— DRAFT —
MATPOWER 4.0b1 User's Manual

Ray D. Zimmerman

December 24, 2009

Contents

1 Introduction

1.1 What is MATPOWER?

MATPOWER is a package of Matlab M-files for solving power flow and optimal power flow problems. It is intended as a simulation tool for researchers and educators that is easy to use and modify. MATPOWER is designed to give the best performance possible while keeping the code simple to understand and modify. The MATPOWER home page can be found at:

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

1.2 Where did it come from?

MATPOWER was initially developed by Ray D. Zimmerman, Carlos E. Murillo-Sánchez and Deqiang Gan of PSERC[1] at Cornell University under the direction of Robert J. Thomas. The initial need for Matlab-based power flow and optimal power flow code was born out of the computational requirements of the PowerWeb project[2]. Many others have contributed to MATPOWER over the years and it continues to be developed and maintained under the direction of Ray Zimmerman.

1.3 Who may use it?

- MATPOWER is free. Anyone may use it.
- We make no warranties, express or implied. Specifically, we make no guarantees regarding the correctness MATPOWER's code or its fitness for any particular purpose.
- Any publications derived from the use of MATPOWER must cite MATPOWER.[3]
- Anyone may modify MATPOWER for their own use as long as the original copyright notices remain in place.
- MATPOWER may not be redistributed without written permission.
- Modified versions of MATPOWER, or works derived from MATPOWER, may not be distributed without written permission.

[1] http://www.pserc.cornell.edu/
[2] http://www.pserc.cornell.edu/powerweb/
[3] Use reference [1], or reference [9] until [1] is officially published.

2 Getting Started

2.1 System Requirements

To use MATPOWER 4.0b1 you will need:

- Matlab version 6.5 or later[4], available from The MathWorks[5]

For the hardware requirements, please refer to the system requirements for the version of Matlab you are using[6]. If the Matlab Optimization Toolbox is installed as well, MATPOWER enables an option to use it to solve optimal power flow problems, though this option is not recommended for most applications.

2.2 Installation

Installation and use of MATPOWER requires familiarity with the basic operation of Matlab, including setting up your Matlab path.

Step 1: Follow the download instructions on the MATPOWER home page[7]. You should end up with a file named `matpowerXXX.zip`, where `XXX` depends on the version of MATPOWER.

Step 2: Unzip the downloaded file. Move the resulting `matpowerXXX` directory to the location of your choice. These files should not need to be modified, so it is recommended that they be kept separate from your own code. We will use `$MATPOWER` to denote the path to this directory.

Step 3: Add the following directories to your Matlab path:

- `$MATPOWER` – core MATPOWER functions
- `$MATPOWER/t` – test scripts for MATPOWER
- (optional) sub-directories of `$MATPOWER/extras` – additional functionality and contributed code (see Appendix D for details).

[4] Although it is likely that many things work fine in earlier versions of Matlab 6, they are not supported. MATPOWER 3.2 required Matlab 6, MATPOWER 3.0 required Matlab 5 and MATPOWER 2.0 and earlier only required Matlab 4.

[5] http://www.mathworks.com/

[6] http://www.mathworks.com/support/sysreq/previous_releases.html

[7] http://www.pserc.cornell.edu/matpower/

Step 4: At the Matlab prompt, type `test_matpower` to run the test suite and verify that MATPOWER is properly installed and functioning. The result should resemble the following, possibly including extra tests, depending on the availablility of optional packages, solvers and extras.

```
>> test_matpower
t_loadcase.........ok
t_ext2int2ext.......ok
t_jacobian.........ok
t_hessian..........ok
t_hasPQcap.........ok
t_pf...............ok
t_opf_pdipm........ok
t_opf_scpdipm.......ok
t_opf_dc_ot........ok
t_opf_dc_pdipm......ok
t_opf_dc_scpdipm....ok
t_opf_userfcns......ok
t_runopf_w_res......ok
t_makePTDF.........ok
t_makeLODF.........ok
t_total_load........ok
t_scale_load........ok
All tests successful (1085 of 1085)
Elapsed time 4.73 seconds.
```

2.3 Running a Simulation

The primary functionality of MATPOWER is to solve power flow and optimal power flow (OPF) problems. This involves (1) preparing the input data defining the all of the relevant power system parameters, (2) invoking the function to run the simulation and (3) viewing and accessing the results that are printed to the screen and/or saved in output data structures or files.

2.3.1 Preparing Case Input Data

The input data for the case to be simulated are specified in a set of data matrices packaged as the fields of a Matlab struct, referred to as a "MATPOWER case" struct and conventionally denoted by the variable `mpc`. This struct is typically defined in a case file, either a function M-file whose return value is the `mpc` struct or a MAT-file that defines a variable named `mpc` when loaded. The main simulation routines,

whose names begin with `run` (e.g. `runpf`, `runopf`), accept either a file name or a MATPOWER case struct as an input.

Use `loadcase` to load the data from a case file into a struct if you want to make modifications to the data before passing it to the simulation.

```
>> mpc = loadcase(casefile);
```

See also `savecase` for writing a MATPOWER case struct to a case file.

The structure of the MATPOWER case data is described a bit further in Section 3.1 and the full details are documented in Appendix A and can be accessed at any time via the command `help caseformat`. The MATPOWER distribution also includes many example case files listed in Table 4.

2.3.2 Solving the Case

The solver is invoked by calling one of the main simulation functions, such as `runpf` or `runopf`, passing in a case file name or a case struct as the first argument. For example, to run a simple Newton power flow with default options on the 9-bus system defined in `case9.m`, at the Matlab prompt, type:

```
>> runpf('case9');
```

If, on the other hand, you wanted to load the 30-bus system data from `case30.m`, increase its real power demand at bus 2 to 30 MW, then run an AC optimal power flow with default options, this could be accomplished as follows:

```
>> define_constants;
>> mpc = loadcase('case30');
>> mpc.bus(2, PD) = 30;
>> runopf(mpc);
```

The `define_constants` in the first line is simply a convenience function that defines a number of variables to serve as named column indexes for the data matrices. In this example, it allows us to access the "real power demand" column of the `bus` matrix using the name `PD` without having to remember that it is the 3$^{\text{rd}}$ column.

Other top-level simulation functions are available for running DC versions of power flow and OPF, for running an OPF with the option for MATPOWER to shut down (decommit) expensive generators, etc. These functions are listed in Table 3 in Appendix C.

2.3.3 Accessing the Results

By default, the results of the simulation are pretty-printed to the screen, displaying a system summary, bus data, branch data and, for the OPF, binding constraint information. The bus data includes the voltage, angle and total generation and load at each bus. It also includes nodal prices in the case of the OPF. The branch data shows the flows and losses in each branch. These pretty-printed results can be saved to a file by providing a filename as the optional 3rd argument to the simulation function.

The solution is also stored in a `results` struct available as an optional return value from the simulation functions. This `results` struct is a superset of the MATPOWER case struct `mpc`, with additional columns added to some of the existing data fields and additional fields. The following example shows how simple it is, after running a DC OPF on the 118-bus system in `case118.m`, to access the final objective function value, the real power output of generator 6 and the power flow in branch 51.

```
>> results = rundcopf('case118');
>> final_objective = results.f;
>> gen6_output     = results.gen(6, PG);
>> branch51_flow   = results.branch(51, PF);
```

Full documentation for the content of the `results` struct can be found in Sections 4.3 and 5.6.

2.3.4 Setting Options

MATPOWER has many options for selecting among the available solution algorithms, controlling the behavior of the algorithms and determining the details of the pretty-printed output. These options are passed to the simulation routines as a MATPOWER options vector. The elements of the vector have names that can be used to set the corresponding value via the `mpoption` function. Calling `mpoption` with no arguments returns the default options vector, the vector used if none is explicitly supplied. Calling it with a set of name and value pairs modifies the default vector.

For example, the following code runs a power flow on the 300-bus example in `case300.m` using the fast-decoupled (XB version) algorithm, with verbose printing of the algorithm progress, but suppressing all of the pretty-printed output.

```
>> mpopt = mpoption('PF_ALG', 2, 'VERBOSE', 2, 'OUT_ALL', 0);
>> results = runpf('case300', mpopt);
```

To modify an existing options vector, for example, to turn the verbose option off and re-run with the remaining options unchanged, simply pass the existing options as the first argument to `mpoption`.

```
>> mpopt = mpoption(mpopt, 'VERBOSE', 0);
>> results = runpf('case300', mpopt);
```

See Appendix B or type:

```
>> help mpoption
```

for more information on MATPOWER's options.

2.4 Documentation

There are two primary sources of documentation for MATPOWER. The first is this manual, which gives an overview of MATPOWER's capabilities and structure and describes the modeling and formulations behind the code. It can be found in your MATPOWER distribution at `$MATPOWER/docs/manual.pdf`

The second is the built-in `help` command. As with Matlab's built-in functions and toolbox routines, you can type `help` followed by the name of a command or M-file to get help on that particular function. Nearly all of MATPOWER's M-files have such documentation and this should be considered the main reference for the calling options for each individual function. See Appendix C for a list of MATPOWER functions.

As an example, the help for `runopf` looks like:

```
>> help runopf
 RUNOPF  Runs an optimal power flow.

    Output arguments options:

    results = runopf(...)
    [results, success] = runopf(...)
    [baseMVA, bus, gen, gencost, branch, f, success, et] = runopf(...)

    Input arguments options:

    runopf(casename)
    runopf(casename, mpopt)
    runopf(casename, mpopt, fname)
```

```
runopf(casename, mpopt, fname, solvedcase)

Runs an optimal power flow and optionally returns the solved values in
the data matrices, the objective function value, a flag which is true if
the algorithm was successful in finding a solution, and the elapsed time
in seconds. Alternatively, the solution can be returned as fields in a
results struct and an optional success flag.

All input arguments are optional. If casename is provided it specifies
the name of the input data file or struct (see also 'help caseformat' and
'help loadcase') containing the opf data. The default value is 'case9'. If
the mpopt is provided it overrides the default MATPOWER options vector and
can be used to specify the solution algorithm and output options among
other things (see 'help mpoption' for details). If the 3rd argument is
given the pretty printed output will be appended to the file whose name is
given in fname. If solvedcase is specified the solved case will be written
to a case file in MATPOWER format with the specified name. If solvedcase
ends with '.mat' it saves the case as a MAT-file otherwise it saves it as
an M-file.
```

3 Modeling

MATPOWER employs all of the standard steady state models typically used for power flow analysis. The AC models are described first, then the simplified DC models. Internally, the magnitudes of all values are expressed in per unit and angles of complex quantities are expressed in radians. Due to the strengths of the Matlab programming language in handling matrices and vectors, the models and equations are presented here in matrix and vector form.

3.1 Data Formats

The data files used by MATPOWER are Matlab M-files or MAT-files which define and return a single Matlab struct. The M-file format is plain text that can be edited using any standard text editor. The fields of the struct are `baseMVA`, `bus`, `branch`, `gen` and optionally `gencost`, where `baseMVA` is a scalar and the rest are matrices. In the matrices, each row corresponds to a single bus, branch, or generator. The columns are similar to the columns in the standard IEEE CDF and PTI formats. The number of rows in `bus`, `branch` and `gen` are n_b, n_l and n_g, respectively. If present, `gencost` has either n_g or $2n_g$ rows, depending on whether it includes costs for reactive power or just real power. Full details of the MATPOWER case format are

documented in Appendix A and can be accessed from the Matlab command line by typing `help caseformat`.

