ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

eeh power systems laboratory

Martin Baltzinger

Performance Analysis and Validation of Optimal Power Flow Solvers of New Open-Source Software Stack

Semester Thesis
PSL1711

EEH – Power Systems Laboratory
Swiss Federal Institute of Technology (ETH) Zurich

Examiner: Prof. Dr. Gabriela Hug
Supervisor: Dr. Tomás Tinoco De Rubira

Zurich, June 21, 2017

Abstract

Solving optimal power flow problems is an important and challenging task in power systems applications. As such, it is an active area of research, and many modeling and computational techniques are periodically being proposed for solving these problems more efficiently and reliably. However, relatively few open-source software platforms exist for the research community to implement, test, and share new methodologies. Currently, the only major open-source package available for solving optimal power flow problems is MATPOWER, which is a MATLAB-based package. Recently, a new open-source software stack has been developed for power systems computational research. This software consists of a group of applications and packages that are modular, extendable, and designed to work together and cover a wide range of the applications spectrum. In particular, it includes low-level tools such as optimization solvers and wrappers, mid-level tools for modeling and constructing optimization problems that arise in power systems, and high-level tools for performing common computational tasks such as solving optimal power flow problems. This software, which is based on C and Python, has thus the potential to be a useful platform for the research community. In this thesis, the tools available in this new open-source software stack for constructing and solving optimal power flow problems are described as well as validated and compared against MATPOWER. In particular, details are provided about the optimization algorithms available, and examples are given that illustrate the capabilities of the new tools.

Contents

List of Acronyms

AC	Alternating Current
DC	Direct Current
KKT	Karush-Kuhn-Tucker
MILP	Mixed-Integer Linear Programming
OPF	Optimal Power Flow
PF	Power Flow
SDP	Semidefinite Programming
SOCP	Second-Order Cone Programming

List of Symbols

$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of non-negative real numbers
$\mathbb{Z}_+$	Set of non-negative integers
$\in$	Set membership
∇	First derivative (gradient) operator
∇^2	Second derivative (Hessian) operator
$\times$	Set product
$\forall$	For all
Σ	Summation
$\lvert \cdot \rvert$	Magnitude of complex number
P_k^g	Generator active power
$P_k^{\min}$	Generator active power lower limit
$P_k^{\max}$	Generator active power upper limit
Q_k^g	Generator reactive power
$Q_k^{\min}$	Generator reactive power lower limit
$Q_k^{\max}$	Generator reactive power upper limit
P_k^l	Load active power
Q_k^l	Load reactive power
v_k	Bus voltage magnitude
$v_k^{\min}$	Bus voltage magnitude lower limit
$v_k^{\max}$	Bus voltage magnitude upper limit
θ_{km}	Phase angle difference

I_{km}	Branch current
$I_{km}^{\max}$	Branch current magnitude limit
$\mathcal{B}$	Set of network branches
n	Number of buses
k	Bus index
m	Bus index
$[n]$	Notation for $\{1, \ldots, n\}$
a_k	Generator cost function coefficient
b_k	Generator cost function coefficient
c_k	Generator cost function coefficient
G_{km}	Element of real part of admittance matrix
B_{km}	Element of imaginary part of admittance matrix
f	Objective function of abstract problem or constraint function of PFNET problem
g	Constraint function of abstract problem
J	Jacobian of g
x^*	Optimal primal point
λ^*	Optimal dual point
μ	Barrier parameter
log	Natural logarithm
Δx_k	Primal step
$\Delta \lambda_k$	Dual step
X	Notation for Diag(x)
e	Vector of ones
L	Lagrangian $f(x) - \lambda^T g(x)$
α_k	Step length
$\alpha_k^{\max}$	Step length upper limit
δ_c	Inertia-controlling parameter
δ_d	Inertia-controlling parameter
ρ_k	Penalty parameter
φ	Objective function of PFNET problem
A	Matrix of linear equality constraints of PFNET problem

b	Right-hand side vector of linear equality constraints of PFNET problem
l	Lower limits of linear inequality constraints of PFNET problem
u	Upper limits of linear inequality constraints of PFNET problem
G	Matrix of of linear inequality constraints of PFNET problem

Chapter 1

Introduction

The Optimal Power Flow (OPF) problem is one of the most basic problems in power system analysis, and as such plays a critical role in many control and operations planning applications. In general, this problem consists of determining control settings and generator dispatches that optimize a specific performance measure, while taking into account various operational and security constraints of the system. Common performance measures include active power losses and total generation cost. The constraints can be various, from thermal limits of transmission lines and transformers, to limits on bus voltage magnitudes and even on carbon emissions. In deregulated electricity markets, OPF problems are solved for obtaining market-clearing prices. However, with accurate network models, OPF problems can be highly nonlinear and complex.

With AC network models, the OPF problem is non-convex and hence achieving global optimality or even feasibility cannot be guaranteed in practice in general. For this reason, several authors have proposed using various types of approximations. Some of these are Semidefinite Programming (SDP), Second-Order Cone Programming (SOCP), and linear power flow model approximations, *e.g.*, DC approximation [1]. Other authors have focused on partitioning the power network and then applying distributed optimization techniques for solving OPF problems [2]. Despite the vast number of ideas and methods proposed in the literature for solving OPF problems with AC models, solving these problems remains a challenge [3]. Improvements in numerical algorithms for solving OPF problems are still necessary.

Of particular importance for OPF research is the existence of open software platforms that facilitate the development and testing of new ideas with realistic models. In the power systems research community, a popular such tool is the package MATPOWER [4]. This package provides several power network data files as well as tools for solving Power Flow (PF) and OPF problems in MATLAB or Octave. From this package, other MATLAB-based tools have been created. These include MATACDC, a program for AC/DC power flow analysis [1], and MATDYN, a tool for dynamic analysis [2]. In addition, other MATLAB-based OPF tools that have been developed include a solver for OPF

[1] `www.esat.kuleuven.be/electa/teaching/matacdc`

[2] `www.esat.kuleuven.be/electa/teaching/matdyn`

SDP-based relaxations [3]. Aside from MATLAB-based packages, a few packages exist for other languages or environments. These include DCOPFJ, a bid/offer-based DC-OPF solver for Java [4], and PYPOWER, a port of MATPOWER to the Python programming language [5].

