PowerModels.jl
a Brief Introduction

Carleton Coffrin, et. al.
A Bit About Me

• Trained as Computer Scientist
  • BS - University of Connecticut
  • PhD - Brown University
• Know about CS Stuff
  • Software Engineering
  • Programming Language Design
  • Computational Research Focus

UNCLASSIFIED
A Bit About Me

- Discrete Optimization Research
- Generalist
  - Local Search / Heuristics
  - Constraint Programming
  - MIP
  - NLP & MINLP (more recently)

Discrete Optimization
A Bit About LANL

• Advanced Network Science Initiative (ANSI)
• 10+ Diverse Staff
  • Optimization, ML, Applied Math, Statistical Physics
• Applications in complex networks
  • e.g. Electric Power, Natural Gas, Water
• Developing novel algorithmic methods
A Bit About LANL

ANSI LOVES JuliaOpt
Outline

- Motivation
  - Challenges of R&D in Power Network Optimization
- A Brief Introduction to PowerModels.jl
- Plans for the Near-Future
Motivation
Power Network Optimization is Complicated

\[ \theta_r = 0 \]

\[ p_i^g - p_i^d = \sum_{(i,j) \in E \cup E^R} p_{ij} \quad \forall i \in N \]

\[ q_i^g - q_i^d = \sum_{(i,j) \in E \cup E^R} q_{ij} \quad \forall i \in N \]

\[ p_{ij} = g_{ij} v_i^2 - g_{ij} v_i v_j \cos(\theta_{ij}^\Delta) - b_{ij} v_i v_j \sin(\theta_{ij}^\Delta) \quad (i, j) \in E \cup E^R \]

\[ q_{ij} = -b_{ij} v_i^2 + b_{ij} v_i v_j \cos(\theta_{ij}^\Delta) - g_{ij} v_i v_j \sin(\theta_{ij}^\Delta) \quad (i, j) \in E \cup E^R \]

\[ \theta_{ij}^\Delta = \theta_i - \theta_j \quad \forall (i, j) \in E \]

\[ p_{ij}^2 + q_{ij}^2 \leq (s_{ij}^u)^2 \quad \forall (i, j) \in E \cup E^R \]
AC-Feasibility on Tree Networks is NP-Hard

Karsten Lehmann, Alban Grastien, and Pascal Van Hentenryck

Abstract—Recent years have witnessed significant interest in convex relaxations of the power flows, with several papers showing that the second-order cone relaxation is tight for tree networks under various conditions on loads or voltages. This paper shows that ac-feasibility, i.e., to find whether some generator dispatch can satisfy a given demand, is NP-hard for tree networks.

Index Terms—Computational complexity, optimal power flow (OPF).

I. INTRODUCTION

MANY interesting applications in power systems, including optimal power flows, optimize an objective function over the steady-state power flow equations, which are nonlinear and nonconvex. These applications typically include an ac-feasibility (AC-FEAS) subproblem: find whether some generator dispatch can satisfy a given demand.

Although the set of ac-feasible solutions is in general a nonconvex set, this does not imply that the ac-feasibility problem is NP-hard, as nonconvexity does not imply NP-hardness. For example, the family of optimization problems min \( y \) such that
\[ 0 \leq y \leq \prod_{i=1}^{n} x_{i} \]
where \( n \in \mathbb{N} \) has a nonconvex constraint and a nonconvex solution set but the optimal solution is always \( y = 0 \) and can be trivially computed.

The first NP-hardness proof for ac-feasibility was given for a cyclic network structure in [1]. It relies on a variant of the dc model [2] but uses a sine function around the phase angle difference. From an ac perspective, this means that conductances and susceptances, and bounds on the phase angles.

This NP-hardness result provides some theoretical and practical consequences. First, it shows that finding an ac-feasible dispatch is a hard problem. Second, it implies that there is no polynomial-time algorithm that can guarantee feasibility for all instances. Finally, it suggests that approximate methods may be necessary to solve the ac-feasibility problem in practice.
Power Network Optimization is Complicated

DC Power Flow Approximation

\[ \theta_r = 0 \]

\[ p_i^g - p_i^d = \sum_{(i,j) \in E \cup E^R} p_{ij} \quad \forall i \in N \]

\[ p_{ij} = -b_{ij}(\theta_{ij}^\Delta) \quad (i, j) \in E \cup E^R \]

\[ \theta_{ij}^\Delta = \theta_i - \theta_j \quad \forall (i, j) \in E \]

\[ |p_{ij}| \leq s_{ij}^u \quad \forall (i, j) \in E \cup E^R \]
Power Network Optimization is Complicated