3.2 Branches

All transmission lines, transformers and phase shifters are modeled with a common branch model, consisting of a standard π transmission line model, with series impedance $z_s = r_s + jx_s$ and total charging capacitance b_c, in series with an ideal phase shifting transformer. The transformer, whose tap ratio has magnitude τ and phase shift angle θ_{shift}, is located at the *from* end of the branch, as shown in Figure 1. The parameters r_s, x_s, b_c, τ and θ_{shift} are specified directly in columns 3, 4, 5, 9 and 10, respectively, of the corresponding row of the `branch` matrix.

The complex current injections i_f and i_t at the *from* and *to* ends of the branch, respectively, can be expressed in terms of the 2×2 branch admittance matrix Y_{br} and the respective terminal voltages v_f and v_t,

$$\begin{bmatrix} i_f \\ i_t \end{bmatrix} = Y_{br} \begin{bmatrix} v_f \\ v_t \end{bmatrix} \tag{1}$$

With the series admittance element in the π model denoted by $y_s = 1/z_s$, the branch admittance matrix can be written

$$Y_{br} = \begin{bmatrix} \left(y_s + j\frac{b_c}{2}\right)\frac{1}{\tau^2} & -y_s\frac{1}{\tau e^{-j\theta_{\text{shift}}}} \\ -y_s\frac{1}{\tau e^{j\theta_{\text{shift}}}} & y_s + j\frac{b_c}{2} \end{bmatrix} \tag{2}$$

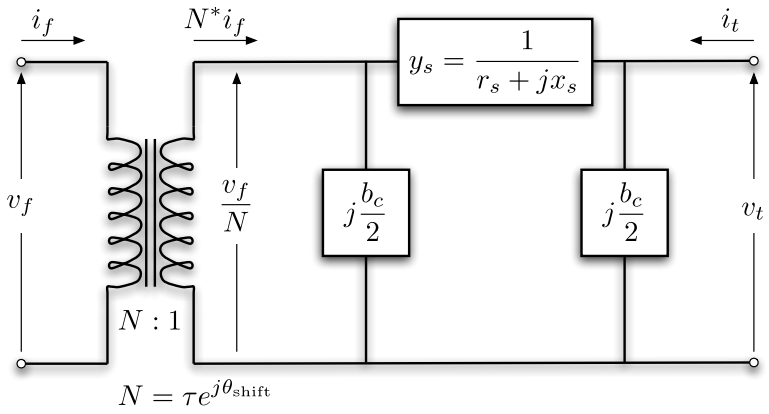

Figure 1: Branch Model

If the four elements of this matrix for branch i are labeled as follows,

$$Y_{br}^{i} = \begin{bmatrix} y_{ff}^{i} & y_{ft}^{i} \\ y_{tf}^{i} & y_{tt}^{i} \end{bmatrix} \tag{3}$$

then four $n_l \times 1$ vectors Y_{ff}, Y_{ft}, Y_{tf} and Y_{tt} can be constructed, where the i-th element of each comes from the corresponding element of Y_{br}^{i}. Furthermore, the $n_l \times n_b$ sparse connection matrices C_f and C_t used in building the system admittance matrices can be defined as follows. The $(i, j)^{\text{th}}$ element of C_f and the $(i, k)^{\text{th}}$ element of C_t are equal to 1 for each branch i, where branch i connects from bus j to bus k. All other elements of C_f and C_t are zero.

3.3 Generators

A generator is modeled as a complex power injection at a specific bus. For generator i, the injection is

$$s_g^i = p_g^i + jq_g^i \tag{4}$$

Let $S_g = P_g + jQ_g$ be the $n_g \times 1$ vector of these generator injections. The MW and MVAr equivalents (before conversion to p.u.) of p_g^i and q_g^i are specified in columns 2 and 3, respectively of row i of the `gen` matrix. A sparse $n_b \times n_g$ generator connection matrix C_g can be defined such that its $(i, j)^{\text{th}}$ element is 1 if generator j is located at bus i and 0 otherwise. The $n_b \times 1$ vector of all bus injections from generators can then be expressed as

$$S_{g,\text{bus}} = C_g \cdot S_g \tag{5}$$

3.4 Loads

Constant power loads are modeled as a specified quantity of real and reactive power consumed at a bus. For bus i, the load is

$$s_d^i = p_d^i + jq_d^i \tag{6}$$

and $S_d = P_d + jQ_d$ denotes the $n_b \times 1$ vector of complex loads at all buses. The MW and MVAr equivalents (before conversion to p.u.) of p_d^i and q_d^i are specified in columns 3 and 4, respectively of row i of the `bus` matrix.

Constant impedance and constant current loads are not implemented directly, but the constant impedance portions can be modeled as a shunt element described below. Dispatchable loads are modeled as negative generators and appear as negative values in S_g.

3.5 Shunt Elements

A shunt connected element such as a capacitor or inductor is modeled as a fixed impedance to ground at a bus. The admittance of the shunt element at bus i is given as

$$y_{sh}^i = g_{sh}^i + jb_{sh}^i \tag{7}$$

and $Y_{sh} = G_{sh} + jB_{sh}$ denotes the $n_b \times 1$ vector of shunt admittances at all buses. The parameters g_{sh}^i and b_{sh}^i are specified in columns 5 and 6, respectively, of row i of the `bus` matrix as equivalent MW (consumed) and MVAr (injected) at a nominal voltage magnitude of 1.0 p.u and angle of zero.

3.6 Network Equations

For a network with n_b buses, all constant impedance elements of the model are incorporated into a complex $n_b \times n_b$ bus admittance matrix Y_{bus} that relates the complex nodal current injections I_{bus} to the complex node voltages V.

$$I_{\text{bus}} = Y_{\text{bus}} V \tag{8}$$

Similarly, for a network with n_l branches, the $n_l \times n_b$ system branch admittance matrices Y_f and Y_t relate the bus voltages to the $n_l \times 1$ vectors I_f and I_t of branch currents at the *from* and *to* ends of all branches, respectively.

$$I_f = Y_f V \tag{9}$$
$$I_t = Y_t V \tag{10}$$

If $[\,\cdot\,]$ is used to denote an operator that takes an $n \times 1$ vector and creates the corresponding $n \times n$ diagonal matrix with the vector elements on the diagonal, these system admittance matrices can be formed as follows.

$$Y_f = [Y_{ff}]\, C_f + [Y_{ft}]\, C_t \tag{11}$$
$$Y_t = [Y_{tf}]\, C_f + [Y_{tt}]\, C_t \tag{12}$$
$$Y_{\text{bus}} = C_f^{\mathsf{T}} Y_f + C_t^{\mathsf{T}} Y_t + [Y_{sh}] \tag{13}$$

The current injections of (8)–(10) can be used to compute the corresponding complex power injections as functions of the complex bus voltages V.

$$S_{\text{bus}}(V) = [V]\, I_{\text{bus}}^* = [V]\, Y_{\text{bus}}^* V^* \tag{14}$$
$$S_f(V) = [C_f V]\, I_f^* = [C_f V]\, Y_f^* V^* \tag{15}$$
$$S_t(V) = [C_t V]\, I_t^* = [C_t V]\, Y_t^* V^* \tag{16}$$

The nodal bus injections are then matched to the injections from loads and generators to form the AC nodal power balance equations, expressed as a function of the complex bus voltages and generator injections in complex matrix form as

$$g_S(V, S_g) = S_{\text{bus}}(V) + S_d - C_g S_g = 0. \tag{17}$$

3.7 DC Modeling

The DC formulation is based on the same parameters, but with the following three additional simplifying assumptions:

- Branches can be considered lossless. In particular, branch resistances r_s and charging capacitances b_c are negligible.

$$y_s = \frac{1}{r_s + jx_s} \approx \frac{1}{jx_s}, \quad b_c \approx 0 \tag{18}$$

- All bus voltage magnitudes are close to 1 p.u.

$$v_i \approx e^{j\theta_i} \tag{19}$$

- Voltage angle differences across branches are small enough that

$$\sin(\theta_f - \theta_t - \theta_{\text{shift}}) \approx \theta_f - \theta_t - \theta_{\text{shift}}. \tag{20}$$

Substituting the first set of assumptions regarding branch parameters from (18), the branch admittance matrix in (2) approximates to

$$Y_{br} \approx \frac{1}{jx_s} \begin{bmatrix} \frac{1}{\tau^2} & -\frac{1}{\tau e^{-j\theta_{\text{shift}}}} \\ -\frac{1}{\tau e^{j\theta_{\text{shift}}}} & 1 \end{bmatrix} \tag{21}$$

Combining this and the second assumption with (1) yields the following approximation for i_f.

$$\begin{aligned} i_f &\approx \frac{1}{jx_s}\left(\frac{1}{\tau^2}e^{j\theta_f} - \frac{1}{\tau e^{-j\theta_{\text{shift}}}}e^{j\theta_t}\right) \\ &= \frac{1}{jx_s\tau}\left(\frac{1}{\tau}e^{j\theta_f} - e^{j(\theta_t+\theta_{\text{shift}})}\right) \end{aligned} \tag{22}$$

The approximate power flow is then derived as follows

$$\begin{aligned} s_f &= p_f + jq_f \\ &= v_f \cdot i_f^* \\ &\approx e^{j\theta_f} \cdot \frac{j}{x_s\tau}\left(\frac{1}{\tau}e^{-j\theta_f} - e^{-j(\theta_t+\theta_{\text{shift}})}\right) \\ &= \frac{j}{x_s\tau}\left(\frac{1}{\tau} - e^{j(\theta_f-\theta_t-\theta_{\text{shift}})}\right) \\ &= \frac{1}{x_s\tau}\Bigg[\sin(\theta_f-\theta_t-\theta_{\text{shift}}) \\ &\qquad + j\left(\frac{1}{\tau} - \cos(\theta_f-\theta_t-\theta_{\text{shift}})\right)\Bigg] \end{aligned} \tag{23}$$

Finally, extracting the real part and applying the last of the DC modeling assumptions from (20) yields

$$p_f \approx \frac{1}{x_s\tau}(\theta_f - \theta_t - \theta_{\text{shift}}) \tag{24}$$

As expected, given the lossless assumption, a similar derivation for p_t leads to $p_t = -p_f$.

The relationship between the real power flows and voltage angles for an individual branch i can then be summarized as

$$\begin{bmatrix} p_f \\ p_t \end{bmatrix} = B_{br}^i \begin{bmatrix} \theta_f \\ \theta_t \end{bmatrix} + P_{\text{shift}}^i \tag{25}$$

where $B_{br}^i = \begin{bmatrix} b_i & -b_i \\ -b_i & b_i \end{bmatrix}$, $P_{\text{shift}}^i = \theta_{\text{shift}}^i \begin{bmatrix} -b_i \\ b_i \end{bmatrix}$ and b_i is defined in terms of the series reactance and tap ratio for that branch as $b_i = \frac{1}{x_s^i \tau^i}$.