In this thesis, a new non-MATLAB-based open-source software stack for solving OPF problems is introduced and compared to MATPOWER. This software stack has important features: It consists of modular Python packages that cover a wide spectrum, from low-level algorithms and solver interfaces (OPTALG), to tools for modeling power networks (PFNET), and high-level PF- and OPF-dedicated methods (GRIDOPT). These various packages are designed to interact with each other easily, and to be extendable. For this thesis, these packages have been updated to model and solve standard OPF problems. More specifically, the contributions made in this thesis consist of the following:

- **GRIDOPT**: This package has been extended so that standard AC OPF problems can be created and solved. Additionally, new features have been added, such as the option of using a flat starting point or the possibility to add thermal limit constraints. These features have also been made available in the GRIDOPT command-line utility.
- **OPTALG**: The barrier strategy of the Augmented Lagrangian solver has been improved. In addition, type-casting has been added to transform optimization problems into the standard form used by OPTALG. This allowed solving PFNET problems with OPTALG directly.
- **Testing**: PFNET-compatible test cases and solution files were created from MATPOWER cases, including ones with binding thermal limits. These cases were used to validate and assess the performance of the OPF tools from the new software stack.
- **Deployment**: The build system of the new software stack has been updated to work with the operating system OS X.

This thesis is structured as follows: Chapter 2 introduces a standard OPF problem formulation and discusses its key properties. Chapter 3 provides an overview of the algorithms used by the OPF solvers available in MATPOWER and the new software stack. Chapter 4 describes MATPOWER and the new software stack and highlights some of the key similarities and differences. Chapter 5 compares the performance and solutions of the two software and explores some modeling features of the new software stack. Lastly, Chapter 6 summarizes the work and suggests future improvements.

[3] `ieor.berkeley.edu/~lavaei/Software.html`
[4] `www2.econ.iastate.edu/tesfatsi/DCOPFJHome.htm`
[5] `github.com/rwl/PYPOWER`

Chapter 2

The Optimal Power Flow Problem

In order to be implemented and then solved, the OPF problem must be first formulated as a mathematical optimization problem. In this section, a standard formulation of the OPF problem is described. First, the physical formulation is provided by introducing characteristics of a power network. Then, an abstract high-level version that captures its main properties is provided. The latter is used for describing the solution algorithms in the next chapter.

Electrical networks are subject to various physical and operational constraints. For example, Kirchhoff's law states that energy is conserved at each bus of the system, leading to the so-called active and reactive power balance equations, which are given by

$$P_k^g - P_k^l = v_k \sum_{m\in[n]} v_m(G_{km}\cos\theta_{km} + B_{km}\sin\theta_{km}) \tag{2.1}$$

$$Q_k^g - Q_k^l = v_k \sum_{m\in[n]} v_m(G_{km}\sin\theta_{km} - B_{km}\cos\theta_{km}), \tag{2.2}$$

for each $k \in [n]$. Here, the symbols P_k^g and Q_k^g denote the active and reactive powers, respectively, produced by the generator connected to bus k. Similarly, P_k^l and Q_k^l denote the active and reactive powers, respectively, consumed by the load connected to bus k. The symbols v_k and θ_{km} denote the voltage magnitude and phase angle differences at bus k and between buses k and m, respectively. G_{km} and B_{km} denote the real and imaginary parts, respectively, of the network admittance matrix, $n \in \mathbb{N}$ denotes the number of buses in the network, and $[n] := \{1, \ldots, n\}$ is the set of bus indices. For notation simplicity, we have assumed that there is one generator and one load connected at each bus of the network. Besides these power balance equations, the active and reactive output powers of generator are limited, *i.e.*,

$$P_k^{\min} \le P_k^g \le P_k^{\max}$$
$$Q_k^{\min} \le Q_k^g \le Q_k^{\max},$$

for each $k \in [n]$. In addition to limits on the generator powers, limits on the maximum current through transmission lines and transformers are necessary to prevent damages. These conditions can be expressed as

$$|I_{km}(v_k, v_m, \theta_{km})| \leq I_{km}^{\max},$$

for each $(k, m) \in \mathcal{B}$, where $\mathcal{B} \subset [n] \times [n]$ denotes the set of branches of the network, I_{km} gives the current leaving bus k through branch (k, m) towards bus m, and $I_{km}^{\max}$ denotes the current magnitude limit. Lastly, for stability and protection purposes, bus voltage magnitudes must be restricted. This can be expressed as

$$v_k^{\min} \leq v_k \leq v_k^{\max},$$

for each $k \in [n]$.

The OPF problem aims to optimize system performance with respect to a certain measure. A commonly-used measure is total generation cost, which is given by

$$\sum_{k \in [n]} a_k (P_k^g)^2 + b_k P_k^g + c_k, \tag{2.3}$$

where $(a_k, b_k, c_k) \in \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R}$. Since a_k is non-negative, this function is convex.

The constraints and objective function described here constitute the standard OPF problem. Mathematically, this problem can be expressed as follows:

$$\begin{aligned}
\underset{v,\theta,P^g,Q^g}{\text{minimize}} \quad & \sum_{k \in [n]} a_k (P_k^g)^2 + b_k P_k^g + c_k && && (2.4)\\
\text{subject to} \quad & P_k^g - P_k^l = v_k \sum_{m \in [n]} v_m (G_{km} \cos\theta_{km} + B_{km} \sin\theta_{km}), && \forall k \in [n] \\
& Q_k^g - Q_k^l = v_k \sum_{m \in [n]} v_m (G_{km} \sin\theta_{km} - B_{km} \cos\theta_{km}), && \forall k \in [n] \\
& |I_{km}(v_k, v_m, \theta_{km})| \leq I_{km}^{\max}, && \forall (k,m) \in \mathcal{B} \\
& P_k^{\min} \leq P_k^g \leq P_k^{\max}, && \forall k \in [n] \\
& Q_k^{\min} \leq Q_k^g \leq Q_k^{\max}, && \forall k \in [n] \\
& v_k^{\min} \leq v_k \leq v_k^{\max}, && \forall k \in [n],
\end{aligned}$$

where v, θ, P^g, Q^g represent the vectors of bus voltage magnitudes, bus voltage angles, generator active powers, and generator reactive powers, respectively. Optimization problem (2.4) has a convex objective function, non-convex nonlinear equality constraints, non-convex nonlinear inequality constraints, and simple variable bounds.

It can be shown that, after some transformations, this problem can be expressed in the following equivalent form:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & f(x) && (2.5)\\
\text{subject to} \quad & g(x) = 0 \\
& x \geq 0.
\end{aligned}$$

This formulation will be used in the next chapter to describe algorithms used for solving the OPF problem.

