# A Modular and Efficient Modeling Library for Power Flow Network Steady-State Analysis and Optimization

Martin Baltzinger
Department of Information Technology
Swiss Federal Insitute of Technology
Zurich, Switzerland
Email: martbalt@student.ethz.ch

Tomás Tinoco De Rubira
Grid Operations and Planning
Electric Power Research Institute
Palo Alto, California, USA
Email: ttinoco@epri.com

Adam Wigington
Grid Operations and Planning
Electric Power Research Institute
Palo Alto, California, USA
Email: awigington@epri.com

***Abstract*—Computational software for steady-state analysis and optimization of power networks is critical. Unfortunately, advances in computational technologies often take a long time to reach industry applications or never get adopted at all. This is partly due to the vast but fragmented spectrum of computational software used by the community, and the lack of an efficient, flexible, and accessible tool that facilitates the transition of new techniques and algorithms into tools that can be tested and used by power system engineers and practitioners. With the goal of filling this gap, a new open-source library has been developed. An overview of this library is presented here along with an example and experimental results that demonstrate its use, performance, and accuracy.**

## I. Introduction

In power network operations and planning, computational software are key. These software are at the core of many applications that are essential for making power networks operate reliably and efficiently. For example, computational software are used for solving Power Flow (PF) problems in order to determine system state and assess system security. They are used for solving Optimal Power Flow (OPF) problems to determine economic and safe generator dispatches. They are also used for determining generator schedules, or Unit Commitments (UC), to clear power markets, and to perform many other critical tasks. Due to this critical role, advances in computational software speed, robustness, and flexibility are in general tightly coupled with significant improvements in various measures of power network performance [1].

The computational software used by the power system community for analysis and optimization is vast. Independent System Operators (ISO) and Regional Transmission Organizations (RTO) rely on commercial power systems software, which ranges from Energy Management Systems (EMS) software to standalone applications. These software are user-oriented and aside from the essential visualization and data-management capabilities, they also provide core computational tools for performing common tasks. The "building blocks" provided by these software are in general very high-level: they constitute tasks such as solving PF or OPF problems, performing contingency analysis, etc, which can be configured to a relatively limited degree. On the other hand, the academic and research community for the most part utilizes a different set of computational software for performing studies and prototyping new algorithms or techniques. A very popular tool is MATPOWER [2], which runs on MATLAB and Octave. The main building blocks provided by this tool are similar to those of commercial software, *e.g.*, PF and OPF solvers, but with a richer capability of customization of the core problem formulations. For example, MATPOWER allows users to extend the core OPF problem formulation to have additional variables, linear constraints, and objective functions. Other commonly-used software by the power systems research and academic community are general-purpose optimization modeling frameworks such as JuMP [3] and CVXPY [4]. Since these tools are general-purpose, the modeling building blocks they provide are much lower level. In particular, they require specifying variables and forming mathematical expressions using a typically extensive library of built-in mathematical functions such as absolute value, Euclidean norm, matrix trace, and many others. A key feature that makes these software suitable for fast prototyping is the ability to transform optimization problem descriptions into the various standard forms used by optimization solvers. This frees users from having to perform this tedious, time-consuming, and error-prone task. Lastly, some academics and researchers opt to work directly with numerical solvers such as KNITRO [5], SNOPT [6], or IPOPT [7]. Working with these low-level tools directly leads to the highest flexibility and performance, but requires expertise and a significant time investment.

Despite this vast spectrum of software, leveraging advances in computational techniques for industry applications is in general challenging due to the fragmentation that exists between the aforementioned parts of the power system community and the corresponding software used. In particular, a flexible and efficient tool is missing that is easily accessible to everyone, from students to power system practitioners and engineers, that can serve as a common framework for facilitating the evaluation and adoption of new computational technologies. Motivated by this need, a new modeling library has been developed. This library, which is called PFNET, has been designed for modeling optimization problems that arise in power networks and for performing core steady-state analyses. In particular, the library has been designed to be modular so that users can add new data parsers, objective functions, and constraints without affecting the core framework. It has been implemented in C for efficiency and it has a Python wrapper that exposes the complete Application Programming Interface (API). The building blocks provided by the library for modeling network optimization problems are lower-level and more flexible than those provided by available power systems software such as MATPOWER, and higher-level than those provided by general-purpose modeling frameworks. In particular, instead of working with high-level tasks such as solving PF and OPF problems, or working with variables and elementary functions such as norms, absolute values, and matrix traces, users of PFNET work with power network quantities such as voltage magnitudes and generator powers, power network functions such as generation cost, voltage regularization, and sparsity-inducing control penalties, and power network constraints such as power balance, voltage regulation, and generator ramping limits. Lastly, the performance and accuracy of this library have been validated against other software such as MATPOWER and commercial power system software.