For a shunt element at bus i, the amount of complex power consumed is

$$\begin{aligned} s_{sh}^i &= v_i (y_{sh}^i v_i)^* \\ &\approx e^{j\theta_i}(g_{sh}^i - jb_{sh}^i)e^{-j\theta_i} \\ &= g_{sh}^i - jb_{sh}^i \end{aligned} \tag{26}$$

So the vector of real power consumed by shunt elements at all buses can be approximated by

$$P_{sh} \approx G_{sh} \tag{27}$$

With a DC model, the linear network equations relate real power to bus voltage angles, versus complex currents to complex bus voltages in the AC case. Let the $n_l \times 1$ vector B_{ff} be constructed similar to Y_{ff}, where the i-th element is b_i and let $P_{f,\text{shift}}$ be the $n_l \times 1$ vector whose i-th element is equal to $-\theta^i_{\text{shift}} b_i$. Then the nodal real power injections can be expressed as a linear function of Θ, the $n_b \times 1$ vector of bus voltage angles

$$P_{\text{bus}}(\Theta) = B_{\text{bus}}\Theta + P_{\text{bus,shift}} \tag{28}$$

where

$$P_{\text{bus,shift}} = (C_f - C_t)^{\mathsf{T}} P_{f,\text{shift}} \tag{29}$$

Similarly, the branch flows at the *from* ends of each branch are linear functions of the bus voltage angles

$$P_f(\Theta) = B_f\Theta + P_{f,\text{shift}} \tag{30}$$

and, due to the lossless assumption, the flows at the *to* ends are given by $P_t = -P_f$. The construction of the system B matrices is analogous to the system Y matrices for the AC model.

$$B_f = [B_{ff}]\,(C_f - C_t) \tag{31}$$

$$B_{\text{bus}} = (C_f - C_t)^{\mathsf{T}} B_f \tag{32}$$

The DC nodal power balance equations for the system can be expressed in matrix form as

$$g_P(\Theta, P_g) = B_{\text{bus}}\Theta + P_{\text{bus,shift}} + P_d + G_{sh} - C_g P_g = 0 \tag{33}$$

4 Power Flow

The standard power flow or loadflow problem involves solving for the set of voltages and flows in a network corresponding to a specified pattern of load and generation. MATPOWER includes solvers for both AC and DC power flow problems, both of which involve solving a set of equations of the form

$$g(x) = 0, \tag{34}$$

constructed by expressing a subset of the nodal power balance equations as functions of unknown voltage quantities.

All of MATPOWER's solvers exploit the sparsity of the problem and, except for Gauss-Seidel, scale well to very large systems. Currently, none of them include any automatic updating of transformer taps or other techniques to attempt to satisfy typical optimal power flow constraints, such as generator, voltage or branch flow limits.

4.1 AC Power Flow

In MATPOWER, by convention, a single generator bus is typically chosen as a reference bus to serve the roles of both a voltage angle reference and a real power slack. The voltage angle at the reference bus has a known value, but the real power generation at the slack bus is taken as unknown to avoid overspecifying the problem. The remaining generator buses are classified as PV buses, with the values of voltage magnitude and generator real power injection given. Since the loads P_d and Q_d are also given, all non-generator buses are PQ buses, with real and reactive injections fully specified. Let $\mathcal{I}_{\text{ref}}$, $\mathcal{I}_{\text{PV}}$ and $\mathcal{I}_{\text{PQ}}$ denote the sets of bus indices of the reference bus, PV buses and PQ buses, respectively.

In the traditional formulation of the AC power flow problem, the power balance equation in (17) is split into its real and reactive components, expressed as functions of the voltage angles Θ and magnitudes V_m and generator injections P_g and Q_g, where the load injections are assumed constant and given.

$$g_P(\Theta, V_m, P_g) = P_{\text{bus}}(\Theta, V_m) + P_d - C_g P_g \;\; = 0 \tag{35}$$

$$g_Q(\Theta, V_m, Q_g) = Q_{\text{bus}}(\Theta, V_m) + Q_d - C_g Q_g = 0 \tag{36}$$

For the AC power flow problem, the function $g(x)$ from (34) is formed by taking the left hand side of the real power balance equations (35) for all non-slack buses and the reactive power balance equations (36) for all PQ buses and plugging in the reference angle, the loads and the known generator injections and voltage magnitudes.

$$g(x) = \begin{bmatrix} g_P^{\{i\}}(\Theta, V_m, P_g) \\ g_Q^{\{j\}}(\Theta, V_m, Q_g) \end{bmatrix} \quad \begin{matrix} \forall i \in \mathcal{I}_{\text{PV}} \cup \mathcal{I}_{\text{PQ}} \\ \forall j \in \mathcal{I}_{\text{PQ}} \end{matrix} \tag{37}$$

The vector x consists of the remaining unknown voltage quantities, namely the voltage angles at all non-reference buses and the voltage magnitudes at PQ buses.

$$x = \begin{bmatrix} \theta_{\{i\}} \\ v_m^{\{j\}} \end{bmatrix} \quad \begin{matrix} \forall i \notin \mathcal{I}_{\text{ref}} \\ \forall j \in \mathcal{I}_{\text{PQ}} \end{matrix} \tag{38}$$

This yields a system of non-linear equations with $n_{pv} + 2n_{pq}$ equations and unknowns, where n_{pv} and n_{pq} are the number of PV and PQ buses, respectively. After solving for x, the remaining real power balance equation can be used to compute the generator real power injection at the slack bus. Similarly, the remaining $n_{pv} + 1$ reactive power balance equations yield the generator reactive power injections.

MATPOWER includes four different algorithms for solving the AC power flow problem. The default solver is based on a standard Newton's method [3] using a

polar form and a full Jacobian updated at each iteration. Each Newton step involves computing the mismatch $g(x)$, forming the Jacobian based on the sensitivities of these mismatches to changes in x and solving for an updated value of x by factorizing this Jacobian. This method is described in detail in many textbooks.

Also included are solvers based on variations of the fast-decoupled method [4], specifically, the XB and BX methods described in [5]. These solvers greatly reduce the amount of computation per iteration, by updating the voltage magnitudes and angles separately based on constant approximate Jacobians which are factored only once at the beginning of the solution process. These per-iteration savings, however, come at the cost of more iterations.

The fourth algorithm is the standard Gauss-Seidel method from Glimm and Stagg [6]. It has numerous disadvantages relative to the Newton method and is included primarily for academic interest.

By default, the AC power flow solvers simply solve the problem described above, ignoring any generator limits, branch flow limits, voltage magnitude limits, etc. However, there is an option (`ENFORCE_Q_LIMS`) that allows for the generator reactive power limits to be respected at the expense of the voltage setpoint. This is done in a rather brute force fashion by adding an outer loop around the AC power flow solution. If any generator has a violated reactive power limit, its reactive injection is fixed at the limit, the corresponding bus is converted to a PQ bus and the power flow is solved again. This procedure is repeated until there are no more violations. Note that this option is based solely on the `QMIN` and `QMAX` parameters for the generator and does not take into account the trapezoidal generator capability curves described in Section 5.4.3.

4.2 DC Power Flow

For the DC power flow problem [7], the vector x consists of the set of voltage angles at non-reference buses

$$x = \left[\theta_{\{i\}}\right], \quad \forall i \notin \mathcal{I}_{\text{ref}} \tag{39}$$

and equation (34) takes the form

$$B_{dc}x - P_{dc} = 0 \tag{40}$$

where B_{dc} is the $(n_b - 1) \times (n_b - 1)$ matrix obtained by simply eliminating from B_{bus} the row and column corresponding to the slack bus and reference angle, respectively. Given that the generator injections P_g are specified at all but the slack bus, P_{dc} can be found directly from the non-slack rows of (33).

The voltage angles in x are computed by a direct solution of the set of linear equations. The branch flows and slack bus generator injection are then calculated directly from the bus voltage angles via (30) and the appropriate row in (33), respectively.

4.3 `runpf`

In MATPOWER, a power flow is executed by calling `runpf`. In addition to printing output to the screen, which it does by default, `runpf` optionally returns the solution in a `results` struct.

```
>> results = runpf(mpc);
```

The `results` struct is a superset of the input MATPOWER case struct `mpc`, with some additional fields as well as additional columns in some of the existing data fields. The solution values are stored as follows:

Table 1: Power Flow Results

name	description
`results.bus(:, VM)`	bus voltage magnitudes
`results.bus(:, VA)`	bus voltage angles
`results.gen(:, PG)`	generator real power injections
`results.gen(:, QG)`	generator reactive power injections
`results.branch(:, PF)`	real power injected into from" end of branch
`results.branch(:, PT)`	real power injected into "to" end of branch
`results.branch(:, QF)`	reactive power injected into "from" end of branch
`results.branch(:, QT)`	reactive power injected into "to" end of branch
`results.success`	1 = solved successfully, 0 = unable to solve
`results.et`	computation time required for solution

4.4 Linear Shift Factors

The DC power flow model can also be used to compute the sensitivities of branch flows to changes in nodal real power injections, sometimes called injection shift factors (ISF) or generation shift factors [7]. These $n_l \times n_b$ sensitivity matrices, also called power transfer distribution factors or PTDF's, carry an implicit assumption about the slack distribution. If H is used to denote a PTDF matrix, then the element in

row i and column j, h_{ij}, represents the change in the real power flow in branch i given a unit increase in the power injected at bus j, *with the assumption* that the additional unit of power is extracted according to some specified slack distribution.

$$\Delta P_f = H \Delta P_{\text{bus}} \tag{41}$$

This slack distribution can be expressed as an $n_b \times 1$ vector w of non-negative weights whose elements sum to 1. Each element specifies the proportion of the slack taken up at each bus. For the special case of a single slack bus k, w is equal to the vector e_k. The corresponding PTDF matrix H_k can be constructed by first creating the $n_l \times (n_b - 1)$ matrix

$$\widetilde{H}_k = \widetilde{B}_f \cdot B_{dc}^{-1} \tag{42}$$

then inserting a column of zeros at column k. Here $\widetilde{B}_f$ and B_{dc} are obtained from B_f and B_{bus}, respectively, by eliminating their reference bus columns and, in the case of B_{dc}, removing row k corresponding to the slack bus.

The PTDF matrix H_w, corresponding to a general slack distribution w, can be obtained from any other PTDF, such as H_k, by subtracting w from each column, equivalent to the following simple matrix multiplication

$$H_w = H_k(I - w \cdot 1^{\mathsf{T}}) \tag{43}$$

These same linear shift factors may also be used to compute sensitivities of branch flows to branch outages, known as line outage distribution factors or LODF's [8]. Given a PTDF matrix H_w, the corresponding $n_l \times n_l$ LODF matrix L can be constructed as follows, where l_{ij} is the element in row i and column j, representing the change in flow in branch i (as a fraction of its initial flow) for an outage of branch j.

First, let H represent the matrix of sensitivities of branch flows to branch flows, found by multplying the PTDF matrix by the node-branch incidence matrix.

$$H = H_w(C_f - C_t)^{\mathsf{T}} \tag{44}$$

If h_{ij} is the sensitivity of flow in branch i with respect to flow in branch j, then l_{ij} can be expressed as

$$l_{ij} = \begin{cases} \dfrac{h_{ij}}{1 - h_{jj}} & i \neq j \\ -1 & i = j \end{cases} \tag{45}$$

MATPOWER includes functions for computing both the DC PTDF matrix and the corresponding LODF matrix for either a single slack bus k or a general slack distribution vector w. See the help for `makePTDF` and `makeLODF` for details.