Chapter 3

Solution Algorithms

In previous chapters, the OPF problem has been described and an abstract formulation has been provided. This chapter focuses on describing some of the algorithms that exist for solving problems of this type. In particular, it covers algorithms that are available in open-source software packages for solving OPF problems. The algorithms presented in this chapter are either based on primal or primal-dual approaches, and they all use a barrier method for handling variable bounds.

3.1 Primal-Dual Barrier Algorithms

In the absence of variable bounds in problem (2.5), it can be shown that, under certain conditions, the optimal primal-dual point (x^*, λ^*) is the solution of the following system of equations:

$$\begin{aligned} \nabla f(x) - J(x)^T \lambda &= 0 \\ g(x) &= 0, \end{aligned}$$

where J denotes the Jacobian of g. This system is know as the Karush-Kuhn-Tucker (KKT) system. To handle the bounds on x, primal-dual barrier algorithms extend the objective function with a barrier term, whose weight is controlled by a positive parameter μ. The resulting problem is the following:

$$\underset{x}{\text{minimize}} \quad f(x) - \mu \sum_i \log(x_i) \tag{3.1}$$

$$\text{subject to} \quad g(x) = 0, \tag{3.2}$$

It can be shown that solutions of this subproblem tend solutions of problem (2.5) as μ approaches zero. The strategy consists of approximately solving subproblem (3.1) for a sequence of barrier parameters $\{\mu_k\}_{k \in \mathbb{Z}_+}$ that satisfies $\mu_k \to 0$ as $k \to \infty$. Various approaches exist for approximately solving subproblem (3.1) for each μ_k.

One approach consists of taking a single primal-dual Newton step $(\Delta x_k, \Delta\lambda_k)$ towards solving the KKT system, which for problem (3.1) with μ_k is given by:

$$\begin{aligned} \nabla f(x) - \mu X^{-1} e - J(x)^T \lambda &= 0 \\ g(x) &= 0, \end{aligned}$$

where $X := \mathrm{Diag}(x)$, and e is the vector of ones. More specifically, the step $(\Delta x_k, \Delta\lambda_k)$ is obtained by solving the Newton system

$$\begin{bmatrix} \nabla^2 L(x_k, \lambda_k) + \mu_k X_k^{-2} & -J(x_k)^T \\ -J(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta\lambda_k \end{bmatrix} = - \begin{bmatrix} \nabla L(x_k, \lambda_k) - \mu_k X_k^{-1} e \\ -g(x_k) \end{bmatrix},$$

where (x_k, λ_k) denotes the current primal-dual solution estimate, and $L(x, \lambda) := f(x) - \lambda^T g(x)$. To ensure that the next primal estimate x_{k+1} stays within bounds, the following update is used:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k \Delta x_k \\ \lambda_{k+1} &= \lambda_k + \alpha_k \Delta\lambda_k, \end{aligned}$$

where $\alpha_k := \min\{1, \alpha_k^{\max}\}$, and $\alpha_k^{\max}$ is a suitable positive scalar. This approach is similar to the one used by the solver MIPS, available in MATPOWER [4].

A second approach for approximately solving subproblem (3.1), for a given μ_k, consists of reducing the error in the KKT conditions by a specific amount with respect to the previous subproblem. To achieve that, several modified Newton steps may be computed and taken for a single subproblem. These modified Newton steps are obtained by solving the modified Newton systems

$$\begin{bmatrix} \nabla^2 L(x_k, \lambda_k) + \mu_k X_k^{-2} + \delta_c I & -J(x_k)^T \\ -J(x_k) & -\delta_d I \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta\lambda_k \end{bmatrix} = - \begin{bmatrix} \nabla L(x_k, \lambda_k) - \mu_k X_k^{-1} e \\ -g(x_k) \end{bmatrix},$$

where the scalar parameters δ_c, δ_d are chosen in such a way that a suitable matrix inertia is achieved. With this, it can be ensured that the steps point in directions along which sufficient progress can be made. The next primal-dual point (x_{k+1}, λ_{k+1}) is again obtained using

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k \Delta x_k \\ \lambda_{k+1} &= \lambda_k + \alpha_k \Delta\lambda_k, \end{aligned}$$

but α_k is now obtained through a filtered line search procedure that ensures that the next point is "better" than the previous. This approach is used by the solver IPOPT [5].

3.2 Augmented Lagrangian Barrier Algorithm

In this approach for solving problem (2.5), variable bounds are again handled by moving them to objective function with a barrier. In addition, the nonlinear constraints are also

moved to the objective function by forming an augmented Lagrangian. The strategy of this method hence consists of approximately solving a sequence of unconstrained subproblems. More specifically, each subproblem is given by

$$\underset{x}{\text{minimize}} \quad f(x) - \mu_k \sum_i \log(x_i) - \lambda_k^T g(x) + \frac{\rho_k}{2} \|g(x)\|_2^2, \tag{3.3}$$

where μ_k are positive parameters that approach zero, ρ_k are non-decreasing positive parameters, and λ_k are Lagrange multiplier estimates. Subproblems (3.3) are approximately solved with a modified Newton approach combined with a line search procedure. Dual variable estimates λ_k are updated periodically between each subproblem, together with μ_k and ρ_k. This approach is used by the solver AugL available in a new open-source software stack, described in the next chapter.

Chapter 4

Packages and Tools

In previous chapters, the standard OPF problem was presented along with some optimization algorithms for solving it. More specifically, the algorithms described were algorithms used by solvers available in MATPOWER and the new software stack. In this chapter, these software are introduced in the context of constructing and solving OPF problems.

4.1 Overview

For research, various OPF software exist, from general nonlinear solvers to dedicated OPF packages. Figure 4.1 shows in green some of these software.

