Announcements

• Homework 6 is due now
• Course evaluations are now available. Goto pica.tamu.edu
  • Please do the evaluation!!
• Final exam is Wednesday Dec 12, 1 to 3pm
  • Closed book, closed notes. Two 8.5 by 11 inch notesheets allowed; calculators allowed
On use **Options, Constraint Options** to enable the enforcement of the Line/Transformer MVA limits.
LP OPF Introductory Example, cont
Example 6_23 Optimal Power Flow

In the example the load is gradually increased.
Locational Marginal Costs (LMPs)

- In an OPF solution, the bus LMPs tell the marginal cost of supplying electricity to that bus.
- The term “congestion” is used to indicate when there are elements (such as transmission lines or transformers) that are at their limits; that is, the constraint is binding.
- Without losses and without congestion, all the LMPs would be the same.
- Congestion or losses causes unequal LMPs.
- LMPs are often shown using color contours; a challenge is to select the right color range!
Example 6_23 Optimal Power Flow with Load Scale = 1.72

Total Hourly Cost: 10308.49 $/h
Total Area Load: 674.2 MW
Marginal Cost ($/MWh): 19.46 $/MWh
Example 6_23 Optimal Power Flow with Load Scale = 1.72

- LP Sensitivity Matrix (A Matrix)

The first row is the power balance constraint, while the second row is the line flow constraint. The matrix only has the line flows that are being enforced.
Example 6_23 Optimal Power Flow with Load Scale = 1.82

- This situation is infeasible, at least with available controls. There is a solution because the OPF is allowing one of the constraints to violate (at high cost)

<table>
<thead>
<tr>
<th>Area 1 MW Constraint</th>
<th>Base Case</th>
<th>Control</th>
<th>Control</th>
<th>Control</th>
<th>Control</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line from 2 to 5 clkt 1</td>
<td>Base Case</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Line from 4 to 5 clkt 1</td>
<td>Base Case</td>
<td>-0.002</td>
<td>1000.000</td>
<td>6</td>
<td>-0.024</td>
<td>0.140</td>
</tr>
</tbody>
</table>

| Total Hourly Cost: | 11297.88 $/h |
| Total Area Load: | 713.4 MW |
| Marginal Cost ($/MWh): | 235.47 $/MWh |
Generator Cost Curve Modeling

- LP algorithms require linear cost curves, with piecewise linear curves used to approximate a nonlinear cost function.
- Two common ways of entering cost information are:
  - Quadratic function
  - Piecewise linear curve
- The PowerWorld OPF supports both types.
Security Constrained OPF

• Security constrained optimal power flow (SCOPF) is similar to OPF except it also includes contingency constraints
  – Again the goal is to minimize some objective function, usually the current system cost, subject to a variety of equality and inequality constraints
  – This adds significantly more computation, but is required to simulate how the system is actually operated (with N-1 reliability)
• A common solution is to alternate between solving a power flow and contingency analysis, and an LP
With the inclusion of contingencies, there needs to be a distinction between what control actions must be done pre-contingent, and which ones can be done post-contingent.

- The advantage of post-contingent control actions is they would only need to be done in the unlikely event the contingency actually occurs.

Pre-contingent control actions are usually done for line overloads, while post-contingent control actions are done for most reactive power control and generator outage re-dispatch.
SCOPF Example

- We’ll again consider Example 6_23, except now it has been enhanced to include contingencies and we’ve also greatly increased the capacity on the line between buses 4 and 5.
PowerWorld SCOPF Application

Just click the button to solve

Number of times to redo contingency analysis
LP OPF and SCOPF Issues

- The LP approach is widely used for the OPF and SCOPF, particularly when implementing a dc power flow approach.
- A key issue is determining the number of binding constraints to enforce in the LP tableau.
  - Enforcing too many is time-consuming, enforcing too few results in excessive iterations.
- The LP approach is limited by the degree of linearity in the power system.
  - Real power constraints are fairly linear, reactive power constraints much less so.
An alternative to using the LP approach is to use Newton’s method, in which all the equations are solved simultaneously.