5 Optimal Power Flow

MATPOWER includes code to solve both AC and DC versions of the optimal power flow problem. The standard version of each takes the following form.

$$\min_x f(x) \tag{46}$$

subject to

$$g(x) = 0 \tag{47}$$
$$h(x) \leq 0 \tag{48}$$
$$x_{\min} \leq x \leq x_{\max} \tag{49}$$

5.1 Standard AC OPF

The optimization vector x for the standard AC OPF problem consists of the $n_b \times 1$ vectors of voltage angles Θ and magnitudes V_m and the $n_g \times 1$ vectors of generator real and reactive power injections P_g and Q_g.

$$x = \begin{bmatrix} \Theta \\ V_m \\ P_g \\ Q_g \end{bmatrix} \tag{50}$$

The objective function (46) is simply a summation of individual polynomial cost functions f_P^i and f_Q^i of real and reactive power injections, respectively, for each generator.

$$\min_{\Theta, V_m, P_g, Q_g} \sum_{i=1}^{n_g} f_P^i(p_g^i) + f_Q^i(q_g^i) \tag{51}$$

The equality constraints in (47) are simply the full set of $2 \cdot n_b$ non-linear real and reactive power balance equations from (35) and (36). The inequality constraints (48) consist of two sets of n_l branch flow limits as non-linear functions of the bus voltage angles and magnitudes, one for the *from* end and one for the *to* end of each branch.

$$h_f(\Theta, V_m) = |F_f(\Theta, V_m)| - F_{\max} \leq 0 \tag{52}$$
$$h_t(\Theta, V_m) = |F_t(\Theta, V_m)| - F_{\max} \leq 0 \tag{53}$$

The flows are typically apparent power flows expressed in MVA, but can be real power or current flows, yielding the following three possible forms for the flow constraints

$$F_f(\Theta, V_m) = \begin{cases} S_f(\Theta, V_m), & \text{apparent power} \\ P_f(\Theta, V_m), & \text{real power} \\ I_f(\Theta, V_m), & \text{current} \end{cases} \tag{54}$$

where I_f is defined in (9), S_f in (15), $P_f = \Re\{S_f\}$ and the vector of flow limits $F_{\max}$ has the appropriate units for the type of constraint. Likewise for $F_t(\Theta, V_m)$.

The variable limits (49) include an equality constraint on any reference bus angle and upper and lower limits on all bus voltage magnitudes and real and reactive generator injections.

$$\theta_i^{\text{ref}} \le \theta_i \le \theta_i^{\text{ref}}, \qquad i \in \mathcal{I}_{\text{ref}} \tag{55}$$
$$v_m^{i,\min} \le v_m^i \le v_m^{i,\max}, \qquad i = 1 \dots n_b \tag{56}$$
$$p_g^{i,\min} \le p_g^i \le p_g^{i,\max}, \qquad i = 1 \dots n_g \tag{57}$$
$$q_g^{i,\min} \le q_g^i \le q_g^{i,\max}, \qquad i = 1 \dots n_g \tag{58}$$

5.2 Standard DC OPF

When using DC network modeling assumptions, the standard OPF problem above can be simplified to a quadratic program, with linear constraints and a quadratic cost function. In this case, the voltage magnitudes and reactive powers are eliminated from the problem completely and real power flows are modeled as linear functions of the voltage angles. The optimization variable is

$$x = \begin{bmatrix} \Theta \\ P_g \end{bmatrix} \tag{59}$$

and the overall problem reduces to the following form.

$$\min_{\Theta, P_g} \sum_{i=1}^{n_g} f_P^i(p_g^i) \tag{60}$$

subject to

$$g_P(\Theta, P_g) = B_{\text{bus}}\Theta + P_{\text{bus,shift}} + P_d + G_{sh} - C_g P_g = 0 \tag{61}$$

$$h_f(\Theta) = \quad B_f\Theta + P_{f,\text{shift}} - F_{\max} \le 0 \tag{62}$$
$$h_t(\Theta) = -B_f\Theta - P_{f,\text{shift}} - F_{\max} \le 0 \tag{63}$$

$$\theta_i^{\text{ref}} \le \theta_i \le \theta_i^{\text{ref}}, \qquad i \in \mathcal{I}_{\text{ref}} \tag{64}$$
$$p_g^{i,\min} \le p_g^i \le p_g^{i,\max}, \qquad i = 1 \dots n_g \tag{65}$$

5.3 Extended OPF Formulation

MATPOWER employs an extensible OPF structure [9] to allow the user to modify or augment the problem formulation without rewriting the portions that are shared with the standard OPF formulation. This is done through optional input parameters, preserving the ability to use pre-compiled solvers. The standard formulation is modified by introducing additional optional user-defined costs f_u, constraints, and variables z and can be written in the following form.

$$\min_{x,z} f(x) + f_u(x, z) \tag{66}$$

subject to

$$g(x) = 0 \tag{67}$$

$$h(x) \leq 0 \tag{68}$$

$$x_{\min} \leq x \leq x_{\max} \tag{69}$$

$$l \leq A \begin{bmatrix} x \\ z \end{bmatrix} \leq u \tag{70}$$

$$z_{\min} \leq z \leq z_{\max} \tag{71}$$

5.3.1 User-defined Costs

The user-defined cost function f_u is specified in terms of parameters H, C, N, $\hat{r}$, k, d and m. All of the parameters are $n_w \times 1$ vectors except the symmetric $n_w \times n_w$ matrix H and the $n_w \times (n_x + n_z)$ matrix N. The cost takes the form

$$f_u(x, z) = \frac{1}{2} w^{\mathsf{T}} H w + C^{\mathsf{T}} w \tag{72}$$

where w is defined in several steps as follows. First, a new vector u is created by applying a linear transformation N and shift $\hat{r}$ to the full set of optimization variables

$$r = N \begin{bmatrix} x \\ z \end{bmatrix}, \tag{73}$$

$$u = r - \hat{r}, \tag{74}$$

then a scaled function with a "dead zone" is applied to each element of u to produce the corresponding element of w.

$$w_i = \begin{cases} m_i f_{d_i}(u_i + k_i), & u_i < -k_i \\ 0, & -k_i \leq u_i \leq k_i \\ m_i f_{d_i}(u_i - k_i), & u_i > k_i \end{cases} \tag{75}$$

Here k_i specifies the size of the "dead zone", m_i is a simple scale factor and f_{d_i} is a pre-defined scalar function selected by the value of d_i. Currently, MATPOWER implements only linear and quadratic options

$$f_{d_i}(\alpha) = \begin{cases} \alpha, & \text{if } d_i = 1 \\ \alpha^2, & \text{if } d_i = 2 \end{cases} \tag{76}$$

as illustrated in Figure 2 and Figure 3, respectively.

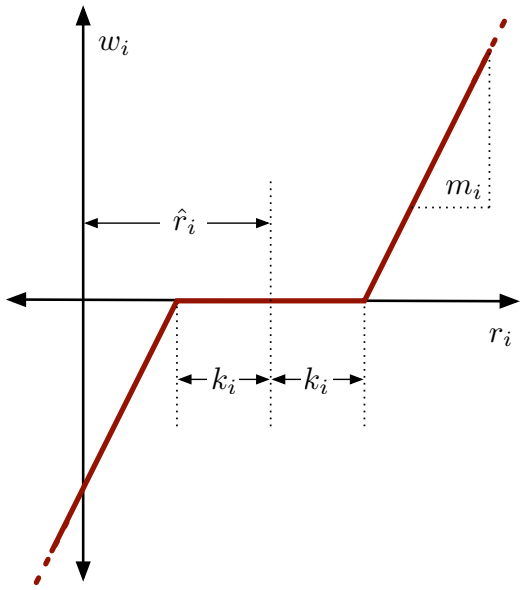

Figure 2: Relationship of w_i to r_i for $d_i = 1$ (linear option)

This form for f_u provides the flexibility to handle a wide range of costs, from simple linear functions of the optimization variables to scaled quadratic penalties on quantities, such as voltages, lying outside a desired range, to functions of linear combinations of variables, inspired by the requirements of price coordination terms found in the decomposition of large loosely coupled problems encountered in our own research.

Some limitations are imposed on the parameters in the case of the DC OPF since MATPOWER uses a generic quadratic programming (QP) solver for the optimization. In particular, $k_i = 0$ and $d_i = 1$ for all i, so the "dead zone" is not considered and only the linear option is available for f_{d_i}. As a result, for the DC case (75) simplifies to $w_i = m_i u_i$.

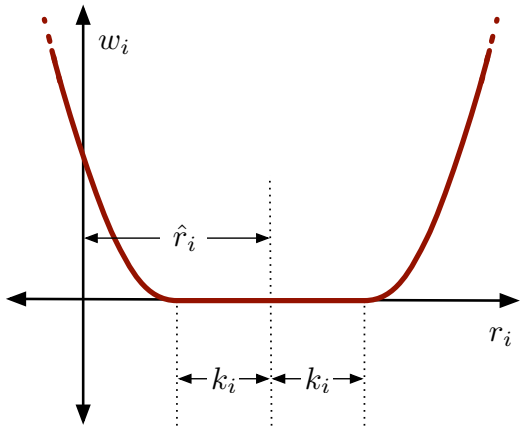

Figure 3: Relationship of w_i to r_i for $d_i = 2$ (quadratic option)

5.3.2 User-defined Constraints

The user-defined constraints (70) are general linear restrictions involving all of the optimization variables and are specified via matrix A and lower and upper bound vectors l and u. These parameters can be used to create equality constraints ($l_i = u_i$) or inequality constraints that are bounded below ($u_i = \infty$), bounded above ($l_i = \infty$) or bounded on both sides.

5.3.3 User-defined Variables

The creation of additional user-defined z variables is done implicitly based on the difference between the number of columns in A and the dimension of x. The optional vectors $z_{\min}$ and $z_{\max}$ are available to impose lower and upper bounds on z, respectively.

5.4 Standard Extensions

In addition to making this extensible OPF structure available to end users, MATPOWER also takes advantage of it internally to implement several additional capabilities.

5.4.1 Piecewise Linear Costs

The standard OPF formulation in (46)–(49) does not directly handle the non-smooth piecewise linear cost functions that typically arise from discrete bids and offers in electricity markets. When such cost functions are convex, however, they can be modeled using a constrained cost variable (CCV) method. The piecewise linear cost function $c(x)$ is replaced by a helper variable y and a set of linear constraints that form a convex "basin" requiring the cost variable y to lie in the epigraph of the function $c(x)$.

Figure 4 illustrates a convex n-segment piecewise linear cost function

$$c(x) = \begin{cases} m_1(x - x_1) + c_1, & x \leq x_1 \\ m_2(x - x_2) + c_2, & x_1 < x \leq x_2 \\ \vdots & \vdots \\ m_n(x - x_n) + c_n, & x_{n-1} < x \end{cases} \tag{77}$$

defined by a sequence of points (x_j, c_j), $j = 0 \ldots n$, where m_j denotes the slope of

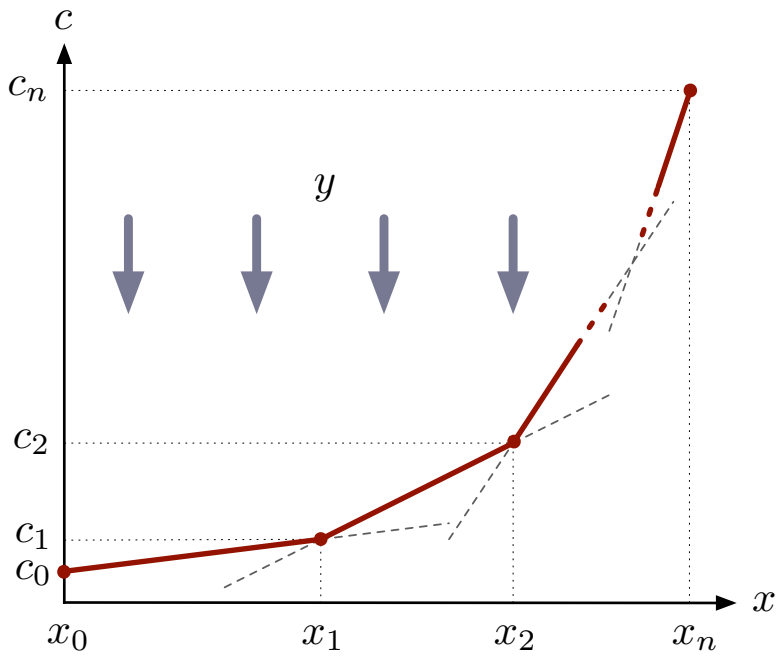

Figure 4: Constrained Cost Variable

the j-th segment,

$$m_j = \frac{c_j - c_{j-1}}{x_j - x_{j-1}}, \quad j = 1 \ldots n \tag{78}$$

and $x_0 < x_1 < \cdots < x_n$ and $m_1 \leq m_2 \leq \cdots < m_n$.