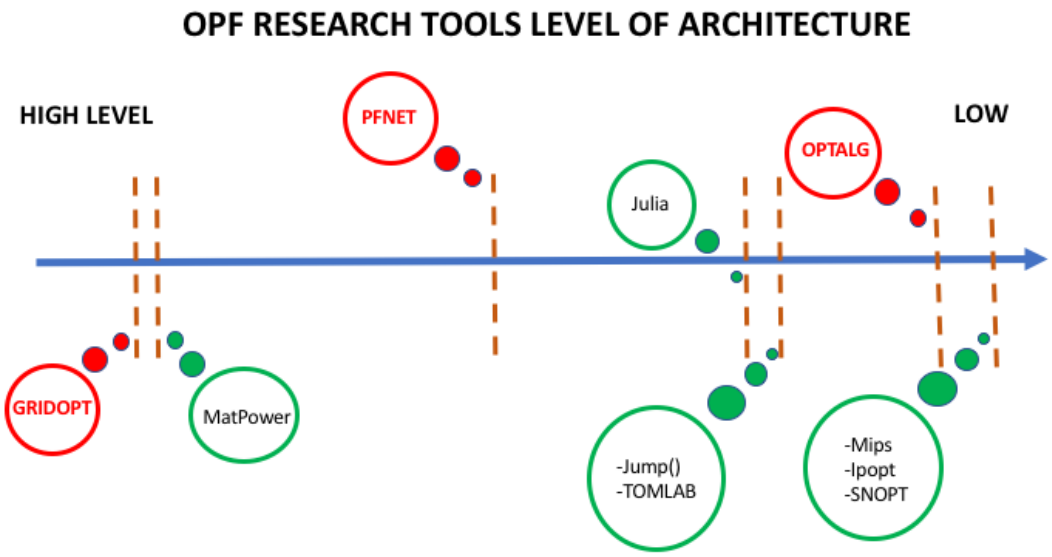

Figure 4.1: OPF research tools

This figure highlights the flexibility of each software. IPOPT is very flexible and thus can be applied to various types of problems. However, low-level functions and data must be provided as routines and vectors. This can be tedious and time-consuming for non-expert users. On a higher level, JuMP [1], which is a modeling framework for mathematical optimization, provides tools for constructing and solving problems that is much more natural and closer to mathematics. MATPOWER, on an even higher level of encapsulation, handles all of the problem construction, transformation, and solver invocation. However, it provides only relatively limited modeling options compared to using JuMP or a solver directly. Recently, a new software stack for power systems research, in particular for OPF, has been developed. It includes the packages PFNET [2], GRIDOPT [3], and OPTALG [4], as shown in red in Figure 4.1. These packages cover the whole spectrum, providing both low-level and high-level tools. Moreover, with PFNET, they fill an important gap between high-level OPF-dedicated software, such as MATPOWER, and lower-level generic modeling libraries, such as JuMP.

This chapter focuses on comparing MATPOWER with the packages available in the new software stack. First, the MATPOWER environment is described. Then, each package of the new software stack is presented. For each, an example related to constructing and solving an AC OPF problem is shown.

4.2 MATPOWER Version 6.0

MATPOWER is a powerful MATLAB package for solving PF and OPF problems. It has been developed primarily for researchers to provide them with a powerful tool, while having easily-modifiable code. Details can be found in the user's manual [5] and on the website [6].

After installing MATPOWER, functions and test cases are available within the MATLAB environment that can be used for solving PF and OPF problems. In particular, for solving OPF problems, MATPOWER uses by default the solver MIPS, which is based on a primal-dual barrier algorithm similar to the one describe in Section 3.1. Additional solvers can be used, including IPOPT. Constructing and solving AC OPF problems is done by first loading a case and then invoking one of the installed functions, namely `runopf`. Loading a case is done using the function `loadcase`, which creates a data structure, say `mpc`. The following code snippet shows this:

```
mpc = loadcase("case9")
```

This structure contains fields that define the electrical network, such as generators,

[1] jump.readthedocs.io
[2] pfnet-python.readthedocs.io
[3] gridopt.readthedocs.io
[4] optalg.readthedocs.io
[5] www.pserc.cornell.edu/matpower/MATPOWER-manual.pdf
[6] www.pserc.cornell.edu/matpower

branches, etc. Once a data structure has been created, it can be passed to the function `runopf` to solve the corresponding OPF problem. Additionally, options can be passed to `runopf` to consider alternative models, such as a DC model, or to use different initial points. The following code snippet shows how to configure the options to use a warm starting point and branch thermal limits based on current magnitudes, and solve the OPF problem:

```
option_OPF= struct("init_from_mpc",1,"flow_lim","I")
overrides = struct("opf",option_OPF);
mpopt = mpoption(overrides);
result = runopf(mpc,mpopt)
```

Information about the solution, such as the final objective value, can be obtained from the `result` data structure, *e.g.*,

```
result.f
```

The use of only one function for solving everything makes it very simple for the user. However, configuring some options is not straightforward. For example, to ignore thermal limits, the right element of each branch in the `mpc` structure must be set to zero, as shown below:

```
for p=1:length(mpc.branch(:,1))
   mpc.branch(p,6)=0;
end
```

As another example, to use soft voltage limits as opposed to hard ones, user-defined functions need to be created. Not many alternative constraints and functions are available with MATPOWER. For custom OPF formulations, one may have to create new ones. For scheduling-type problems, a new add-on to MATPOWER called MOST has been recently developed. The interested reader is referred to the user's manual for more details [7].

4.3 The New Software Stack

This new software stack is composed mainly of three software packages that interact with one another. They have specific roles, which are described in this section.

4.3.1 PFNET Version 1.3.0

PFNET is a modeling library for electric power networks. Written in C, it provides data parsers and routines for analyzing electrical networks and formulating related opti-

[7] `www.pserc.cornell.edu/matpower/MOST-manual.pdf`

mization problems, such as OPF. Moreover, it is designed so that users can plug in new parsers, constraints and functions. PFNET also provides a Python wrapper that makes all of its functionality accessible in a high-level and object-oriented manner. Using a MATLAB function, it is possible to transform an `mpc` MATPOWER case structure into a csv file (of extension ".mat") that is usable by PFNET. This ".mat" case can be parsed in PFNET to create a network, as follows:

```
from pfnet import *
net = ParserMAT().parse("case9.mat")
```

PFNET provides several routines for analyzing a network. For example, one can obtain the maximum bus active power mismatch and the total generation cost of the network by accessing attributes of the network object:

```
print(net.bus_P_mis)
print(net.gen_P_cost)
```

Rather than relying on a "black-box" function for creating variables and forming optimization problems, PFNET provides the user with the possibility to define a problem from scratch. By setting flags, one can manually choose which quantities in a network are considered to be variables and bounded. This provides flexibility to users. The following code snippet shows how to do this for a standard AC OPF problem:

```
# Voltage magnitude
net.set_flags("bus",["variable","bounded"],"any","voltage magnitude")

# Voltage angles
net.set_flags("bus","variable","not slack","voltage angle")

# Generator active and reactive powers
net.set_flags("generator",["variable","bounded"],"any",
              ["active power","reactive power"])
```

The user can then create a customized OPF problem by choosing the constraints and functions to be used. The following example shows how to do this:

```
problem = Problem(net)
problem.add_constraint(Constraint("AC power balance",net))
problem.add_constraint(Constraint("variable bounds",net))
problem.add_function(Function("generation cost",1.,net))
problem.analyze()
```

The resulting problem is of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) \\ \text{subject to} \quad & Ax = b \\ & f(x) = 0 \\ & l \leq Gx \leq u. \end{aligned} \tag{4.1}$$

Users can easily access the different elements of the problem, such as the matrix A, the vector b, or even the Hessian matrix of each individual nonlinear constraint. This is also possible for the individual constraints and functions:

```
constr = problem.find_constraint("AC power balance")
print(constr.f)
print(problem.A,problem.b)
```

The problem can then be passed to a solver. In particular, it can be solved directly by the solvers available in the package OPTALG from the new software stack.