**SOC Power Flow Relaxation**

\[
p_i^g - p_i^d = \sum_{(i,j) \in E \cup E^R} p_{ij} \quad \forall i \in N
\]

\[
q_i^g - q_i^d = \sum_{(i,j) \in E \cup E^R} q_{ij} \quad \forall i \in N
\]

\[
p_{ij} = g_{ij} w_i - g_{ij} w_{ij}^R - b_{ij} w_{ij}^I \quad (i, j) \in E \cup E^R
\]

\[
q_{ij} = -b_{ij} w_i + b_{ij} w_{ij}^R - g_{ij} w_{ij}^I \quad (i, j) \in E \cup E^R
\]

\[
p_{ij}^2 + q_{ij}^2 \leq (s_{ij}^u)^2 \quad \forall (i, j) \in E \cup E^R
\]

\[
(w_{ij}^R)^2 + (w_{ij}^I)^2 \leq w_i w_j \quad (i, j) \in E
\]

\[
\theta_{ij}^\Delta w_{ij}^R \leq w_{ij} \leq w_{ij}^R \theta_{ij}^\Delta u \quad (i, j) \in E
\]
R&D Challenges

• Two Core Issues
  • Power Flow Formulations
  • Test Cases for Benchmarking
The Formulation Problem

• It is possible to publish a new approximation or relaxation, without comparing to many previous works
• There has been an explosion of proposed power flow alternatives (often hard to find)
• No clear top performers, in terms of citations at least…
Formulation Taxonomy
(as of 2014)

Just Relaxations

UNCLASSIFIED
The Instance Problem

• It is possible to publish a new method, by only testing on a few (5-10)
• typically these are very-easy test cases
  • e.g. convex objective function with no binding constraints
• Industry more-or-less ignores academic results
  • One reason is that the test cases are too easy
My Solution?
A novel scientific methodology

Brute-Force R&D
Run All Formulations on All Instances

“No clever ideas required!”
AMPL Implementation

- AC_b_only.mod
- AC_basic.mod
- AC_cb.mod
- AC_cb2.mod
- AC_cp.mod
- AC_current_inject.mod
- AC_current.mod
- AC_distflow_cvx.mod
- AC_first_order.mod
- AC_global_rect.mod
- AC_global_w_rect.mod
- AC_global_w.mod
- AC_global.mod
- AC_line-flex.mod
- AC_line_fp.mod
- AC_line_fp2.mod
- AC_LI_theta.mod
- AC_loss.mod
- AC_nf_lb_lin.mod
- AC_nf_lin.mod
- AC_nf_LI_cvx.mod
- AC_nf_LI.mod
- AC_pf_opf.mod
- AC_pf_soft.mod
- AC_pf.mod
- AC_polar.mod
- AC_poly_cvx.mod
- AC_poly_LI_cvx.mod
- AC_rect.cvx.mod
- AC_rect_gammatg_lin.mod
- AC_rect_gammatg.mod
- AC_rect_LI_cvx.mod
- AC_rect_nf_cvx.mod
- AC_rect_polar.mod
- AC_rect.mod
- DC_cp.mod
- DC_distflow_cvx.mod
- DC_LI_cvx.mod
- DC_LI.mod
- DC_nf_LI_cvx.mod
- DC_nf.mod
- DC.mod
- QC_bus_flex.mod
- QC_cs_cvx.mod
- QC_cut_cvx_fp.mod
- QC_cut_flex_nlp.mod
- QC_cvx_fp_qp.mod
- QC_cvx_fp.mod
- QC_cvx_init.mod
- QC_cvx_sym.mod
- QC_cvx.mod
- QC_dir_cvx.mod
- QC_flex_cvx_pre.mod
- QC_flex_cvx.mod
- QC_flex_nlp_pre.mod
- QC_flex_nlp.mod
- QC_line_flex_nlp.mod
- QC_line:flex.mod
- QC_line_fp2_nlp.mod
- QC_line_fp2.mod
- QC_nccxv.mod
- QC_nlp_old.mod
- QC_nlp.mod
- QC_tan_cvx.mod
- QC_w_cvx.mod
- QPAC.mod
- SOC_cut_flex_cvx.mod
- SOC_cvx_fp.mod
- SOC_flex_cvx.mod
- SOC_w_cs.mod
- SOC_w_cvx_er.mod
- SOC_w_cvx_er2.mod
- SOC_w_cvx_lp.mod
- SOC_w_cvx.mod
- SOC_w_cyc3.mod
- SOC_w_inc_cvx.mod
- SOC_w_sdp3.mod
- SOC_w_tan_cvx.mod
- SOC_w_theta.mod
- SOC_w.mod
- SOC_w_lcvx.mod
Test Case Archive