The "basin" corresponding to this cost function is formed by the following n constraints on the helper cost variable y.

$$y \geq m_j(x - x_j) + c_j, \quad j = 1 \ldots n \tag{79}$$

The cost term added to the objective function in place of $c(x)$ is simply the variable y.

MATPOWER uses this CCV approach internally to automatically convert any piecewise linear costs on real or reactive generation into the appropriate helper variable and corresponding set of constraints. All of MATPOWER's OPF solvers use the CCV approach with the exception of two that are part of the optional TSPOPF package [10], namely the step-controlled primal/dual interior point method (SCPDIPM) and the trust region based augmented Lagrangian method (TRALM), both of which use a cost smoothing technique instead [11].

5.4.2 Dispatchable Loads

A simple approach to dispatchable or price-sensitive loads is to model them as negative real power injections with associated negative costs. This is done by specifying a generator with a negative output, ranging from a minimum injection equal to the negative of the largest possible load to a maximum injection of zero.

Consider the example of a price-sensitive load whose marginal benefit function is shown in Figure 5. The demand p_d of this load will be zero for prices above λ_1, p_1 for prices between λ_1 and λ_2, and $p_1 + p_2$ for prices below λ_2.

This corresponds to a negative generator with the piecewise linear cost curve shown in Figure 6. Note that this approach assumes that the demand blocks can be partially dispatched or "split". Requiring blocks to be accepted or rejected in their entirety would pose a mixed-integer problem that is beyond the scope of the current MATPOWER implementation.

With an AC network model, there is also the question of reactive dispatch for such loads. Typically the reactive injection for a generator is allowed to take on any value within its defined limits. Since this is not normal load behavior, the model used in MATPOWER assumes that dispatchable loads maintain a constant power factor. When formulating the AC OPF problem, MATPOWER will automatically generate

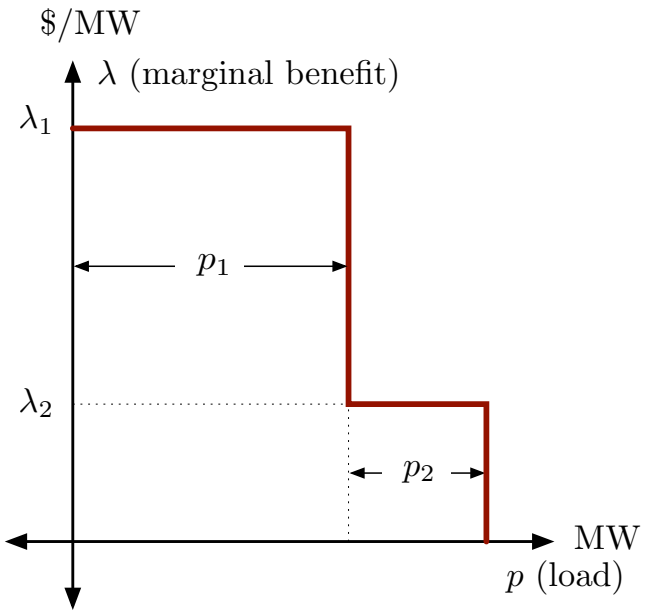

Figure 5: Marginal Benefit or Bid Function

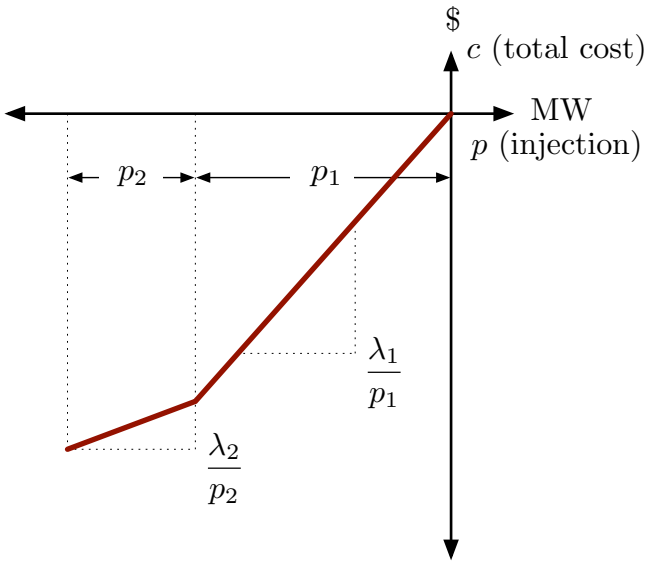

Figure 6: Total Cost Function for Negative Injection

an additional equality constraint to enforce a constant power factor for any "negative generator" being used to model a dispatchable load.

It should be noted that, with this definition of dispatchable loads as negative generators, if the negative cost corresponds to a benefit for consumption, minimizing the cost $f(x)$ of generation is equivalent to maximizing social welfare.

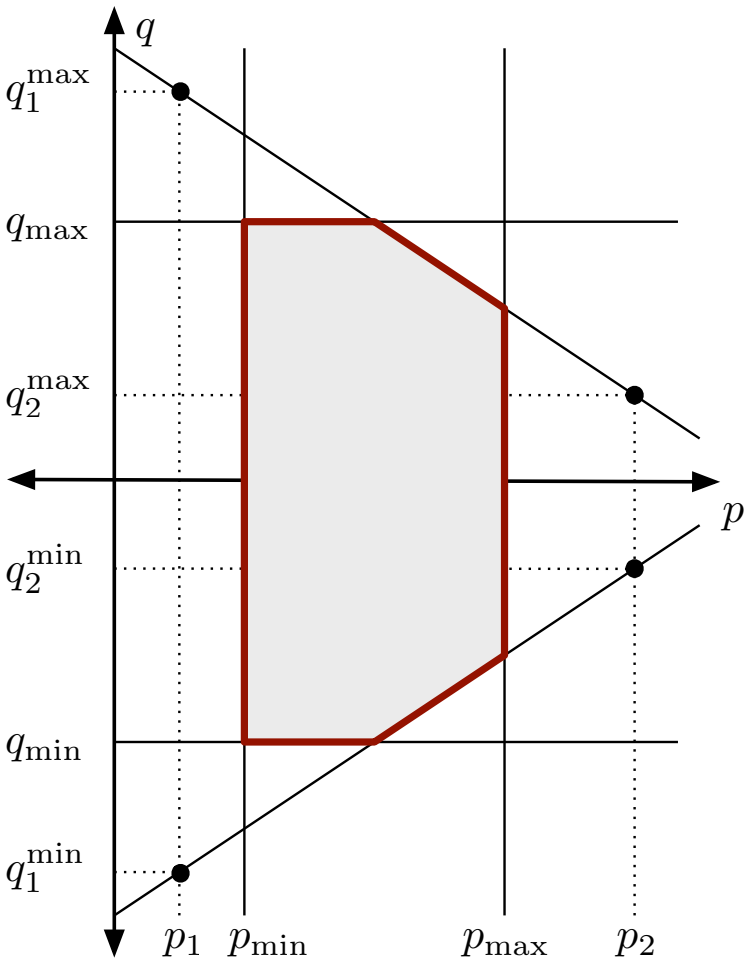

Figure 7: Generator P-Q Capability Curve

5.4.3 Generator Capability Curves

The typical AC OPF formulation includes box constraints on a generator's real and reactive injections, specified as simple lower and upper bounds on p ($p_{\min}$ and $p_{\max}$) and q ($q_{\min}$ and $q_{\max}$). On the other hand, the true P-Q capability curves of physical generators usually involve some tradeoff between real and reactive capability, so that it is not possible to produce the maximum real output and the maximum (or minimum) reactive output simultaneously. To approximate this tradeoff, MATPOWER includes the ability to add an upper and lower sloped portion to the standard box constraints as illustrated in Figure 7, where the shaded portion represents the feasible operating region for the unit.

The two sloped portions are constructed from the lines passing through the two pairs of points defined by the six parameters p_1, $q_1^{\min}$, $q_1^{\max}$, p_2, $q_2^{\min}$, and $q_2^{\max}$. If these six parameters are specified for a given generator, MATPOWER automatically constructs the corresponding additional linear inequality constraints on p and q for that unit.

5.4.4 Branch Angle Difference Limits

The difference between the bus voltage angle θ_f at the *from* end of a branch and the angle θ_t at the *to* end can be bounded above and below to act as a proxy for a transient stability limit, for example. If these limits are provided, MATPOWER creates the corresponding constraints on the voltage angle variables.

5.5 Solvers

Early versions of MATPOWER relied on Matlab's Optimization Toolbox [12] to provide the NLP and QP solvers needed to solve the AC and DC OPF problems, respectively. While they worked reasonably well for very small systems, they did not scale well to larger networks. Eventually, optional packages with additional solvers were added to improve performance, typically relying on Matlab extension (MEX) files implemented in Fortran or C and pre-compiled for each machine architecture. These MEX files are distributed as optional packages due to differences in terms of use. For DC optimal power flow, there is a MEX build [13] of the high performance BPMPD solver [14] for LP/QP problems. For the AC OPF problem, the MINOPF [15] and TSPOPF [10] packages provide solvers suitable for much larger systems. The former is based on MINOS [16] and the latter includes the primal-dual interior point and trust region based augmented Lagrangian methods described in [11].

Beginnning with version 4, MATPOWER also includes its own primal-dual interior point method (PDIPM) implemented in pure-Matlab code, derived from the MEX implementation of the algorithms described in [11]. If no optional packages are installed, this PDIPM solver will be used by default for both the AC OPF and as the QP solver used by the DC OPF. The AC OPF solver also employs a unique technique for efficiently forming the required Hessians via a few simple matrix operations. This solver has application to general non-linear optimization problems outside of MATPOWER and comes with a convenience wrapper function to make it trivial to set up and solve LP and QP problems.

5.6 `runopf`

In MATPOWER, an optimal power flow is executed by calling `runopf`. In addition to printing output to the screen, which it does by default, `runopf` optionally returns the solution in a `results` struct.

```
>> results = runopf(mpc);
```

The `results` struct is a superset of the input MATPOWER case struct `mpc`, with some additional fields as well as additional columns in some of the existing data fields. In addition to the solution values included in the `results` for a simple power flow, shown in Table 1 in Section 4.3, the following additional optimal power flow solution values are stored as follows:

Table 2: Optimal Power Flow Results

name	description
`results.bus(:, LAM_P)`	Lagrange multiplier on real power mismatch
`results.bus(:, LAM_Q)`	Lagrange multiplier on reactive power mismatch
`results.bus(:, MU_VMAX)`	Kuhn-Tucker multiplier on upper voltage limit
`results.bus(:, MU_VMIN)`	Kuhn-Tucker multiplier on lower voltage limit
`results.gen(:, MU_PMAX)`	Kuhn-Tucker multiplier on upper Pg limit
`results.gen(:, MU_PMIN)`	Kuhn-Tucker multiplier on lower Pg limit
`results.gen(:, MU_QMAX)`	Kuhn-Tucker multiplier on upper Qg limit
`results.gen(:, MU_QMIN)`	Kuhn-Tucker multiplier on lower Qg limit
`results.branch(:, MU_SF)`	Kuhn-Tucker multiplier on flow limit at “from” bus
`results.branch(:, MU_ST)`	Kuhn-Tucker multiplier on flow limit at “to” bus

6 Unit De-commitment Algorithm

The standard OPF formulation described in the previous section has no mechanism for completely shutting down generators which are very expensive to operate. Instead they are simply dispatched at their minimum generation limits. MATPOWER includes the capability to run an optimal power flow combined with a unit de-commitment for a single time period, which allows it to shut down these expensive units and find a least cost commitment and dispatch. To run this for `case30`, for example, type:

```
>> runuopf('case30')
```

MATPOWER uses an algorithm similar to dynamic programming to handle the de-commitment. It proceeds through a sequence of stages, where stage N has N generators shut down, starting with $N = 0$.