4.3.2 OPTALG Version 1.1.3

OPTALG is a Python package for solving optimization problems. Primarily designed for being used with PFNET or other modeling libraries, it can solve any optimization problems that can be expressed in a specific standard form. The algorithms available in OPTALG consist of a Newton-Raphson algorithm for solving systems of equations (NR), an interior-point algorithm for solving convex quadratic problems (IQP), and an augmented Lagrangian algorithm for solving nonlinear optimization problems with convex objective functions (AugL). Additionally, OPTALG provides interfaces for the linear programming solver Clp [8], the mixed-integer programming solver Cbc [9], the interior-point solver IPOPT [5], and for the linear solvers SuperLU [10] and MUMPS [11].

To solve our OPF problem constructed in the previous section using the AugL solver and print the optimal objective value, the following code can be used:

```
from optalg.opt_solver import OptSolverAugL

# Choosing the Solver
augl = OptSolverAugL()

# Solving
augl.solve(problem)
print(problem.phi)
```

[8] projects.coin-or.org/Clp
[9] projects.coin-or.org/Cbc
[10] crd-legacy.lbl.gov/~xiaoye/SuperLU
[11] mumps.enseeiht.fr

4.3.3 GRIDOPT Version 1.3.2

By using PFNET and OPTALG, the user has a great degree of freedom. However, in power system applications, it is often the case that users want to construct and solve standard OPF or PF problems. For convenience, the Python package GRIDOPT from the new software stack encapsulates these common tasks. The user just needs to choose a PF or OPF method, and GRIDOPT takes care of the rest. Coming back to our AC OPF example, the code in GRIDOPT would be the following:

```
from gridopt.power_flow import new_method

# Choosing the Method
method = new_method("AugLOPF")

# Solving
method.solve(net)
results = method.get_results()
method.update_network(net)
print(net.gen_P_cost)
```

In addition, GRIDOPT also provides a command-line utility. For our example, this can be done as follows:

```
$gridopt case9.mat AugLOPF --params feastol=1e-3 thermal_lim=True
```

The parameter key-value pairs shown above can be used to configure the model as well as the underlying optimization solver.

4.4 Summary and Comparison

With regards to network modeling, the MATLAB environment of MATPOWER provides a convenient way to visualize and work with the network data structures. However, unlike PFNET, MATPOWER does not contain routines for analyzing the network. PFNET represents networks and their components as objects and provides various options to the user to get information about them. GRIDOPT, as MATPOWER, provides a convenient way to perform common tasks such as solving standard PF and OPF problems.

With regards to installation, the new software stack requires more expertise since compilation and linking must be done by the user, and instructions only for UNIX-type systems are currently available. To make it more accessible to regular users, a Docker container has been created, called PSCHUB [12]. This container has all the packages from the software stack installed, and runs a Jupyter Notebook [13] that allows users to use

[12] github.com/ttinoco/PSCHUB
[13] jupyter.org

the tools through a web browser. The only installation task required by the user is the installation of Docker [14].

[14] www.docker.com

Chapter 5

Validation and Performance

This chapter focuses on comparing the solution of AC OPF problems using the new software stack and MATPOWER, and on assessing the benefits of some modeling options available in PFNET. First, the cases used for the comparison are described. Then, the results obtained from comparing the software are presented along with a discussion. Lastly, the evaluation of the additional modeling options available in PFNET is provided.

5.1 Test Cases

The cases used for validating and evaluating the OPF tools of the new software were obtained or derived from MATPOWER cases [6] [7]. The derived cases were created by modifying some of the original cases to have binding thermal limits and/or quadratic generator costs. Tables 5.1 and 5.2 provide details about the two groups of test cases used, namely original and modified.

Table 5.1: Original Cases

name	buses	branches	thermal limits	binding
case9	9	9	9	no
case14	14	20	0	no
case39	39	46	46	no
case118	118	186	0	no
case300	300	411	0	no
case2869pegase	2869	4582	2743	no
case9241pegase	9241	16049	6295	no

Table 5.2: Modified Cases

name	buses	branches	thermal limits	binding
case9_mod	9	9	9	yes
case14_mod	14	20	20	yes
case39_mod	39	46	46	yes
case118_mod	118	186	186	yes
case300_mod	300	411	411	yes
case2869pegase_mod	2869	4582	2743	yes
case9241pegase_mod	9241	16049	6295	yes

5.2 MATPOWER Comparison

The experiments carried consisted of constructing standard AC OPF problems (see Chapter 2) from the modified test cases shown in Table 5.2 using PFNET. Then, the problems were solved using the solvers available in OPTALG, namely IPOPT and AugL. The optimal generation cost, solver execution time, and iterations were compared against those obtained with the solver MIPS of MATPOWER. For each test case, this was done using both warm and flat starting points, and with thermal limits being enabled and disabled. The results obtained are shown in Figures 5.1-5.4.