This paper is organized as follows: In Section II, power networks are described and mathematical notation is introduced. In Section III, the PFNET library is introduced and its key features are described. In Section IV, an example that shows how to construct power network optimization problems with PFNET in Python is provided. In Section V, experimental results comparing the performance and accuracy of PFNET and MATPOWER are shown. Lastly, in Section VI, the work is summarized and important future features of PFNET are discussed.

## II. Power Networks

A brief overview of power networks is provided, with the main goal being to set the stage and introduce important notation that will be used in subsequent sections.

Power networks are composed of buses indexed by $\mathcal{N}$, branches indexed by $\mathcal{B}$, generators indexed by $\mathcal{G}$, loads indexed by $\mathcal{L}$, and other components. Each bus $k \in \mathcal{N}$ has a voltage magnitude $v_k$ and a voltage angle $\theta_k$. The voltage magnitude typically has to be within limits $v_k^{\min}$, $v_k^{\max}$ in order to prevent equipment damage. Each branch can be either a transmission line or a transformer. If a branch $(k,m) \in \mathcal{B}$ is a transformer, it can be a tap-changer, in which case it can adjust its taps ratio $t_{km}$, or it can be a phase-shifter, in which case in can adjust its phase shift $\phi_{km}$. A branch $(k,m) \in \mathcal{B}$ has thermal limits, which can be expressed by the condition $|i_{km}| \leq i_{km}^{\max}$, where $i_{km}$ denotes the branch current [8]. Each generator $k \in \mathcal{G}$ injects active and reactive powers $P_k^g \in [P_k^{\min}, P_k^{\max}]$ and $Q_k^g \in [Q_k^{\min}, Q_k^{\max}]$, respectively, into the bus to which it is connected. Similarly, each load $k \in \mathcal{L}$ consumes active and reactive powers $P_k^l$ and $Q_k^l$, respectively, from the bus to which it is connected. We assume for simplicity that each bus has one generator and load connected to it. Lastly, each shunt device can be fixed or switching. If a shunt device $k \in \mathcal{S}$ is switching, it can adjust its susceptance $b_k$.

Some of the key aspects of power networks that make them challenging and interesting to optimize are conservation of power, voltage regulation mechanisms, operational limits, and economic forces. In particular, conservation of power leads to the well-known power flow equations, which are nonlinear equality constraints given by

$$P_k^g + P_k^l - \sum_{m \in \mathcal{N}} v_k v_m (G_{km} \cos\theta_{km} + B_{km} \sin\theta_{km}) = 0$$
$$Q_k^g + Q_k^l - \sum_{m \in \mathcal{N}} v_k v_m (G_{km} \sin\theta_{km} - B_{km} \cos\theta_{km}) = 0,$$

for all $k \in \mathcal{N}$, where $\theta_{km} := \theta_k - \theta_m$, and $G$ and $B$ are the real and imaginary parts, respectively, of the network admittance matrix [9]. Bus voltage regulation can be provided by several network components, including generators, tap-changing transformers, and switching shunt devices. Some generators, for example, can adjust their reactive power output to try to keep a bus voltage magnitude at a set-point. Mathematically, these generator controls can be modeled as

$$\begin{aligned} &v_k = v_k^s + v_k^y - v_k^z \\ &0 \leq (Q_k^g - Q_k^{\min}) \perp v_k^y \geq 0 \\ &0 \leq (Q_k^{\max} - Q_k^g) \perp v_k^z \geq 0, \end{aligned}$$

for all buses $k \in \mathcal{N}$ that are regulated by generators, where $v_k^s$ denotes voltage set-point, $v_k^y$ and $v_k^z$ denote positive and negative deviations, respectively, of $v_k$ from $v_k^s$, and the symbol $\perp$ denotes complementarity, which is defined as follows: $a \perp b \iff ab = 0$ [9]. Other forms of voltage regulation, *e.g.*, voltage-band regulation by tap-changers and switching shunt devices, can be represented with similar, albeit slightly more complex constraints [10]. Operational limits of the network include the aforementioned limits on bus voltage magnitudes and branch current magnitudes, which lead to the linear and nonlinear inequality constraints

$$\begin{aligned} &v_k^{\min} \leq v_k \leq v_k^{\max}, \ \forall k \in \mathcal{N} \\ &|i_{km}| \leq i_{km}^{\max}, \ \forall (k,m) \in \mathcal{B}. \end{aligned}$$

