have 0.02% normally distributed measurement uncertainty \((v_{PMU}^P, v_{PMU}^Q)\), and RTUs to have 1% normally distributed measurement uncertainty for their real and reactive power injections, and 0.4% for their voltage magnitude measurement.
Fig. 87 Real and imaginary voltages of the IEEE 14-bus case identifying “true” grid states.

Fig. 87 shows results of this algorithm for the IEEE 14-bus test case, where we filter results for the lowest 1000 minimal objective function values out of 1 million Monte Carlo samples. In Fig. 87, the original PF solution of the case is demarked by a red vertical line and the “best” 1000 samples are plotted as a histogram. We see a clustering around the “true” grid states for all system variables, which suggests that this algorithm was able to identify the “true” grid states (based on assumption of a perfect topology for this example).
5 Conclusion and Future Work

This thesis describes background and implementation of the Power System simulator SUGAR (Simulation with Unified Grid Analyses and Renewables) in C++ that is suitable for probabilistic power flow analysis. This simulator builds on the equivalent circuit formulation, which formulates Power System problems in their real state variables, currents, and voltages. This facilitates physically correct modeling of grid components and enables the use of circuit simulation techniques to develop and implement robust solution methods.

In addition to solving the traditional Power Flow problem, our implementation can solve optimization problems that are modeled in terms of equivalent circuits. One example is an optimization algorithm that identifies Power Flow feasibility, which is needed for our State Estimation implementation that is built upon a fully linear optimization-based State Estimation algorithm that utilizes linear ECF-based RTU and PMU models.

The framework for probabilistic Power System analyses is based on a Monte Carlo approach. The framework can be used for probabilistic Power Flow and probabilistic State Estimation. Data structures are implemented that enable studies involving variable correlations within and across models.

Presented results include those from the algorithm implementations and probabilistic PF studies. The algorithms include an adaptive step-size Tx-stepping implementation that is optimized to robustly solve Eastern Interconnection planning cases with around 78k buses as efficiently as possible. Furthermore, we present a probabilistic sample continuation algorithm that enables robust convergence of probabilistic samples and outperforms the Tx-stepping algorithm within Monte Carlo sample evaluation due to the utilization of additional available information.

The probabilistic power flow algorithm is presented by conducting a probabilistic study of an Eastern Interconnection planning case. And studying slack bus power injections and tie-line flows between grid operators. Furthermore, we presented correlation effects and an approach for
probabilistic generation control modeling utilizing a 2000-bus synthetic grid model on the footprint of Texas.

The ECF based State Estimation algorithm is first compared to a traditional WLS State Estimator. Further, utilizing our SE algorithm’s linearity, we proposed and studied a probabilistic SE algorithm on different cases up to a study on a synthetic 82k-bus system. Finally, we described a stochastic optimization algorithm to identify “true” system states based on minimal optimization objectives.

One area of focus for future work for probabilistic ECF-based Power Flow is to improve modeling of generation control algorithms. Depending on the time frame of a probabilistic study, different control algorithms should be implemented. For short target time-scales, the implementation of realistic droop and AGC controls would greatly improve results. However, for probabilistic studies concerned with timeframes that stretch across market intervals, stochastic Optimal Power Flow algorithms are preferred.

Further improvements include improved modeling of uncertainty distributions. These are beta distributions for loads [118] and Weibull or Rayleigh based distributions for wind generations. Additionally, realistic modeling of correlations within the system would greatly improve results. Unfortunately, while all of these are simple to implement from a software development perspective, all improvements are contingent upon the availability of data that enables realistic assumptions.

Developments within our Monte Carlo framework could include exploring variance reduction techniques to reduce computational burden of these algorithms. Similarly, methods from statistical theory could improve results for special cases. For example, techniques resulting from Extreme Value Theory or Large Deviation Theory could prove to be helpful for extreme event analysis.

Finally, interesting practical problems arise when dealing with probabilistic data. Important areas of development could be pre- and post-processing techniques to tackle the vast amounts of created data. This could be solved by employing big-data techniques. Also, finding ways to efficiently visualize and present probabilistic output to grid operators could prove to be a milestone in the adoption of probabilistic methods in this field.
REFERENCES