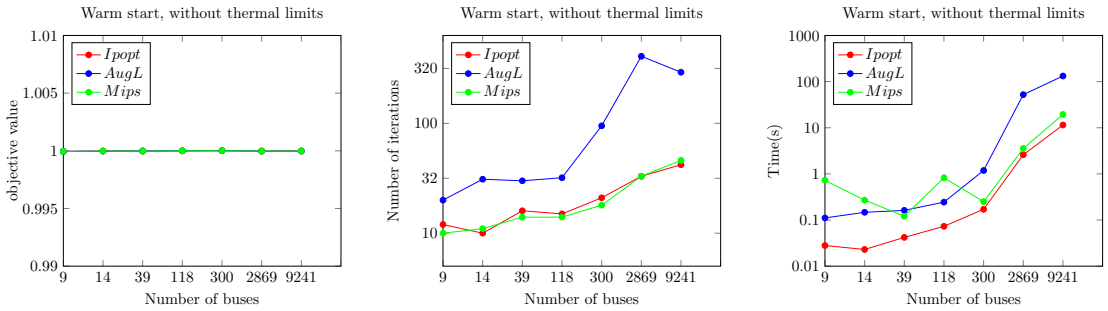

Figure 5.1: Warm start, without thermal limits

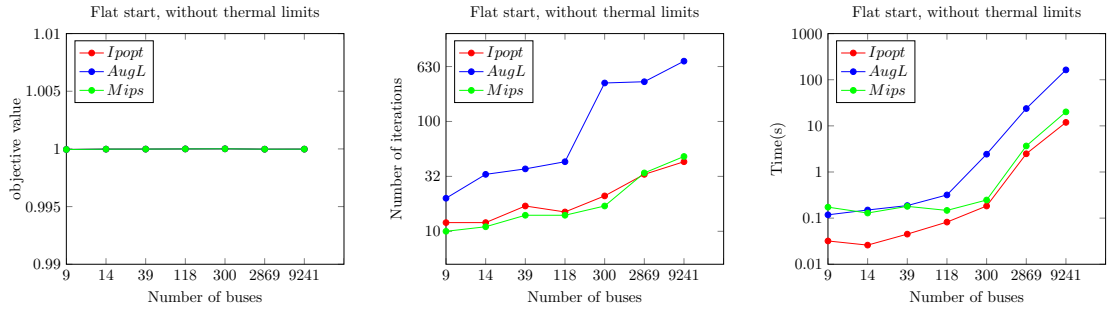

Figure 5.2: Flat start, without thermal limits

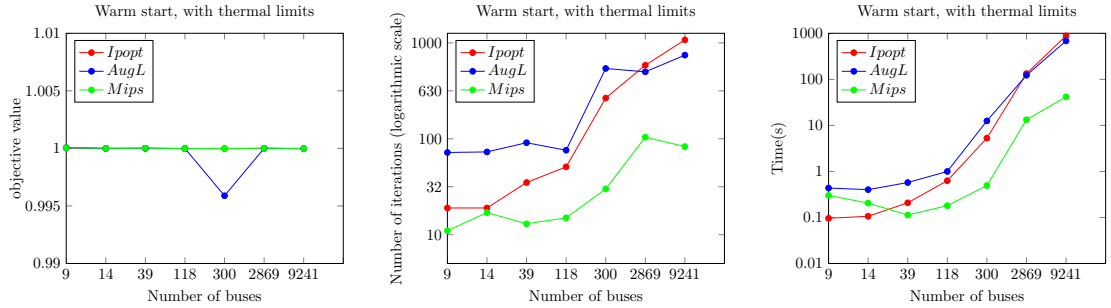

Figure 5.3: Warm start, with thermal limits

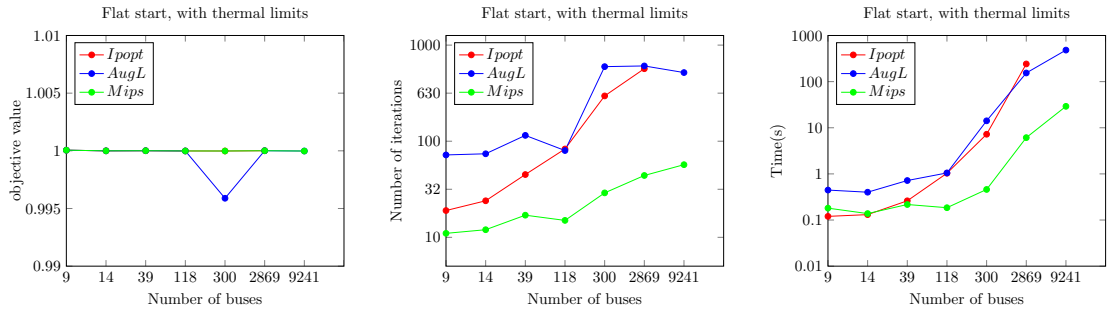

Figure 5.4: Flat start, with thermal limits

It can be seen from the figures that the optimal objective values obtained with the algorithms available in the new software stack are in agreement with those obtained with MATPOWER. However, depending on the options used, the performance of the solvers IPOPT and AugL have some noticeable differences with that of MIPS. In terms of time and iterations, both IPOPT and MIPS are much more efficient than AugL. This is in a way expected since AugL is a primal method and it uses an aggregate measure of constraint violations. The performance differences between IPOPT and MIPS are small except when considering thermal limits, as seen on Figure 5.5. When considering

thermal limits, IPOPT requires more iterations and time, and it even fails to solve a case. We suspect here that this outcome is not due to an issue of IPOPT but to the way the AC OPF problem is constructed by PFNET. In particular, PFNET introduces slack variables in order to express thermal limits in a form consistent with (4.1), and this could be responsible for the discrepancies in performance observed between IPOPT and MIPS.

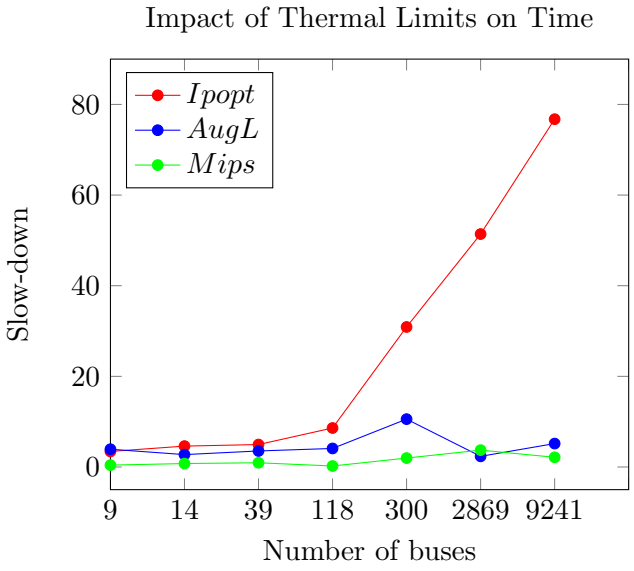

Figure 5.5: Impact of thermal limits on solver time

5.3 Linearized Thermal Limits

An alternative to the nonlinear thermal limits is available in the new software stack through an external library based on the techniques described in [8]. This library can be linked to PFNET, providing the software with an additional constraint option. This option provides accurate linear conservative approximations of the thermal limits. This feature has been tested on the original cases shown on Table 5.1 to investigate its benefit. Figure 5.6 shows the objective values, decrease in iterations, and speed-up factors obtained when solving the AC OPF problem with linearized thermal limits with respect to when solving with nonlinear thermal limits.