The algorithm proceeds as follows:

Step 1: Begin at stage zero ($N = 0$), assuming all generators are on-line with all limits in place.

Step 2: Solve a normal OPF. Save the solution as the current best.

Step 3: Go to the next stage, $N = N+1$. Using the best solution from the previous stage as the base case for this stage, form a candidate list of generators with minimum generation limits binding. If there are no candidates, skip to Step 5.

Step 4: For each generator on the candidate list, solve an OPF to find the total system cost with this generator shut down. Replace the current best solution with this one if it has a lower cost. If any of the candidate solutions produced an improvement, return to Step 3.

Step 5: Return the current best solution as the final solution.

7 Acknowledgments

The authors would like to acknowledge contributions from others who have helped make MATPOWER what it is today. Thanks to Chris DeMarco, one of our PSERC associates at the University of Wisconsin, for the technique for building the Jacobian matrix. Our appreciation to Bruce Wollenberg for all of his suggestions for improvements to version 1. The enhanced output functionality in version 2.0 is primarily due

to his input. Thanks also to Andrew Ward for code which helped us verify and test the ability of the OPF to optimize reactive power costs. Thanks to Alberto Borghetti for contributing code for the Gauss-Seidel power flow solver. Thanks to Roman Korab for data for the Polish system. Some state estimation code was contributed by James S. Thorp and Rui Bo contributed additional code for state estimation and continuation power flow. Thanks also to many others who have contributed code, bug reports and suggestions over the years. Last but not least, we would like to acknowledge the input and support of Bob Thomas throughout the development of MATPOWER.

Appendix A Data File Format

The data files used by MATPOWER are simply Matlab M-files or MAT-files which define and return the variables `baseMVA`, `bus`, `branch`, `gen`, `areas`, and `gencost`. The `baseMVA` variable is a scalar and the rest are matrices. Each row in the matrix corresponds to a single bus, branch, or generator. The columns are similar to the columns in the standard IEEE and PTI formats. The details of the specification of the MATPOWER case file can be found in the help for `caseformat.m`:

```
>> help caseformat
 CASEFORMAT    Defines the MATPOWER case file format.
    A MATPOWER case file is an M-file or MAT-file which defines the variables
    baseMVA, bus, gen, branch, areas (optional), and gencost (optional). With
    the exception of baseMVA, a scalar, each data variable is a matrix, where
    a row corresponds to a single bus, branch, gen, etc. The format of the
    data is similar to the PTI format described in
        http://www.ee.washington.edu/research/pstca/formats/pti.txt
    except where noted. An item marked with (+) indicates that it is included
    in this data but is not part of the PTI format. An item marked with (-) is
    one that is in the PTI format but is not included here. Those marked with
    (2) were added for version 2 of the case file format. The columns for
    each data matrix are given below.

    MATPOWER Case Version Information:
    A version 1 case file defined the data matrices directly. The last two,
    areas and gencost, were optional since they were not needed for running
    a simple power flow. In version 2, each of the data matrices are stored
    as fields in a struct. It is this struct, rather than the individual
    matrices, that is returned by a version 2 M-casefile. Likewise a version 2
    MAT-casefile stores a struct named 'mpc' (for MATPOWER case). The struct
    also contains a 'version' field so MATPOWER knows how to interpret the
    data. Any case file which does not return a struct, or any struct which
    does not have a 'version' field is considered to be in version 1 format.

    See also IDX_BUS, IDX_BRCH, IDX_GEN, IDX_AREA and IDX_COST regarding
    constants which can be used as named column indices for the data matrices.
    Also described in the first three are additional columns that are added
    to the bus, branch and gen matrices by the power flow and OPF solvers.

    Bus Data Format
        1   bus number (1 to 29997)
        2   bus type
                PQ bus          = 1
                PV bus          = 2
                reference bus   = 3
```

```
            isolated bus    = 4
    3   Pd, real power demand (MW)
    4   Qd, reactive power demand (MVAr)
    5   Gs, shunt conductance (MW (demanded) at V = 1.0 p.u.)
    6   Bs, shunt susceptance (MVAr (injected) at V = 1.0 p.u.)
    7   area number, 1-100
    8   Vm, voltage magnitude (p.u.)
    9   Va, voltage angle (degrees)
(-)     (bus name)
    10  baseKV, base voltage (kV)
    11  zone, loss zone (1-999)
(+) 12  maxVm, maximum voltage magnitude (p.u.)
(+) 13  minVm, minimum voltage magnitude (p.u.)

Generator Data Format
    1   bus number
(-)     (machine identifier, 0-9, A-Z)
    2   Pg, real power output (MW)
    3   Qg, reactive power output (MVAr)
    4   Qmax, maximum reactive power output (MVAr)
    5   Qmin, minimum reactive power output (MVAr)
    6   Vg, voltage magnitude setpoint (p.u.)
(-)     (remote controlled bus index)
    7   mBase, total MVA base of this machine, defaults to baseMVA
(-)     (machine impedance, p.u. on mBase)
(-)     (step up transformer impedance, p.u. on mBase)
(-)     (step up transformer off nominal turns ratio)
    8   status,  >  0 - machine in service
                 <= 0 - machine out of service
(-)     (% of total VAr's to come from this gen in order to hold V at
            remote bus controlled by several generators)
    9   Pmax, maximum real power output (MW)
    10  Pmin, minimum real power output (MW)
(2) 11  Pc1, lower real power output of PQ capability curve (MW)
(2) 12  Pc2, upper real power output of PQ capability curve (MW)
(2) 13  Qc1min, minimum reactive power output at Pc1 (MVAr)
(2) 14  Qc1max, maximum reactive power output at Pc1 (MVAr)
(2) 15  Qc2min, minimum reactive power output at Pc2 (MVAr)
(2) 16  Qc2max, maximum reactive power output at Pc2 (MVAr)
(2) 17  ramp rate for load following/AGC (MW/min)
(2) 18  ramp rate for 10 minute reserves (MW)
(2) 19  ramp rate for 30 minute reserves (MW)
(2) 20  ramp rate for reactive power (2 sec timescale) (MVAr/min)
(2) 21  APF, area participation factor

Branch Data Format
```

```
      1   f, from bus number
      2   t, to bus number
  (-)     (circuit identifier)
      3   r, resistance (p.u.)
      4   x, reactance (p.u.)
      5   b, total line charging susceptance (p.u.)
      6   rateA, MVA rating A (long term rating)
      7   rateB, MVA rating B (short term rating)
      8   rateC, MVA rating C (emergency rating)
      9   ratio, transformer off nominal turns ratio ( = 0 for lines )
          (taps at 'from' bus, impedance at 'to' bus,
           i.e. if r = x = 0, then ratio = Vf / Vt)
     10   angle, transformer phase shift angle (degrees), positive => delay
  (-)     (Gf, shunt conductance at from bus p.u.)
  (-)     (Bf, shunt susceptance at from bus p.u.)
  (-)     (Gt, shunt conductance at to bus p.u.)
  (-)     (Bt, shunt susceptance at to bus p.u.)
     11   initial branch status, 1 - in service, 0 - out of service
  (2) 12  minimum angle difference, angle(Vf) - angle(Vt) (degrees)
  (2) 13  maximum angle difference, angle(Vf) - angle(Vt) (degrees)

(+) Generator Cost Data Format
      NOTE: If gen has n rows, then the first n rows of gencost contain
      the cost for active power produced by the corresponding generators.
      If gencost has 2*n rows then rows n+1 to 2*n contain the reactive
      power costs in the same format.
      1   model, 1 - piecewise linear, 2 - polynomial
      2   startup, startup cost in US dollars
      3   shutdown, shutdown cost in US dollars
      4   n, number of cost coefficients to follow for polynomial
          cost function, or number of data points for piecewise linear
      5 and following, cost data defining total cost function
          For polynomial cost:
                  c2, c1, c0
          where the polynomial is c0 + c1*P + c2*P^2
          For piecewise linear cost:
                  x0, y0, x1, y1, x2, y2, ...
          where x0 < x1 < x2 < ... and the points (x0,y0), (x1,y1),
          (x2,y2), ... are the end- and break-points of the cost function.

(+) Area Data Format (deprecated)
    (this data is not used by MATPOWER and is no longer necessary for
     version 2 case files with OPF data).
      1   i, area number
      2   price_ref_bus, reference bus for that area
```

Appendix B MATPOWER Options

MATPOWER uses an options vector to control the many options available. It is similar to the options vector produced by the `foptions` function in early versions of Matlab's Optimization Toolbox. The primary difference is that modifications can be made by option name, as opposed to having to remember the index of each option. The default MATPOWER options vector is obtained by calling `mpoption` with no arguments. So, typing:

```
>> runopf('case30', mpoption)
```

is another way to run the OPF solver with the all of the default options.

The MATPOWER options vector controls the following:

- power flow algorithm
- power flow termination criterion
- power flow options (e.g. enforcing of reactive power generation limits)
- OPF algorithm
- OPF default algorithms for different cost models
- OPF cost conversion parameters
- OPF termination criterion
- OPF options (e.g. active vs. apparent power vs. current for line limits)
- verbose level
- printing of results