Lastly, economic forces provide encouragement for operating power networks at a minimum cost. With regards to this, a widely-used model for total generation cost is given by the quadratic function

$$\sum_{k \in \mathcal{G}} a_k (P_k^g)^2 + b_k P_k^g + c_k,$$

where $(a_k, b_k, c_k) \in \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R}$ [8].

These key aspects of power networks appear in some of the most important optimization problems that need to be solved in practice, namely PF and OPF. The PF problem consists of determining the system state given system loading conditions, modeling all voltage-regulation controllers. On the other hand, the OPF problem typically consists of determining system state and new generator dispatches in order to minimize cost while ensuring that the network remains within operational limits. Both of these core problems can be expressed or closely approximated by problems of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) \\ \text{subject to} \quad & Ax = b \\ & f(x) = 0 \\ & l \leq Gx \leq u, \end{aligned} \tag{1}$$

where $\varphi$ and $f$ are general smooth scalar-valued and vector-valued functions, respectively.

## III. Modeling Library

The PFNET library allows modeling power networks over multiple time periods and constructing network optimization problems of the form of (1). The following subsections provide relevant details. These details are reflective of version 1.3.2 of the library. For future versions, these may be slightly different.

### A. Architecture

The PFNET library has four main parts: network, optimization, parsing, and external features. The network part consists of data structures and routines for storing and manipulating network data over multiple time periods, including topology, state, and configuration, *e.g.*, outage flags, voltage regulation information, etc. Aside from the network components mentioned in Section II, PFNET networks can also include variable generators, *e.g.*, wind farms and photovoltaic plants, and batteries, which are not be discussed here. The optimization part consists of a collection of power network functions and constraints that can be used independently or together to form optimization problems of the form of (1). The parsing part consists of data parsers or interfaces that can translate different power network data files to and from PFNET networks. Lastly, the external features part consists of optional components of PFNET that perform specialized tasks using external libraries, such as creating layouts for large networks using Graphviz [1], or creating conservative linear branch thermal constraints [8]. Figure 1 illustrates the architecture of the library.

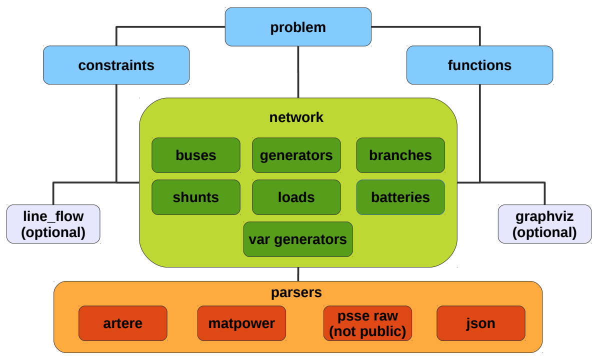

Fig. 1. PFNET architecture

[1]http://www.graphviz.org/

### B. Variables

PFNET can be used to specify quantities of the network that are to be considered optimization *variables*. Some of these quantities are bus voltage magnitudes $v_k$ and angles $\theta_k$, branch tap ratios $t_{km}$ and phase shifts $\phi_{km}$, generator active powers $P_k^g$ and reactive powers $Q_k^g$, load active powers $P_k^l$ and reactive powers $Q_k^l$, and shunt susceptances $b_k$. For each type of quantity, all or some can be considered variables. Quantities configured to be variables can be further specified to remain fixed at their current value, or to remain bounded within their associated limits, if any.

### C. Functions

*Functions* in PFNET are mappings of the form $\varphi : \mathbb{R}^n \to \mathbb{R}$, where $n$ denotes the number of network variables, *e.g.*, network quantities that have been configured to be optimization variables. Functions can be evaluated at an arbitrary vector $x$ of variable values, and then information such as value $\varphi(x)$, gradient $\nabla\varphi(x)$, and Hessian $\nabla^2\varphi(x)$ at that point can be extracted. The collection of functions available in PFNET include quadratic generation cost, quadratic load consumption utility, sparsity-inducing control penalty, various regularization penalties, and others.