The NICTA Energy System Test Case Archive

Carleton Coffrin1,2,3, Dan Gordon1, and Paul Scott1,2

1Optimisation Research Group, NICTA
2College of Engineering and Computer Science, Australian National University
3Computing and Information Systems, University of Melbourne

August 12, 2016

Abstract
In recent years the power systems research community has seen an explosion of work applying operations research techniques to challenging power network optimization problems. Regardless of the application under consideration, all of these works rely on power system test cases for evaluation and validation. However, many of the well established power system test cases were developed as far back as the 1960s with the aim of testing AC power flow algorithms. It is unclear if these power flow test cases are suitable for power system optimization studies. This report surveys all of the publicly available AC transmission system test cases, to the best of our knowledge, and assess their suitability for optimization.

https://arxiv.org/abs/1411.0359

UNCLASSIFIED
Brute Force R&D Example

The QC Relaxation: Theoretical and Computational Results on Optimal Power Flow


Power Formulations

TABLE III

<table>
<thead>
<tr>
<th>Test Case</th>
<th>QC</th>
<th>SDP</th>
<th>SOC</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>case162</td>
<td>22.21</td>
<td>16.06</td>
<td>36.79</td>
<td>14.51</td>
</tr>
<tr>
<td>case2737sop</td>
<td>17551.89</td>
<td>15100.34</td>
<td>9452.93</td>
<td>21.83</td>
</tr>
<tr>
<td>case9241</td>
<td>16891.28</td>
<td>17551.89</td>
<td>14.51</td>
<td>21.83</td>
</tr>
<tr>
<td>case3012wp</td>
<td>96573.10</td>
<td>96573.10</td>
<td>14.51</td>
<td>21.83</td>
</tr>
</tbody>
</table>

UNCLASSIFIED

Unexpected Insights!
Brute Force R&D Lessons Learned

• Reproducing previous works is challenging
  • working from a base implementation is very helpful
• AMPL was not built for this…
  • limited means to avoid excessive code replication
  • really hard to automate from the command line
  • limited licenses was the bottle neck in the All Formulations by All Instances Experiment
The Matpower Effect

• If a formulation is not implemented in Matpower, it **does not exist**
• At least for the majority of Power System PhD students
Inception of **PowerModels.jl**

- A baseline implementation of Power Flow formulations from the literature
  - Hopefully, mitigates the Matpower effect
- **Using Julia/JuMP** Resolves the AMPL Issues
  - Easy to automate at the command line
  - Fully open-source makes large-scale experiments easy
  - Julia enables advanced software design
My Dream

• I learn about a newly **proposed Power Flow formulation**

• It is **implemented** in PowerModels.jl and tested on all started test cases, **in 7 days or less**

• Lots of **code abstractions** in PowerModels.jl to enable this
The Value of Open-Source

https://lanl-ansi.github.io/PowerModels.jl/latest/

Software Versions

PowerModels.jl: v0.3.1-18-ga0785a2, a0785a28341986f92cebeee9a4be3482a6dd4d2e
Ipopt.jl: v0.2.6, 959b9c67e396a6e2307fc022d26b0d95692ee6a4
NESTA: v0.6.1, 466cd045d852c8c2cd86167b91ad8fa842ddf3da

Hardware: Dual Intel 2.1GHz CPUs, 128GB RAM

Typical Operating Conditions (TYP)