The details are given below:

```
>> help mpoption
 MPOPTION  Used to set and retrieve a MATPOWER options vector.

    opt = mpoption
        returns the default options vector

    opt = mpoption(name1, value1, name2, value2, ...)
        returns the default options vector with new values for up to 7
```

```
    options, name# is the name of an option, and value# is the new
    value. Example: options = mpoption('PF_ALG', 2, 'PF_TOL', 1e-4)

opt = mpoption(opt, name1, value1, name2, value2, ...)
    same as above except it uses the options vector opt as a base
    instead of the default options vector.

The currently defined options are as follows:

   idx - NAME, default          description [options]
   ---   -------------          -------------------------------------
power flow options
    1  - PF_ALG, 1              power flow algorithm
        [   1 - Newton's method                                      ]
        [   2 - Fast-Decoupled (XB version)                          ]
        [   3 - Fast-Decoupled (BX version)                          ]
        [   4 - Gauss Seidel                                         ]
    2  - PF_TOL, 1e-8           termination tolerance on per unit
                                P & Q mismatch
    3  - PF_MAX_IT, 10          maximum number of iterations for
                                Newton's method
    4  - PF_MAX_IT_FD, 30       maximum number of iterations for
                                fast decoupled method
    5  - PF_MAX_IT_GS, 1000     maximum number of iterations for
                                Gauss-Seidel method
    6  - ENFORCE_Q_LIMS, 0      enforce gen reactive power limits
                                at expense of |V|
        [    0 - do NOT enforce limits                               ]
        [    1 - enforce limits, simultaneous bus type conversion    ]
        [    2 - enforce limits, one-at-a-time bus type conversion   ]
    10 - PF_DC, 0               use DC power flow formulation, for
                                power flow and OPF
        [    0 - use AC formulation & corresponding algorithm opts   ]
        [    1 - use DC formulation, ignore AC algorithm options     ]
OPF options
    11 - OPF_ALG, 0             solver to use for AC OPF
        [    0 - choose default solver available in the following    ]
        [        order, 500, 540, 560                                ]
        [  300 - generalized formulation, constr                     ]
        [  320 - generalized formulation, dense LP                   ]
        [  340 - generalized formulation, sparse LP (relaxed)        ]
        [  360 - generalized formulation, sparse LP (full)           ]
        [  500 - generalized formulation, MINOS                      ]
        [  520 - generalized formulation, fmincon                    ]
        [  540 - generalized formulation, PDIPM                      ]
        [        primal/dual interior point method                   ]
```

```
        [  545 - generalized formulation (except CCV), SCPDIPM      ]
        [       step-controlled primal/dual interior point method  ]
        [  550 - generalized formulation (except CCV), TRALM        ]
        [       trust region based augmented Langrangian method    ]
        [  560 - generalized formulation, PDIPM (pure Matlab)       ]
        [       primal/dual interior point method                  ]
        [  565 - generalized formulation, SCPDIPM (pure Matlab)     ]
        [       step-controlled primal/dual interior point method  ]
    16 - OPF_VIOLATION, 5e-6    constraint violation tolerance
    17 - CONSTR_TOL_X, 1e-4     termination tol on x for copf & fmincopf
    18 - CONSTR_TOL_F, 1e-4     termination tol on F for copf & fmincopf
    19 - CONSTR_MAX_IT, 0       max number of iterations for copf & fmincopf
                                [       0 => 2*nb + 150             ]
    20 - LPC_TOL_GRAD, 3e-3     termination tolerance on gradient for lpopf
    21 - LPC_TOL_X, 1e-4        termination tolerance on x (min step size)
                                for lpopf
    22 - LPC_MAX_IT, 400        maximum number of iterations for lpopf
    23 - LPC_MAX_RESTART, 5     maximum number of restarts for lpopf
    24 - OPF_FLOW_LIM, 0        qty to limit for branch flow constraints
        [   0 - apparent power flow (limit in MVA)                  ]
        [   1 - active power flow (limit in MW)                     ]
        [   2 - current magnitude (limit in MVA at 1 p.u. voltage   ]
    25 - OPF_IGNORE_ANG_LIM, 0  ignore angle difference limits for branches
                                even if specified       [   0 or 1  ]
    26 - OPF_ALG_DC, 0          solver to use for DC OPF
        [    0 - choose default solver from available solvers in    ]
        [        the following order, 100, 200                      ]
        [  100 - BPMPD_MEX                                          ]
        [  200 - PDIPM (pure Matlab), primal/dual interior pt method]
        [  250 - SCPDIPM (pure Matlab), step-controlled PDIPM       ]
        [  300 - Optimization Tbx, quadprog(), linprog()            ]
output options
    31 - VERBOSE, 1             amount of progress info printed
        [   0 - print no progress info                              ]
        [   1 - print a little progress info                        ]
        [   2 - print a lot of progress info                        ]
        [   3 - print all progress info                             ]
    32 - OUT_ALL, -1            controls printing of results
        [  -1 - individual flags control what prints                ]
        [   0 - don't print anything                                ]
        [       (overrides individual flags, except OUT_RAW)        ]
        [   1 - print everything                                    ]
        [       (overrides individual flags, except OUT_RAW)        ]
    33 - OUT_SYS_SUM, 1         print system summary    [   0 or 1  ]
    34 - OUT_AREA_SUM, 0        print area summaries    [   0 or 1  ]
    35 - OUT_BUS, 1             print bus detail        [   0 or 1  ]
```

```
    36 - OUT_BRANCH, 1          print branch detail     [   0 or 1  ]
    37 - OUT_GEN, 0             print generator detail  [   0 or 1  ]
                                (OUT_BUS also includes gen info)
    38 - OUT_ALL_LIM, -1        control constraint info output
        [  -1 - individual flags control what constraint info prints]
        [   0 - no constraint info (overrides individual flags)     ]
        [   1 - binding constraint info (overrides individual flags)]
        [   2 - all constraint info (overrides individual flags)    ]
    39 - OUT_V_LIM, 1           control output of voltage limit info
        [   0 - don't print                                         ]
        [   1 - print binding constraints only                      ]
        [   2 - print all constraints                               ]
        [   (same options for OUT_LINE_LIM, OUT_PG_LIM, OUT_QG_LIM) ]
    40 - OUT_LINE_LIM, 1        control output of line limit info
    41 - OUT_PG_LIM, 1          control output of gen P limit info
    42 - OUT_QG_LIM, 1          control output of gen Q limit info
    43 - OUT_RAW, 0             print raw data for Perl database
                                interface code          [   0 or 1  ]
other options
    51 - SPARSE_QP, 1           pass sparse matrices to QP and LP
                                solvers if possible     [   0 or 1  ]
fmincon options
    55 - FMC_ALG, 1             algorithm used by fmincon for OPF
                                for Optimization Toolbox 4 and later
         [  1 - active-set                                          ]
         [  2 - interior-point, w/default 'bfgs' Hessian approx     ]
         [  3 - interior-point, w/ 'lbfgs' Hessian approx           ]
         [  4 - interior-point, w/exact user-supplied Hessian       ]
         [  5 - interior-point, w/Hessian via finite differences    ]

MINOPF options
    61 - MNS_FEASTOL, 0 (1E-3)  primal feasibility tolerance,
                                set to value of OPF_VIOLATION by default
    62 - MNS_ROWTOL, 0  (1E-3)  row tolerance
                                set to value of OPF_VIOLATION by default
    63 - MNS_XTOL, 0     (1E-3) x tolerance
                                set to value of CONSTR_TOL_X by default
    64 - MNS_MAJDAMP, 0 (0.5)   major damping parameter
    65 - MNS_MINDAMP, 0 (2.0)   minor damping parameter
    66 - MNS_PENALTY_PARM, 0 (1.0)  penalty parameter
    67 - MNS_MAJOR_IT, 0 (200)  major iterations
    68 - MNS_MINOR_IT, 0 (2500) minor iterations
    69 - MNS_MAX_IT, 0 (2500)   iterations limit
    70 - MNS_VERBOSITY, -1
        [  -1 - controlled by VERBOSE flag                          ]
        [   0 - print nothing                                       ]
```

```
            [   1 - print only termination status message              ]
            [   2 - print termination status and screen progress       ]
            [   3 - print screen progress, report file (usually fort.9) ]
        71 - MNS_CORE, 1200 * nb + 2 * (nb + ng)^2
        72 - MNS_SUPBASIC_LIM, 0 (2*nb + 2*ng) superbasics limit
        73 - MNS_MULT_PRICE, 0 (30) multiple price

    PDIPM, SC-PDIPM, and TRALM options
        81 - PDIPM_FEASTOL, 0       feasibility (equality) tolerance for
                                    PDIPM and SC-PDIPM
                                    set to value of OPF_VIOLATION by default
        82 - PDIPM_GRADTOL, 1e-6    gradient tolerance for PDIPM
                                    and SC-PDIPM
        83 - PDIPM_COMPTOL, 1e-6    complementary condition (inequality)
                                    tolerance for PDIPM and SC-PDIPM
        84 - PDIPM_COSTTOL, 1e-6    optimality tolerance for PDIPM and
                                    SC-PDIPM
        85 - PDIPM_MAX_IT,  150     maximum number of iterations for
                                    PDIPM and SC-PDIPM
        86 - SCPDIPM_RED_IT, 20     maximum number of SC-PDIPM reductions
                                    per iteration
        87 - TRALM_FEASTOL, 0       feasibility tolerance for TRALM
                                    set to value of OPF_VIOLATION by default
        88 - TRALM_PRIMETOL, 5e-4   prime variable tolerance for TRALM
        89 - TRALM_DUALTOL, 5e-4    dual variable tolerance for TRALM
        90 - TRALM_COSTTOL, 1e-5    optimality tolerance for TRALM
        91 - TRALM_MAJOR_IT, 40     maximum number of major iterations
        92 - TRALM_MINOR_IT, 100    maximum number of minor iterations
        93 - SMOOTHING_RATIO, 0.04  piecewise linear curve smoothing ratio
                                    used in SC-PDIPM and TRALM
```

A typical usage of the options vector might be as follows: Get the default options vector:

```
>> opt = mpoption;
```

Use the fast-decoupled method to solve power flow:

```
>> opt = mpoption(opt, 'PF_ALG', 2);
```

Display only system summary and generator info:

```
>> opt = mpoption(opt, 'OUT_BUS', 0, 'OUT_BRANCH', 0, 'OUT_GEN', 1);
```

Show all progress info:

```
>> opt = mpoption(opt, 'VERBOSE', 3);
```

Now, run a bunch of power flows using these settings:

```
>> runpf('case57', opt)
>> runpf('case118', opt)
>> runpf('case300', opt)
```

Appendix C Summary of MATPOWER Functions

This appendix lists the all of the functions that MATPOWER provides. In most cases, the function is found in a Matlab M-file of the same name in the top-level of the distribution and the `.m` extension is omitted from this listing.

Table 3: Top-Level Simulation Functions

name	description
`runpf`	power flow[a]
`runopf`	optimal power flow[a]
`runuopf`	optimal power flow with unit-decommitment[a]
`rundcpf`	DC power flow[b]
`rundcopf`	DC optimal power flow[b]
`runduopf`	DC optimal power flow with unit-decommitment[b]
`runopf_w_res`	optimal power flow with fixed reserve requirements[a]

[a] Uses AC model by default.

[b] Simple wrapper function to set option to use DC model before calling the corresponding general function above.

Table 4: Example Cases

name	description
`caseformat`	help file documenting MATPOWER case format
`case_ieee30`	IEEE 30-bus case
`case24_ieee_rts`	IEEE RTS 24-bus case
`case4gs`	4-bus example case from Grainger & Stevenson
`case6ww`	6-bus example case from Wood & Wollenberg
`case9`	9-bus example case from Chow
`case9Q`	`case9` with reactive power costs
`case14`	IEEE 14-bus case
`case30`	30-bus case, based on IEEE 30-bus case
`case30pwl`	`case30` with piecewise linear costs
`case30Q`	`case30` with reactive power costs
`case39`	39-bus New England case
`case57`	IEEE 57-bus case
`case118`	IEEE 118-bus case
`case300`	IEEE 300-bus case
`case2383wp`	Polish system - winter 1999-2000 peak
`case2736sp`	Polish system - summer 2004 peak
`case2737sop`	Polish system - summer 2004 off-peak
`case2746wop`	Polish system - winter 2003-04 off-peak
`case2746wp`	Polish system - winter 2003-04 evening peak

Table 5: Input/Output Functions

name	description
`cdf2matp`	converts data from IEEE Common Data Format to MATPOWER format
`loadcase`	loads data from a case file or struct into data matrices or a MATPOWER case struct
`mpoption`	
`printpf`	
`savecase`	

Table 6: Power Flow Functions

name	description
`dcpf`	DC power flow implementation
`fdpf`	fast-decoupled power flow implementation
`gausspf`	Gauss-Seidel power flow implementation
`newtonpf`	Newton-method power flow implmementation
`pfsoln`	Computes branch flows, generator reactive power (and real power for slack bus). Updates `bus`, `gen`, `branch` matrices with solved values.