### D. Constraints

*Constraints* in PFNET are of the form

$$A\begin{bmatrix} x \\ y \end{bmatrix} = b, \quad f(x,y) = 0, \quad l \leq G\begin{bmatrix} x \\ y \end{bmatrix} \leq u,$$

where $x \in \mathbb{R}^n$ are network variables, $y \in \mathbb{R}^{\bar{n}}$ are extra or auxiliary variables of the constraint, and $f$ is a general smooth vector-valued function. The extra or auxiliary variables are used to increase the variety of constraints that can be expressed in the chosen standard form. For example, they are slack variables in the case of branch thermal limits, and they are the $v^y$ and $v^z$ variables in the case of the voltage regulation constraints described in Section II. Constraints can be evaluated at an arbitrary vector $(x,y)$ of variable values, and then information such as $f(x,y)$, $J(x,y)$ and $\nabla^2(\alpha^T f(x,y))$ can be extracted, where $J$ denotes the Jacobian matrix of $f$, and $\alpha$ is an arbitrary vector of suitable size. The collection of constraints available in PFNET include AC and DC power balance equations, voltage regulation by generators, tap-changing transformers, and switching shunt devices, branch thermal limits, general ramping limits, variable bounds, and others.

### E. Problems

*Problems* in PFNET are of the form (1). They are constructed by defining network variables and selecting a collection of functions indexed by $\mathcal{F}$ and constraints indexed by $\mathcal{C}$. The resulting problem optimization variables in (1) are given by $x = (x_n, y_1, \ldots, y_{|\mathcal{C}|})$, where $x_n$ are network variables, $y_i$ are extra or auxiliary variables of constraint $i \in \mathcal{C}$, and $|\cdot|$ denotes set cardinality. The objective function $\varphi$ is given by

$$\varphi(x) = \sum_{i \in \mathcal{F}} w_i \varphi_i(x_n), \ \forall x,$$

where $w_i$ is a scalar weight, and $\varphi_i$ is function $i \in \mathcal{F}$. The matrices $A$ and $G$, vectors $b$, $l$, and $u$, and function $f$ of (1) are give by

$$A = \left[A_1^T \cdots A_{|\mathcal{C}|}^T\right]^T \quad G = \left[G_1^T \cdots G_{|\mathcal{C}|}^T\right]^T$$

$$b = \left[b_1^T \cdots b_{|\mathcal{C}|}^T\right]^T \quad l = \left[l_1^T \cdots l_{|\mathcal{C}|}^T\right]^T \quad u = \left[u_1^T \cdots u_{|\mathcal{C}|}^T\right]^T$$

$$f(x) = \left[\, f_1(x_n, y_1)^T \cdots f_{|\mathcal{C}|}(x_n, y_{|\mathcal{C}|})^T \,\right]^T, \ \forall x,$$

where $A_i$ and $b_i$ are the matrix and vector associated with the linear equality constraints of constraint $i \in \mathcal{C}$, $G_i$, $l_i$, and $u_i$ are the matrix and vectors associated with the linear inequality constraints of constraint $i \in \mathcal{C}$, and $f_i$ is the function associated with the general nonlinear equality constraints of constraint $i \in \mathcal{C}$. As with constraints and functions, problems can be evaluated at an arbitrary point $x$, and then information such as $\varphi(x)$, $\nabla\varphi(x)$, $\nabla^2\varphi(x)$, $f(x)$, $J(x)$, and $\nabla^2(\alpha^T f(x))$ can be extracted, where $J$ denotes here the Jacobian of $f$, and $\alpha$ is an arbitrary vector of suitable size.

### *F. Implementation*

The PFNET library has been implemented in C and has a Python wrapper written in Cython that exposes the complete API. The code is available in Github [2] under a BSD license. The PFNET Python module can be conveniently installed using the package management system `pip`, which automatically builds the C library. A tutorial and API documentation is available online through "Read the docs" [3]. The library is well tested and is currently under active development. The continuous-integration service Travis CI [4] is being used for automatically running tests in various platforms.