Key paper in area is:

Problem is:

\[
\text{Minimize } f(x) \\
\text{s.t. } g(x) = 0 \\
h(x) \leq 0
\]

For simplicity \( x \) represents all the variables and we can use \( h \) to impose limits on individual variables.
OPF Solution by Newton’s Method

- During the solution the inequality constraints are either binding (\(=0\)) or nonbinding (\(<0\))
  - The nonbinding constraints do not impact the final solution
- We’ll modify the problem to split the \(h\) vector into the binding constraints, \(h_1\) and the nonbinding constraints, \(h_2\)

Minimize \(f(x)\)

s.t. \(g(x)=0\)

\(h_1(x)=0\) \(h_2(x)<0\)
OPF Solution by Newton’s Method

• To solve first define the Lagrangian

\[ L(x, \lambda_1, \lambda_2) = f(x) + \mu^T g(x) + \lambda^T h_1(x) \]

Let \( z = [x \; \mu \; \lambda] \)

• A necessary condition for a minimum is that the gradient is zero

\[ \nabla L(z) = 0 = \begin{bmatrix} \frac{\partial L(z)}{\partial z_1} \\ \frac{\partial L(z)}{\partial z_2} \\ \vdots \end{bmatrix} \]

Both \( \mu \) and \( \lambda \) are Lagrange Multipliers
OPF Solution by Newton’s Method

- Solve using Newton’s method. To do this we need to define the Hessian matrix

\[ \nabla^2 L(z) = H(z) = \begin{bmatrix} \frac{\partial^2 L(z)}{\partial x_i \partial x_j} & \frac{\partial^2 L(z)}{\partial x_i \partial \mu_j} & \frac{\partial^2 L(z)}{\partial x_i \partial \lambda_j} \\ \frac{\partial^2 L(z)}{\partial \mu_i \partial x_j} & 0 & 0 \\ \frac{\partial^2 L(z)}{\partial \lambda_i \partial x_{ji}} & 0 & 0 \end{bmatrix} \]

- Because this is a second order method, as opposed to a first order linearization, it can better handle system nonlinearities
OPF Solution by Newton’s Method

• Solution is then via the standard Newton’s method. That is
  
  Set iteration counter $k=0$, set $k_{\text{max}}$
  
  Set convergence tolerance $\varepsilon$
  
  Guess $z^{(k)}$
  
  While $\left(\|\nabla L(z)\| \geq \varepsilon\right)$ and $\left(k < k_{\text{max}}\right)$
    
    $z^{(k+1)} = z^{(k)} - \left[H(z)\right]^{-1} \nabla L(z)$
  
    $k = k + 1$

End While

No iteration is needed for a quadratic function with linear constraints.
Example

• Solve
  Minimize $x_1^2 + x_2^2$ such that $x_1 + x_2 - 2 \geq 0$
  Solve initially assuming the constraint is binding
  \[
  L(x, \lambda) = x_1^2 + x_2^2 + \lambda (3x_1 + x_2 - 2)
  \]
  \[
  \nabla L(x, \lambda) = \begin{bmatrix}
  \frac{\partial L}{\partial x_1} \\
  \frac{\partial L}{\partial x_2} \\
  \frac{\partial L}{\partial \lambda}
  \end{bmatrix} = \begin{bmatrix}
  2x_1 + 3\lambda \\
  2x_2 + \lambda \\
  3x_1 + x_2 - 2
  \end{bmatrix}
  \]

No iteration is needed so any “guess” is fine. Pick $(1,1,0)$

\[
\nabla^2 L(x, \lambda) = H(x, \lambda) = \begin{bmatrix}
  2 & 0 & 3 \\
  0 & 2 & 1 \\
  3 & 1 & 0
  \end{bmatrix} \rightarrow \begin{bmatrix}
  x_1 \\
  x_2 \\
  \lambda
  \end{bmatrix} = \begin{bmatrix}
  1 \\
  1 \\
  0
  \end{bmatrix} - \begin{bmatrix}
  2 & 0 & 1 \\
  0 & 2 & 1 \\
  1 & 1 & 0
  \end{bmatrix}^{-1} \begin{bmatrix}
  2 \\
  2 \\
  2
  \end{bmatrix} = \begin{bmatrix}
  0.6 \\
  0.2 \\
  0.4
  \end{bmatrix}
  \]

Because $\lambda$ is positive the constraint is binding
Newton OPF Comments

- The Newton OPF has the advantage of being better able to handle system nonlinearities.
- There is still the issue of having to deal with determining which constraints are binding.
- The Newton OPF needs to implement second order derivatives plus all the complexities of the power flow solution.
  - The power flow starts off simple, but can rapidly get complex when dealing with actual systems.
- There is still the issue of handling integer variables.
Mixed-Integer Programming

- A mixed-integer program (MIP) is an optimization problem of the form

\[
\begin{align*}
\text{Minimize} & \quad c^T x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0 \\
\end{align*}
\]

where \( x \) = n-dimensional column vector
\( c \) = n-dimensional row vector
\( b \) = m-dimensional column vector
\( A \) = m×n matrix
some or all \( x_j \) integer
Mixed-Integer Programming

- The advances in the algorithms have been substantial

Speedups 1991-2008

Mixed-Integer Programming

• Suppose you were given the following choices?
  – Solve a MIP with today’s solution technology on a 1991 machine
  – Solve a MIP with a 1991 solution on a machine from today?
• The answer is to choose option 1, by a factor of approximately 300
• This leads to the current debate of whether the OPF (and SCOPF) should be solved using generic solvers or more customized code (which could also have quite good solvers!)