 Appendix A: A JSON-based SUGAR input file

```json
{
    "input-file" : "../cases/case_ACTIVSg2000.m",
    "output-file" : "",
    "write-raw-file" : "",
    "write-line-loadings" : "",
    "write-generator-data" : "",

    "sugar-config" : {
        "create-log-file" : false,
        "solver-config" : {
            "run-feasibility" : false,
            "max-iterations" : 1e3,
            "nr-max-itors" : 15,
            "epsilon" : 1e-6,
            "voltage-limit" : 0.1,
            "max-voltage" : 2.0,
            "min-voltage" : 0.1,
            "var-limiting" : false,
            "no-nr-loop" : false,
            "loading-factor" : 1.0,
            "ignore-bus-labels" : false,
            "zbr-cutoff-value" : 0.00028,
            "tx-stepping-settings" : {
                "max-tx-stepping-itors" : 10,
                "forward-scaling-factor" : 1.25,
                "backward-scaling-factor" : 0.5,
                "tx-step" : 0.1,
                "tx-step-up" : 2.0,
                "tx-step-down" : .2,
                "remote-x-value" : 0.001,
                "pick-random-generator-lines" : 0,
                "G-cont" : 40.0,
                "B-cont" : 40.0
            },
            "q-limiting-settings" : {
                "enforce-q-limits" : true,
                "limit-one" : false,
                "limit-biggest-q-first" : false
            },
            "dc-line-settings" : {
                "dc-xfmr-control" : false,
                "converge-dc-lines" : false,
                "no-converge-dc-lines" : false
            }
        },
        "monte-carlo" : {
            "samples" : 100,
            "threads" : 1,
            "ci-99-limit" : 0.0,
            "ci-95-limit" : 0.0,
            "output" : "mc_out",
```
"prng-seed" : "6364136223846793005 1442695040888963407 5573589319906701683",
"rerun-samples" : ",",
"lock-random-number-generation" : false,
"use-probabilistic-continuation" : true,
"record-solutions" : false,
"max-solutions-per-file" : 10,
"record-buses" : [],
"probabilistic-agc" : {
  "on" : true,
  "participating" : "coal:hydrop:ng:nuclear",
  "min-size" : 0.0,
  "control" : "area"
},
"state-estimation" : {
  "run-state-estimation" : false,
  "create-data-from-pf" : true,
  "samples" : 1,
  "rel-ci99-max" : 0.05,
  "use-nonlinear-rtu" : false,
  "pmu-weighting-factor" : 10,
  "use-weighting" : false
},
"data" : {
  "probabilistic" : {
    "global" : {
      "intermodel-correlations" : {
        "branch" : {
          "Gser" : {
            "distribution" : "normal",
            "stddev" : 0.1,
            "correlation" : {
              "Bser" : 0.1
            }
          },
          "Bser" : {
            "distribution" : "normal",
            "stddev" : 0.1
          }
        },
        "xfmr" : {
          "Rloss" : {
            "distribution" : "normal",
            "stddev" : 0.0
          }
        }
      }
    }
  }
}
"Xloss": {
  "distribution": "normal",
  "stddev": 0.0
},
"load":{
  "P": {
  },
  "Q": {
  }
},
"gen":{
  "P": {
  }
},
"gen:wind":{
  "P": {
    "distribution": "normal",
    "stddev": 0.01
  }
},
"bus": [ {
  "id": 2,
  "data": {
    "load": {
      "P": {
        "distribution": "normal",
        "stddev": 1.1
      }
    }
  }
},
"zone": [ {
  "id": 2,
  "data": {
    "branch": {
      "Gser": {
        "distribution": "normal",
        "stddev": 0.10
      },
      "Bser": {
        "distribution": "normal",
        "stddev": 0.1
      }
    },
    "xfmr": {
      "Rloss": {
        "distribution": "normal",
        "stddev": 0.0
      },
      "Xloss": {
        "distribution": "normal",
        "stddev": 0.0
      }
    }
  }
}]}
"load": {
  "P": {
    "distribution": "normal",
    "stddev": 0.1
  },
  "Q": {
  }
}
"id": 1,
"data": {
  "branch": {
    "Gser": {
      "distribution": "normal",
      "stddev": 0.20
    },
    "Bser": {
      "distribution": "normal",
      "stddev": 0.2
    }
  },
  "xfmr": {
    "Rloss": {
      "distribution": "normal",
      "stddev": 0.0
    },
    "Xloss": {
      "distribution": "normal",
      "stddev": 0.0
    }
  },
  "load": {
    "P": {
      "distribution": "normal",
      "stddev": 0.2
    },
    "Q": {
    }
  }
}
"state-estimation": {
  "accurate-PMUs": 0.04,
  "non-accurate-PMUs": {
    "quantity": 0.06,
    "vr-uncertainty": 0.0002,
    "vi-uncertainty": 0.0002,
    "ir-uncertainty": 0.0002,
    "ii-uncertainty": 0.0002,
    "distribution": "normal"
  }
}
"RTUs" : {
    "vm-uncertainty" : 0.004,
    "p-uncertainty" : 0.01,
    "q-uncertainty" : 0.01,
    "distribution" : "normal",
    "stale-measurements" : {
        "quantity" : 0.0,
        "weight" : 10.0
    }
}
}