PFNET uses its own data structures for representing vectors and sparse matrices. In the Python wrapper, vectors are efficiently cast as Numpy arrays while sparse matrices are efficiently cast as Scipy sparse matrices in coordinate form. This makes the PFNET Python module work well together with the core scientific computing libraries in Python. The library has been designed to be modular. In particular, new parsers, functions, and constraints can be "plugged in" either in C or Python. For functions and constraints, users need to provide a few routines that given a sequence of network branches and time periods perform various tasks such as counting number of non-zeros of function or constraint matrices, allocating and storing sparsity patterns, and computing values of functions and their derivatives. Readers interested in implementing new parsers, functions, or constraints are encouraged to look at the online documentation or to contact the authors.

The PFNET library is actually one part of a computational software ecosystem for power flow network analysis and optimization. Other software of this ecosystem are OPTALG [5] and GRIDOPT [6] [11], which are also available under BSD licenses. OPTALG provides optimization solvers based on Augmented Lagrangian and primal-dual interior-point algorithms, and wrappers to other solvers including IPOPT [7]. It has also been designed to work directly with problems of the form of (1). GRIDOPT, on the other hand, is a convenient package that uses PFNET and OPTALG to construct and solve common or standard network optimization problems such as PF and OPF.

[2]https://github.com/ttinoco/PFNET
[3]http://pfnet-python.readthedocs.io
[4]https://travis-ci.org/

## IV. Example

A simple Python example is now presented that shows how to a construct a network from a MATPOWER-converted data file, specify quantities of the network that are to be considered variables and which of these are to be forced to remain bounded, and construct an optimization problem from functions and constraints. The problem constructed, which consists of a single-period OPF problem, is solved with a primal-dual interior-point algorithm from OPTALG:

```python
from pfnet import Function as F
from pfnet import Constraint as C
from pfnet import Problem, ParserMAT
from optalg.opt_solver import OptSolverINLP

net = ParserMAT().parse("ieee300.mat",
                        num_periods=1)

net.set_flags("bus",
              ["variable","bounded"],
              "any",
              "voltage magnitude")
net.set_flags("bus",
              "variable",
              "not slack",
              "voltage angle")
net.set_flags("generator",
              ["variable","bounded"],
              "any",
              ["active power","reactive power"])

prob = Problem(net)
prob.add_function(F("generation cost",1.,net))
prob.add_constraint(C("AC power balance",net))
prob.add_constraint(C("variable bounds",net))
prob.add_constraint(C("AC branch flow limits",net))
prob.analyze()

solver = OptSolverINLP()
solver.solve(prob)
```

[5]https://github.com/ttinoco/OPTALG
[6]https://github.com/ttinoco/GRIDOPT

## V. Validation and Performance

Experimental results are now shown that compare the performance and accuracy of PFNET with respect to MATPOWER for solving AC PF and OPF problems. The PF and OPF problems were constructed using PFNET and the resulting problems were solved using OPTALG. In particular, the Newton-Raphson (NR) solver was used to solve the PF problem, while the Interior-Point Non-Linear Programming (INLP) solver was used to solve the OPF problem. For the PF problem, generator reactive power limits were ignored (to avoid potential interference from PV-PQ switching heuristics). For the OPF problem, branch thermal limits were considered. Figure 2 and Tables I and II show the results obtained. The plots show the normalized time (total execution time divided by the number of iterations) as a function of the number of buses of the test networks used. The tables show the objective value, voltage magnitude, and voltage angle discrepancies between the solutions found with PFNET and MATPOWER. In the notation $\mu \pm \sigma$ used in the tables, $\mu$ and $\sigma$ denote the average and standard deviation, respectively, of a quantity over the network buses. As the results show, the solutions found with PFNET and MATPOWER are in close agreement, and the performance is comparable.