More General Solvers Overview

• OPF is currently an area of active research
• Many formulations and solution methods exist…
  – As do many *tools* for highly complex, large-scale computing!

• While many options exist, some may work better for certain problems or with certain programs you already use

• Consider experimenting with a new language/solver!
Gurobi and CPLEX

• Gurobi and CPLEX are two well-known commercial optimization solvers/packages for **linear programming (LP)**, quadratic programming (QP), quadratically constrained programming (QCP), and the mixed integer (MI) counterparts of LP/QP/QCP

• Gurobi and CPLEX are accessible through object-oriented interfaces (C++, Java, Python, C), matrix-oriented interfaces (MATLAB) and other modeling languages (AMPL, GAMS)
## Solver Comparison

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>LP/MILP</th>
<th>QP/MIQP</th>
<th>SOCP</th>
<th>SDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPLEX</strong>*</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><strong>GLPK</strong></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Gurobi</strong>*</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><strong>IPOPT</strong></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mosek</strong>*</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>SDPT3/SeDuMi</strong></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Linear programming can be solved by quadratic programming, which can be solved by second-order cone programming, which can be solved by semidefinite programming.
## Modeling Tools

<table>
<thead>
<tr>
<th></th>
<th>AMPL</th>
<th>CVX (Matlab)</th>
<th>GAMS</th>
<th>Pyomo (Python)</th>
<th>YALMIP (Matlab)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPLEX</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>GLPK</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Gurobi</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>IPOPT</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mosek</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDPT3/SeDuMi</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Introduction to AMPL

• **AMPL (A Mathematical Programming Language)** is a modeling language that enables the compact and logical representation of optimization models.

• Visit [http://www.ampl.com](http://www.ampl.com) to download and start using a student version.
  
  - To actually solve problems with AMPL, you’ll also need a solver!
  
  - CPLEX is a good one to start with (already included in Windows download)

• There is an AMPL book and many examples also available for download at [http://www.ampl.com](http://www.ampl.com)
Introduction to AMPL

- The **model file** (.mod) declares the **data parameters**, the **variables**, the **objective function** and the **constraints**
- The **data** is provided in the **data file** (.dat)
  - You don’t need to change the model for every small change in the data!

- Every declaration ends in a semicolon ;
- Comments begin with a pound sign #
- Parameters in the data file must first be declared in the model file
An AMPL Example

- Example 1 (lumber mill problem) from Lecture 24

\[
\begin{align*}
\text{Maximize} & \quad 100x_1 + 120x_2 \\
\text{s.t.} & \quad 2x_1 + 2x_2 \leq 8 \\
& \quad 3x_1 + 5x_2 \leq 15 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

Console output:

```
AMPL
ampl: option solver gurobi;
ampl: model example.mod
ampl: solve;
Gurobi 8.1.0: optimal solution; objective 430
2 simplex iterations
ampl: display x1,x2;
x1 = 2.5
x2 = 1.5
ampl: |
```
An (Improved) AMPL Example

- Example 1 (lumber mill problem) from Lecture 24

Can change the values in the .dat file, change solver, etc. with just a few quick clicks!
DCOPF in AMPL

- Example 6_23

Total Hourly Cost: 5724.32 $/h  
Load Scalar: 1.00  
Total Area Load: 392.0 MW
DCOPF in AMPL: Parameters

- LOTS of parameters (not all are used in this example, but the framework is there for more involved examples)
DCOPF in AMPL: Parameters, cont.

```ampl
param deg2rad := 3.14156/180;
param base_mva := 100;

# Make YBUS
set YBUS := setof{i in BUS} (i,i) union
    setof{(l,k,m) in BR} (k,m) union
    setof{(l,k,m) in BR} (m,k);

param B{(k,m) in YBUS} := if(k ==m) then (sum{(l,k,i) in BR} (0) +
    sum{(l,i,k) in BR} (0))
else if(k != m) then (sum{(l,k,m) in BR} 1/br_x[l,k,m] +
    sum{(l,m,k) in BR} 1/br_x[l,m,k]);
```