<table>
<thead>
<tr>
<th>Case Name</th>
<th>Nodes</th>
<th>Edges</th>
<th>AC ($/h)</th>
<th>QC Gap (%)</th>
<th>SOC Gap (%)</th>
<th>AC Time (sec.)</th>
<th>QC Time (sec.)</th>
<th>SOC Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nesta_case3_cc</td>
<td>3</td>
<td>3</td>
<td>2.0756e+02</td>
<td>1.55</td>
<td>1.62</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case3_cgs</td>
<td>3</td>
<td>3</td>
<td>1.0171e+02</td>
<td>1.69</td>
<td>1.69</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case3_lmbd</td>
<td>3</td>
<td>3</td>
<td>5.8126e+03</td>
<td>1.22</td>
<td>1.32</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case3_ch</td>
<td>3</td>
<td>5</td>
<td>9.8740e+01</td>
<td>100.01</td>
<td>100.01</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case4_gs</td>
<td>4</td>
<td>4</td>
<td>1.5643e+02</td>
<td>0.01</td>
<td>0.01</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case5_pjm</td>
<td>5</td>
<td>6</td>
<td>1.7552e+04</td>
<td>14.55</td>
<td>14.55</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nesta_case5_lmbd</td>
<td>5</td>
<td>7</td>
<td>3.0000e+00</td>
<td>0.01</td>
<td>0.01</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PowerModels.jl
Core Features
PowerModels.jl

CAUTION

Under Construction

UNCLASSIFIED
PowerModels.jl Structure

Average user not interested in the modeling details, just wants it to work.
Matpower Data is the R&D Standard

```matlab
function mpc = nesta_case3_lmbd
mpc.version = '2';
mpc.baseMVA = 100.0;

mpc.bus = [1 3 110.0 40.0 0.0 0.0 1 1.10000 -0.00000 240.0 1 1.10000 0.90000; 2 2 110.0 40.0 0.0 0.0 1 0.92617 7.25883 240.0 1 1.10000 0.90000; 3 2 95.0 50.0 0.0 0.0 1 0.90000 -17.26710 240.0 2 1.10000 0.90000];

mpc.gen = [1 148.067 54.697 1000.0 -1000.0 1.1 100.0 1 2000.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 2 170.006 -8.791 1000.0 -1000.0 0.92617 100.0 1 2000.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 3 0.0 -4.843 1000.0 -1000.0 0.9 100.0 1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0];

mpc.gencost = [2 0.0 0.0 3 0.110000 5.000000 0.000000; 2 0.0 0.0 3 0.085000 1.200000 0.000000; 2 0.0 0.0 3 0.000000 0.000000 0.000000];

mpc.branch = [1 3 0.065 0.62 0.45 9000.0 0.0 0.0 0.0 0.0 1 -30.0 30.0; 3 2 0.025 0.75 0.7 50.0 0.0 0.0 0.0 0.0 1 -30.0 30.0; 1 2 0.042 0.9 0.3 9000.0 0.0 0.0 0.0 0.0 1 -30.0 30.0];
```
Parsing Matpower Files

using PowerModels

network_data = PowerModels.parse_file("nesta_case3_lmbd.m")

println(network_data["bus"]["1"]["pd"])  
> 1.1

Parser supports user-defined extensions to the Matpower format

https://lanl-ansi.github.io/PowerModels.jl/latest/data.html
Your First PowerModel (OPF)

using PowerModels; using Ipopt
solver = IpoptSolver()

result = run_ac_opf("nesta_case3_lmbd.m", solver)
result = run_dc_opf("nesta_case3_lmbd.m", solver)
run_opf("nesta_case3_lmbd.m", ACPPowerModel, solver)
run_opf("nesta_case3_lmbd.m", DCPPowerModel, solver)
run_opf("nesta_case3_lmbd.m", SOCWRPowerModel, solver)

Non-Convex Form
Linear Approximation
Convex Relaxation
Inspecting the Results

```julia
using PowerModels; using Ipopt
solver = IpoptSolver()
result = run_opf("nesta_case3_lmbd.m", ACPPowerModel, solver)
println(result["objective"])  #> 5812.64293503618
println(result["solve_time"])  #> 0.009732971
println(result["solution"])  #> Dict{String,Any}(Pair{String,Any}("baseMVA",100.0),Pair{String,Any}("gen",Dict{String,Any}(Pair{String,Any}("1",Dict{String,Any}...))
```

julia dictionary (standard structure)
Modifying Network Data

using PowerModels; using Ipopt
solver = IpoptSolver()

network_data = PowerModels.parse_file("nesta_case3_lmbd.m")

network_data["bus"]["3"]['pd'] = 0.0
network_data["bus"]["3"]['qd'] = 0.0

result_1 = run_ac_opf(network_data, solver)

network_data["bus"]["3"]['pd'] = 1.0
network_data["bus"]["3"]['qd'] = 0.5

result_2 = run_ac_opf(network_data, solver)
using PowerModels; using Ipopt

# Base Non-Convex Model
run_pf("case5_pjm_tnep.m", ACPPowerModel, solver)
run_opf("case5_pjm_tnep.m", ACPPowerModel, solver)
run_ots("case5_pjm_tnep.m", ACPPowerModel, solver)
run_tnep("case5_pjm_tnep.m", ACPPowerModel, solver)