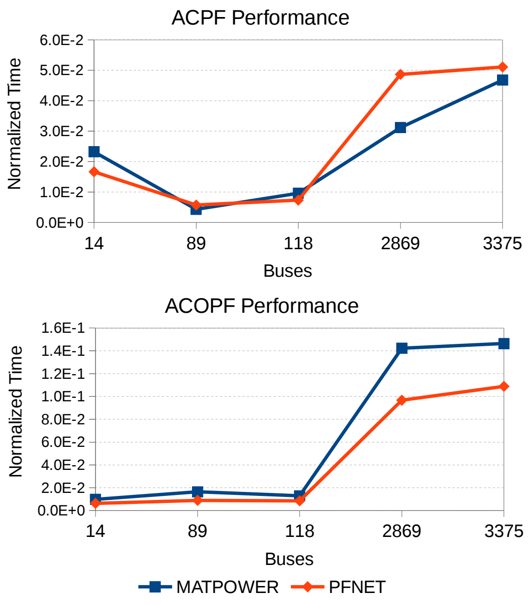

Fig. 2. ACPF and ACOPF performane

TABLE I
ACPF accuracy

| buses | $\lvert\Delta v^*\rvert$ (p.u.) | $\lvert\Delta\theta^*\rvert$ (rad) |
|---|---|---|
| 14 | $(0 \pm 0) \times 10^{-0}$ | $(0 \pm 0) \times 10^{-0}$ |
| 89 | $(0 \pm 0) \times 10^{-0}$ | $(0 \pm 0) \times 10^{-0}$ |
| 118 | $(3 \pm 9) \times 10^{-4}$ | $(2 \pm 3) \times 10^{-4}$ |
| 2869 | $(0 \pm 0) \times 10^{-0}$ | $(0 \pm 0) \times 10^{-0}$ |
| 3375 | $(1 \pm 2) \times 10^{-4}$ | $(3 \pm 3) \times 10^{-5}$ |

TABLE II
ACOPF accuracy

| buses | $\lvert\Delta\phi(x^*)\rvert$ (%) | $\lvert\Delta v^*\rvert$ (p.u.) | $\lvert\Delta\theta^*\rvert$ (rad) |
|---|---|---|---|
| 14 | $7 \times 10^{-4}$ | $(1 \pm 0) \times 10^{-4}$ | $(6 \pm 6) \times 10^{-5}$ |
| 89 | $4 \times 10^{-2}$ | $(2 \pm 2) \times 10^{-2}$ | $(5 \pm 3) \times 10^{-3}$ |
| 118 | $1 \times 10^{-4}$ | $(3 \pm 3) \times 10^{-4}$ | $(7 \pm 7) \times 10^{-5}$ |
| 2869 | $4 \times 10^{-3}$ | $(1 \pm 2) \times 10^{-3}$ | $(2 \pm 3) \times 10^{-3}$ |
| 3375 | $1 \times 10^{-1}$ | $(4 \pm 7) \times 10^{-3}$ | $(6 \pm 7) \times 10^{-3}$ |

## VI. Conclusions

In this work, the PFNET library has been introduced and its key features have been described. This library is open-source and can be used for modeling electric power networks and for constructing related optimization problems. It is efficient, modular, and flexible. In particular, new parsers, functions, and constraints can be added without altering the core framework. The library provides a collection of functions and constraints that can be used independently or together to form optimization problems.

New features that are planned for future versions of the library include unit commitment-related variables, functions and constraints, an interface for Common Information Model (CIM) data, and automatic differentiation. The latter will facilitate the addition of new functions and constraints into the library.

It is the hope of the authors that PFNET becomes a valuable tool for the community, and that students, researchers, and practitioners participate in its development and growth.

## References

[1] NERC, "Real-time tools survey: Analysis and recommendations," Real-Time Tools Best Practices Task Force, North American Electric Reliability Corporation, Tech. Rep., March 2008.

[2] R. Zimmerman, C. Murillo-Sanchez, and R. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, February 2011.

[3] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[4] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[5] R. Byrd, J. Nocedal, and R. Waltz, *KNITRO: An Integrated Package for Nonlinear Optimization*. Boston, MA: Springer US, 2006, pp. 35–59.

[6] P. Gill, W. Murray, and M. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[7] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, March 2006.

[8] D. Shchetinin, T. Tinoco De Rubira, and G. Hug, "Conservative linear line flow constraints for ac optimal power flow," in *IEEE PowerTech Conference*, June 2017, pp. 1–6.

[9] T. Tinoco De Rubira, "Numerical optimization and modeling techniques for power system operations and planning," Ph.D. dissertation, Stanford University, March 2015.

[10] T. Tinoco De Rubira and A. Wigington, "Extending complementarity-based approach for handling voltage band regulation in power flow," in *IEEE Power Systems Computation Conference*, June 2016, pp. 1–6.

[11] M. Baltzinger, "Performance analysis and validation of optimal power flow solvers of new open-source software stack," Semester thesis, ETH Zurich, June 2017.