- Initialization

```ampl
# data init
for{i in BUS} {
    let bus_angle[i] := bus_angle0[i] * deg2rad;
    let pg0[i] := pg0[i]/base_mva;
    let pmin[i] := pmin[i]/base_mva;
    let pmax[i] := pmax[i]/base_mva;
    let bus_p_load[i] := bus_p_load[i]/base_mva;
}

# fixing variables
fix {i in BUS : bus_type[i] == 3} bus_angle[i];   # slack angle fixed
```
DCOPF in AMPL: Model

- **Objective: Polynomial**

```plaintext
## Available Controls
var bus_angle  {BUS};
var pflow  {BR};  # used for output

## generator MW outputs
var pg {k in BUS} = bus_p_load[k]  
  + sum{(k,m) in YBUS} (B[k,m]*(bus_angle[k] - bus_angle[m]));

## Objective ##
minimize cost: 0.016*(pg[1]*base_mva)^2+10*pg[1]*base_mva+373.50  
  + 0.018*(pg[2]*base_mva)^2+8*pg[2]*base_mva+403.60  
  + 0.018*(pg[4]*base_mva)^2+12*pg[4]*base_mva+253.20;

## Equality Constraints
# bus real power balance
subject to pL {k in BUS}: pg[k] - bus_p_load[k]  
  - sum{(k,m) in YBUS} (B[k,m]*(bus_angle[k] - bus_angle[m])) = 0;

## Inequality Constraints
# generator MW limits
subject to pG {k in BUS}:  
  pmin[k] <= pg[k] <= pmax[k];
```
DCOPF in AMPL: Data

- Bus, Branch, and Generator data saved in .txt files
DCOPF in AMPL: Solution

AMPL+Gurobi
DCOPF with polynomial objective

PowerWorld OPF solution

MATPOWER ACOPF with polynomial objective function
## MATPOWER

- Uses data stored in MATLAB struct

```matlab
>> runopf(Example6_23)

MATPOWER Version 4.1, 14-Dec-2011 -- AC Optimal Power Flow
MATLAB Interior Point Solver -- MIPS, Version 1.0, 07-Feb-2011
Converged!

Converged in 1.13 seconds
Objective Function Value = 5724.29 $/hr

<table>
<thead>
<tr>
<th>System Summary</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>How many?</th>
<th>How much?</th>
<th>P (MW)</th>
<th>Q (MVAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buses</td>
<td>5</td>
<td>1200.0</td>
<td>-29700.0 to 29700.0</td>
</tr>
<tr>
<td>Generators</td>
<td>3</td>
<td>1200.0</td>
<td>-29700.0 to 29700.0</td>
</tr>
<tr>
<td>Committed Gens</td>
<td>3</td>
<td>Generation (actual)</td>
<td>392.0</td>
</tr>
<tr>
<td>Loads</td>
<td>4</td>
<td>Load</td>
<td>392.0</td>
</tr>
<tr>
<td>Fixed</td>
<td>4</td>
<td>Fixed</td>
<td>392.0</td>
</tr>
<tr>
<td>Dispatchable</td>
<td>0</td>
<td>Dispatchable</td>
<td>-0.0 of -0.0</td>
</tr>
<tr>
<td>Shunts</td>
<td>0</td>
<td>Shunt (inj)</td>
<td>-0.0</td>
</tr>
<tr>
<td>Branches</td>
<td>7</td>
<td>Losses (I^2 * Z)</td>
<td>0.00</td>
</tr>
<tr>
<td>Transformers</td>
<td>0</td>
<td>Branch Charging (inj)</td>
<td>-</td>
</tr>
<tr>
<td>Inter-ties</td>
<td>0</td>
<td>Total Inter-tie Flow</td>
<td>0.0</td>
</tr>
<tr>
<td>Areas</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MATPOWER Solvers

• Section 6.5 of manual
  – Originally used MATLAB’s Optimization Toolbox
  – Now can use MINOPF/TSPOPF packages, IPOPT solver (open-source), CPLEX/MOSEK/Gurobi (for DC OPF), KNITRO (for AC OPFs)
  – Default: own primal-dual interior point method implementation MIPS (MATPOWER Interior Point Solver) for AC and DC (QP solver)

http://www.pserc.cornell.edu/matpower/
Questions

• What changes would you make to the .mod file to do a full ACOPF?
• What sensitivity analysis could you do by only changing the .dat/.txt files?
• What might you consider when comparing solvers?

• What tools best fit your needs?