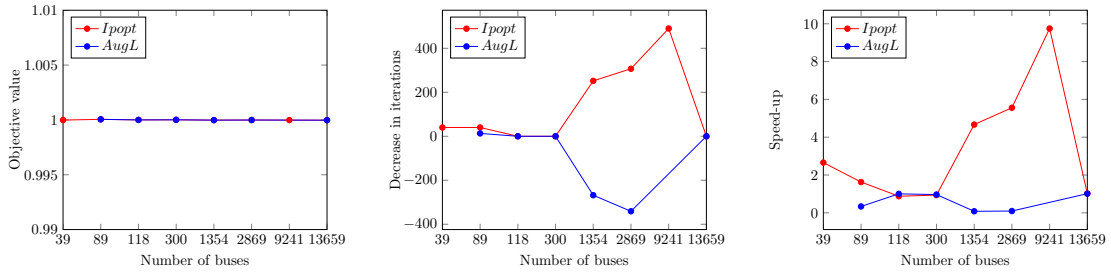

Figure 5.6: Linearized Thermal limits

As can be seen from the figure, IPOPT needs in general less number of iterations and time to solve the problem with linearized thermal limits compared to with nonlinear thermal limits. For the solver AugL, on the other hand, it is the opposite. This is because a large number of linear constraints is needed to accurately approximate the nonlinear thermal limits. For several cases, IPOPT and AugL show no difference in performance with linear and nonlinear thermal limits. The reason for this is that these cases had no thermal limits, as shown in Table 5.1.

Chapter 6

Conclusion

In this thesis, the OPF tools from a new software stack for power system research have been explored. First, the formulation of the OPF problem was introduced. Then, solutions algorithms were described, including those used by the solvers available in the new software stack, namely the primal-dual interior point solver IPOPT and the augmented Lagrangian solver AugL. The OPF tools from the new software stack were then presented and contrasted with MATPOWER. In particular, their modularity and flexibility was highlighted. Subsequently, experimental results evaluating and comparing the performance of the OPF algorithms of the new software stack and MATPOWER were shown and discussed. In particular, it was verified that all of the algorithms of the new software stack achieved the same optimal objective value as with MATPOWER under various conditions. In terms of performance, it was found that the AugL solver was in general slower that IPOPT and MATPOWER's MIPS. In addition, IPOPT was found to have comparable performance with MATPOWER's MIPS except when considering branch thermal limits. IPOPT was slower when considering branch thermal limits and even failed to solve a case. It was hypothesized that this is due to the problem transformations that are done by the tools from the software stack, namely those needed to express the problem in the standard forms of PFNET first, then of OPTALG, and finally of IPOPT.

This thesis highlighted the efficiency of the primal-dual method used by MIPS for solving OPF problems. Thus, further work could focus on including a similar solver to the new software stack inside the package OPTALG. In addition, wrappers for other nonlinear optimization solvers such as SNOPT and KNITRO need to be added for further comparisons. Lastly, variables, constraints and functions need to be included in PFNET in order to formulate unit commitment problems. Once this is done, OPTALG should be extended to target Mixed-Integer Linear Programming (MILP) solvers such as Gurobi and CPLEX.

Appendix A

Result Tables

The following tables show the performance of each solver according to four criteria. The cases where "nan" appears are cases for which the solver failed. The symbol φ represents the final objective value. The subscripts IP, AL, and MP are used to associate quantities with the solvers IPOPT, AugL, and MIPS, respectively. The symbol It stands for iterations, T for time in seconds, and Δ for the maximum bus power mismatch in per unit system base power. Tables A.1-A.4 show the results obtained using the modified cases shown in Table 5.2. Tables A.5-A.6 show the results obtained using the original cases shown in Table 5.1 and others.

Number of buses	9	14	39	118	300	2869	9241
φ_{IP}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
φ_{AL}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
φ_{MP}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
It_{IP}	12	10	16	15	21	33	42
It_{AL}	20	31	30	32	95	408	292
It_{MP}	10	11	14	14	18	33	46
T_{IP}	0.028	0.023	0.042	0.073	0.171	2.594	11.481
T_{AL}	0.111	0.147	0.162	0.244	1.185	52.182	132.201
T_{MP}	0.726	0.270	0.121	0.824	0.250	3.563	19.434
Δ_{IP}	3.07e-08	8.20e-08	6.15e-07	2.34e-07	2.22e-07	1.35e-05	1.34e-05
Δ_{AL}	2.09e-07	2.36e-09	5.20e-06	2.33e-06	1.91e-07	3.34e-06	2.25e-05
Δ_{MP}	9.42e-07	7.80e-11	3.14e-07	8.95e-08	2.48e-07	6.11e-08	1.32e-08

Table A.1: Warm start, without thermal limits

Number of buses	9	14	39	118	300	2869	9241
φ_{IP}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
φ_{AL}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
φ_{MP}	5.30e+03	8.08e+03	4.19e+04	1.30e+05	7.20e+05	1.52e+05	3.91e+05
It_{IP}	12	12	17	15	21	33	43
It_{AL}	20	33	37	43	224	230	354
It_{MP}	10	11	14	14	17	34	48
T_{IP}	0.032	0.026	0.045	0.082	0.183	2.482	11.891
T_{AL}	0.118	0.150	0.187	0.318	2.432	23.743	163.413
T_{MP}	0.173	0.129	0.180	0.147	0.247	3.666	20.103
Δ_{IP}	3.07e-08	8.20e-08	6.15e-07	2.34e-07	2.22e-07	1.35e-05	1.34e-05
Δ_{AL}	2.09e-07	8.32e-08	5.16e-05	3.00e-06	3.49e-05	3.71e-05	2.86e-05
Δ_{MP}	9.42e-07	3.73e-10	6.78e-07	1.18e-07	4.25e-07	6.32e-08	1.18e-08