# Linear Approximation
run_pf("case5_pjm_tnep.m", DCPPowerModel, solver)
run_opf("case5_pjm_tnep.m", DCPPowerModel, solver)
run_ots("case5_pjm_tnep.m", DCPPowerModel, solver)
run_tnep("case5_pjm_tnep.m", DCPPowerModel, solver)
Solving Different Problems

using PowerModels; using Ipopt
solver = IpoptSolver()

# Base Non-Convex Model
run_pf("case5_pjm_tnep.m", ACPPowerModel, solver)
run_opf("case5_pjm_tnep.m", ACPPowerModel, solver)
run_ots("case5_pjm_tnep.m", ACPPowerModel, solver)
run_tnep("case5_pjm_tnep.m", ACPPowerModel, solver)

# Convex Relaxation
run_pf("case5_pjm_tnep.m", SOCWRPowerModel, solver)
run_opf("case5_pjm_tnep.m", SOCWRPowerModel, solver)
run_ots("case5_pjm_tnep.m", SOCWRPowerModel, solver)
run_tnep("case5_pjm_tnep.m", SOCWRPowerModel, solver)

This software design helps to organize 100s of possible Problem / Formulation combinations.
Where is JuMP?

```julia
using PowerModels; using Ipopt
solver = IpoptSolver()

result = run_opf("nesta_case3_lmbd.m", ACPPowerModel, solver)

pm = build_generic_model("nesta_case3_lmbd.m", ACPPowerModel, PowerModels.post_opf)
result = solve_generic_model(pm, solver)

pm = build_generic_model("nesta_case3_lmbd.m", ACPPowerModel, PowerModels.post_opf)
println(pm.model)  # show / modify the JuMP model
result = solve_generic_model(pm, solver)
```
function post_opf(pm::GenericPowerModel)
    variable_voltage(pm)
    variable_generation(pm)
    variable_line_flow(pm)
    objective_min_fuel_cost(pm)
    constraint_theta_ref(pm)
    constraint_voltage(pm)

    for (i,bus) in pm.ref[:bus]
        constraint_kcl_shunt(pm, bus)
    end

    for (i,branch) in pm.ref[:branch]
        constraint_ohms_yt_from(pm, branch)
        constraint_ohms_yt_to(pm, branch)
        constraint_phase_angle_difference(pm, branch)
        constraint_thermal_limit_from(pm, branch)
        constraint_thermal_limit_to(pm, branch)
    end

Unclassified
PowerModels.jl
Road Map
Versions Convention

vX.Y.Z

- Will be zero for some time
- breaking changes
- Non-breaking changes

UNCLASSIFIED
Versions Past and Planned

- **v0.1.0** (2016 Q2-Q3)
  - First draft (basically learning Julia / JuMP)
- **v0.2.0** (2016 Q3-Q4)
  - First public version, Thanks to Miles
- **v0.3.0** (2017 Q1-Present)
  - Significant engineering improvements
- **v0.4.0** (2017, I hope)
  - Massive renaming of stuff
  - Adding many more formulations from the literature
Contributions Welcome!

- This is a community resource for established problems and formulations
- Excited to add,
  - New problem classes
  - New formulations (especially complex ones, e.g. moment-based relaxations)
- Addressing anything in the github issues
Questions / Comments?

cjc@lanl.gov
Why have these ***PowerModel Things?

run_opf("nesta_case3_lmbd.m", ACPPowerModel, solver)
run_opf("nesta_case3_lmbd.m", DCPowerModel, solver)
run_opf("nesta_case3_lmbd.m", SOCWRPowerModel, solver)
...

Shared Modeling

GenericPowerModel

ACPPowerModel

AbstractWRPowerModel

DCPPowerModel

SOCWRPowerModel

QCWRPowerModel

Formulation

Specialized Modeling