Table A.2: Flat start, without thermal limits

Number of buses	9	14	39	118	300	2869	9241
φ_{IP}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.32e+05	1.53e+05	3.91e+05
φ_{AL}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.29e+05	1.53e+05	3.91e+05
φ_{MP}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.32e+05	1.53e+05	3.91e+05
It_{IP}	19	19	35	51	266	585	1074
It_{AL}	72	73	91	76	541	499	745
It_{MP}	11	17	13	15	30	104	83
T_{IP}	0.096	0.106	0.208	0.628	5.283	133.327	881.081
T_{AL}	0.435	0.402	0.573	0.998	12.513	122.599	683.295
T_{MP}	0.301	0.205	0.113	0.180	0.493	13.256	41.606
Δ_{IP}	2.61e-08	7.66e-08	6.62e-07	1.78e-07	1.18e-06	1.35e-05	1.35e-05
Δ_{AL}	4.80e-07	3.21e-07	5.40e-07	3.95e-06	3.96e-05	7.11e-06	1.67e-06
Δ_{MP}	3.63e-11	1.11e-07	4.54e-09	1.82e-07	1.40e-07	7.96e-09	1.97e-09

Table A.3: Warm start, with thermal limits

Number of buses	9	14	39	118	300	2869	9241
φ_{IP}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.32e+05	1.53e+05	nan
φ_{AL}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.29e+05	1.53e+05	3.91e+05
φ_{MP}	6.55e+03	8.19e+03	4.79e+04	1.31e+05	7.32e+05	1.53e+05	3.91e+05
It_{IP}	19	24	45	83	296	568	nan
It_{AL}	72	74	115	80	597	607	518
It_{MP}	11	12	17	15	29	44	57
T_{IP}	0.120	0.131	0.261	1.033	7.234	241.707	nan
T_{AL}	0.446	0.401	0.718	1.049	14.149	154.008	482.426
T_{MP}	0.181	0.138	0.217	0.185	0.460	6.110	29.132
Δ_{IP}	2.61e-08	7.66e-08	6.62e-07	1.78e-07	1.18e-06	1.35e-05	nan
Δ_{AL}	4.80e-07	3.71e-08	3.10e-08	3.20e-06	2.41e-06	7.16e-06	1.76e-05
Δ_{MP}	3.63e-11	3.10e-08	2.71e-07	2.62e-07	1.09e-07	9.10e-08	4.00e-09

Table A.4: Flat start, with thermal limits

Number of buses	9	14	39	89	118	300	1354	2869	3375	9241	13659
φ_{IP}	5.30e+03	8.08e+03	4.19e+04	5.82e+03	1.30e+05	7.20e+05	7.41e+04	1.34e+05	nan	3.16e+05	3.86e+05
φ_{AL}	5.30e+03	8.08e+03	4.19e+04	5.82e+03	1.30e+05	7.20e+05	7.41e+04	1.34e+05	nan	3.16e+05	3.86e+05
It_{IP}	18	10	61	65	15	21	292	351	nan	544	155
It_{AL}	55	31	71	299	32	95	361	427	nan	591	300
T_{IP}	0.095	0.027	0.364	0.456	0.058	0.135	17.816	48.090	nan	323.993	68.556
T_{AL}	0.398	0.141	0.484	2.994	0.249	1.026	37.724	106.312	nan	554.542	204.665
Δ_{IP}	2.06e-08	8.20e-08	5.81e-07	2.11e-07	2.34e-07	2.22e-07	1.49e-05	1.36e-05	nan	1.36e-05	2.07e-05
Δ_{AL}	3.55e-07	2.36e-09	8.56e-05	7.99e-07	2.33e-06	1.91e-07	2.55e-06	2.16e-05	nan	2.56e-06	5.90e-05

Table A.5: Warm start, with nonlinear thermal limits (original cases)

Number of buses	9	14	39	89	118	300	1354	2869	3375	9241	13659
φ_{IP}	5.30e+03	8.08e+03	4.19e+04	5.82e+03	1.30e+05	7.20e+05	7.41e+04	1.34e+05	7.40e+06	3.16e+05	3.86e+05
φ_{AL}	nan	8.08e+03	nan	5.82e+03	1.30e+05	7.20e+05	7.41e+04	1.34e+05	7.40e+06	nan	3.86e+05
It_{IP}	15	10	21	25	15	21	40	44	53	54	155
It_{AL}	nan	31	nan	286	32	95	629	768	907	nan	300
T_{IP}	0.087	0.021	0.137	0.280	0.066	0.144	3.816	8.650	13.083	33.234	67.157
T_{AL}	nan	0.158	nan	8.893	0.248	1.062	460.534	1119.818	1795.201	nan	202.003
Δ_{IP}	2.22e-08	8.20e-08	5.81e-07	2.11e-07	2.34e-07	2.22e-07	1.49e-05	1.36e-05	2.96e-05	1.36e-05	2.07e-05
Δ_{AL}	nan	2.36e-09	nan	9.44e-05	2.33e-06	1.91e-07	5.17e-07	1.10e-05	1.74e-06	nan	5.90e-05

Table A.6: Warm start, with linearized thermal limits (original cases)

Bibliography

[1] D. Molzahn, F. Dörfler, H. Sandberg, S. Low, S. Chakrabarti, R. Baldick, and J. Lavaei. A Survey of Distributed Optimization and Control Algorithms for Electric Power Systems. *IEEE Transactions on Smart Grid*, pages 1–21, February 2017.

[2] J. Guo, G. Hug, and O. Tonguz. Intelligent Partitioning in Distributed Optimization of Electric Power Systems. *IEEE Transactions on Smart Grid*, 7(3):1249–1258, May 2016.

[3] M. Huneault and F. Galiana. A survey of the optimal power flow literature. *IEEE Transactions on Power Systems*, 6(2), 1991.

[4] 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*, 26(1):12–19, February 2011.

[5] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering.* PhD thesis, Carnegie Mellon University, January 2002.

[6] A. Birchfield, T. Xu, K. Gegner, K. Shetye, and T. Overbye. Grid Structural Characteristics as Validation Criteria for Synthetic Networks. *IEEE Transactions on Power Systems*, 2017.

[7] C. Josz, S. Fliscounakis, J. Maeght, and P. Panciatici. AC Power Flow Data in Matpower and QCQP Format: iTesla, RTE Snapshots, and PEGASE. `arxiv.org/abs/1603.01533`, 2016.

[8] D. Shchetinin, T. Tinoco De Rubira, and G. Hug. Conservative linear line flow constraints for AC optimal power flow. In *IEEE PES Powertech Conference*, June 2017.