Table 7: OPF Model Object

name	description
`@opf_model/`	
`addconstraints`	
`add_costs`	
`add_vars`	
`build_cost_params`	
`display`	
`get_cost_params`	
`get_idx`	
`get_lin_N`	
`get_mpc`	
`get_nln_N`	
`get_var_N`	
`get`	
`getv`	
`linear_constraints`	
`opf_model`	
`userdata`	

Table 8: OPF and Wrapper Functions

name	description
`dcopf`	
`fmincopf`	
`opf`	
`uopf`	

Table 9: OPF Solver Functions

name	description
`copf_solver`*	
`dcopf_solver`	
`fmincopf_solver`	
`fmincopf6_solver`*	
`lpopf_solver`*	
`pdipm_solver`	
`pdipm6_solver`*	

* Deprecated. Will be removed in a subsequent version.

Table 10: Other OPF Functions

name	description
consfmin	
costfmin	
fun_copf*	
grad_copf*	
hessfmin	
LPconstr*	
LPeqslvr*	
LPrelax*	
LPsetup*	
makeAang	
makeApq	
makeAvl	
makeAy	
opf_args	
totcost	
update_mupq	

* Deprecated. Will be removed in a subsequent version.

Table 11: OPF User Callback Functions

name	description
add_userfcn	
remove_userfcn	
run_userfcn	
toggle_iflims	
toggle_reserves	

Table 12: Power Flow Derivative Functions

name	description
d2AIbr_dV2	
d2ASbr_dV2	
d2Ibr_dV2	
d2Sbr_dV2	
d2Sbus_dV2	
dAbr_dV	
dIbr_dV	
dSbr_dV	
dSbus_dV	

Table 13: NLP, LP & QP Solver Functions

name	description
bpmpd_qp	
mp_lp	
mp_qp	
pdipm	
pdipm6*	
pdipm_qp	
pdipm6_qp*	

* Deprecated. Will be removed in a subsequent version.

Table 14: Matrix Building Functions

name	description
`makeB`	
`makeBdc`	
`makeLODF`	
`makePTDF`	
`makeSbus`	
`makeYbus`	

Table 15: Conversion Functions

name	description
`ext2int`	
`int2ext`	
`get_reorder`	
`set_reorder`	

Table 16: Utility Functions

name	description
bustypes	
compare_case	
define_constants	
fairmax	
hasPQcap	
have_fcn	
idx_area	
idx_brch	
idx_bus	
idx_cost	
idx_gen	
isload	
mpver	
poly2pwl	
polycost	
pqcost	
scale_load	
total_load	

C.1 Automated Test Suite

Table 17: Test Utility Functions

name	description
t/	
t_begin	
t_end	
t_is	
t_ok	
t_run_tests	
t_skip	

Table 18: Test Data

name	description
t/	
soln9_dcopf.mat	
soln9_dcpf.mat	
soln9_opf_ang.mat	
soln9_opf_extras1.mat	
soln9_opf_Plim.mat	
soln9_opf_PQcap.mat	
soln9_opf.mat	
soln9_pf.mat	
t_case_ext.m	
t_case_int.m	
t_case9_opf.m	
t_case9_opfv2.m	
t_case9_pf.m	
t_case9_pfv2.m	
t_case30_userfcns.m	

Table 19: MATPOWER Tests

name	description
t/	
t_auction_case	
t_auction_fmincopf	
t_auction_minopf	
t_auction_pdipm	
t_auction_tspopf_pdipm	
t_ext2int2ext	
t_hasPQcap	
t_hessian	
t_jacobian	
t_loadcase	
t_makeLODF	
t_makePTDF	
t_off2case	
t_opf_constr*	
t_opf_dc_bpmpd	
t_opf_dc_ot	
t_opf_dc_pdipm	
t_opf_dc_scpdipm	
t_opf_fmincon	
t_opf_lp_den*	
t_opf_lp_spf*	
t_opf_lp_spr*	
t_opf_minopf	
t_opf_pdipm	
t_opf_scpdipm	
t_opf_tspopf_pdipm	
t_opf_tspopf_scpdipm	
t_opf_tspopf_tralm	
t_opf_userfcns	
t_pf	
t_runmarket	
t_runopf_w_res	
t_scale_load	
t_total_load	
test_matpower	

* Deprecated. Will be removed in a subsequent version.

Appendix D Extras Directory

`$MATPOWER`/t — Test scripts that can be used to verify that MATPOWER is installed and working correctly (type `test_matpower` at the Matlab prompt).

`$MATPOWER`/extras/smartmarket
Code that implements a "smart market" auction clearing mechanism based on MATPOWER's optimal power flow solver. See Appendix E for details.

`$MATPOWER`/extras/state_estimator
Example state estimation code.

`$MATPOWER`/extras/se — State-estimation code contributed by Rui Bo. Type `test_se` to run an example.

`$MATPOWER`/extras/cpf — Continuation power flow code contributed by Rui Bo. Type `test_cpf` to run an example.

Appendix E Auctions Code

MATPOWER 3 and later includes in the `extras/smartmarket` directory code that implements a "smart market" auction clearing mechanism. The purpose of this code is to take a set of offers to sell and bids to buy and use MATPOWER's optimal power flow to compute the corresponding allocations and prices. It has been used extensively by the authors with the optional MINOPF package [15] in the context of POWERWEB[8] but has not been widely tested in other contexts.

The smart market algorithm consists of the following basic steps:

1. Convert block offers and bids into corresponding generator capacities and costs.
2. Run an optimal power flow with decommitment option (`uopf`) to find generator allocations and nodal prices (λ_P).
3. Convert generator allocations and nodal prices into set of cleared offers and bids.
4. Print results.

For step 1, the offers and bids are supplied as two structs, `offers` and `bids`, each with fields `P` for real power and `Q` for reactive power (optional). Each of these is also a struct with matrix fields `qty` and `prc`, where the element in the i-th row and j-th column of `qty` and `prc` are the quantity and price, respectively of the j-th block of capacity being offered/bid by the i-th generator. These block offers/bids are converted to the equivalent piecewise linear generator costs and generator capacity limits by the `off2case` function. See `help off2case` for more information.

Offer blocks must be in non-decreasing order of price and the offer must correspond to a generator with $0 \leq$ `PMIN` $<$ `PMAX`. A set of price limits can be specified via the `lim` struct, e.g. and offer price cap on real energy would be stored in `lim.P.max_offer`. Capacity offered above this price is considered to be withheld from the auction and is not included in the cost function produced. Bids must be in non-increasing order of price and correspond to a generator with `PMIN` $<$ `PMAX` ≤ 0 (see Section 5.4.2 on page 27). A lower limit can be set for bids in `lim.P.min_bid`. See `help pricelimits` for more information.

The data specified by a MATPOWER case file, with the `gen` and `gencost` matrices modified according to step 1, are then used to run an OPF. A decommitment mechanism is used to shut down generators if doing so results in a smaller overall system cost (see Section 6).

[8] See http://www.pserc.cornell.edu/powerweb/.

In step 3 the OPF solution is used to determine for each offer/bid block, how much was cleared and at what price. These values are returned in `co` and `cb`, which have the same structure as `offers` and `bids`. The `mkt` parameter is a struct used to specify a number of things about the market, including the type of auction to use, type of OPF (AC or DC) to use and the price limits.

There are two basic types of pricing options available through `mkt.auction_type`, discriminative pricing and uniform pricing. The various uniform pricing options are best explained in the context of an unconstrained lossless network. In this context, the allocation is identical to what one would get by creating bid and offer stacks and finding the intersection point. The nodal prices (λ_P) computed by the OPF and returned in `bus(:,LAM_P)` are all equal to the price of the marginal block. This is either the last accepted offer (LAO) or the last accepted bid (LAB), depending which is the marginal block (i.e. the one that is split by intersection of the offer and bid stacks). There is often a gap between the last accepted bid and the last accepted offer. Since any price within this range is acceptable to all buyers and sellers, we end up with a number of options for how to set the price, as listed in Table 20.

Table 20: Auction Types

Auction Type	Name	Description
0	discriminative	The price of each cleared offer (bid) is equal to the offered (bid) price.
1	LAO	Uniform price equal to the last accepted offer.
2	FRO	Uniform price equal to the first rejected offer.
3	LAB	Uniform price equal to the last accepted bid.
4	FRB	Uniform price equal to the first rejected bid.
5	first price	Uniform price equal to the offer/bid price of the marginal unit.
6	second price	Uniform price equal to min(FRO, LAB) if the marginal unit is an offer, or max(FRB, LAO) if it is a bid.
7	split-the-difference	Uniform price equal to the average of the LAO and LAB.
8	dual LAOB	Uniform price for sellers equal to LAO, for buyers equal to LAB.

Generalizing to a network with possible losses and congestion results in nodal prices λ_P which vary according to location. These λ_P values can be used to normalize

all bids and offers to a reference location by adding a locational adjustment. For bids and offers at bus i, the adjustment is $\lambda_P^{\text{ref}} - \lambda_P^i$, where λ_P^{ref} is the nodal price at the reference bus. The desired uniform pricing rule can then be applied to the adjusted offers and bids to get the appropriate uniform price at the reference bus. This uniform price is then adjusted for location by subtracting the locational adjustment. The appropriate locationally adjusted uniform price is then used for all cleared bids and offers.[9]

There are certain circumstances under which the price of a cleared offer determined by the above procedures can be less than the original offer price, such as when a generator is dispatched at its minimum generation limit, or greater than the price cap `lim.P.max_cleared_offer`. For this reason, all cleared offer prices are clipped to be greater than or equal to the offer price but less than or equal to `lim.P.max_cleared_offer`. Likewise, cleared bid prices are less than or equal to the bid price but greater than or equal to `lim.P.min_cleared_bid`.

E.1 Handling Supply Shortfall

In single sided markets, in order to handle situations where the offered capacity is insufficient to meet the demand under all of the other constraints, resulting in an infeasible OPF, we introduce the concept of emergency imports. We model an import as a fixed injection together with an equally sized dispatchable load which is bid in at a high price. Under normal circumstances, the two cancel each other and have no effect on the solution. Under supply shortage situations, the dispatchable load is not fully dispatched, resulting in a net injection at the bus, mimicking an import. When used in conjunction with the LAO pricing rule, the marginal load bid will not set the price if all offered capacity can be used.

[9]Since this code was initially written, we realized it has a problem that has not yet been corrected. While it is true that the adjusted prices resulting from this technique do fall within the "gap" between the last accepted offer and last accepted bid, they do not necessarily correspond to the optimal solution to the OPF. The correct procedure involves multiplying or dividing by locational scale factors as opposed to adding or subtracting the locational adjustment values described here. Multiplying all prices in an OPF solution by a single scale factor is equivalent to simply changing the units used to measure the cost and therefore corresponds to the same optimal solution to the OPF. In light of this, the only auction types we recommend using are 0 and 5, since neither requires use of the locational adjustments described. The former simply uses the bid and offer prices and the latter, the nodal prices directly from the OPF solution.

E.2 Example

Six generators with three blocks of capacity each, offering as follows:

Table 21: Generator Offers

Generator	Block 1 MW @ $/MWh	Block 2 MW @ $/MWh	Block 3 MW @ $/MWh
1	12 @ $20	24 @ $50	24 @ $60
2	12 @ $20	24 @ $40	24 @ $70
3	12 @ $20	24 @ $42	24 @ $80
4	12 @ $20	24 @ $44	24 @ $90
5	12 @ $20	24 @ $46	24 @ $75
6	12 @ $20	24 @ $48	24 @ $60

Fixed load totaling 151.64 MW. Three dispatchable loads, bidding three blocks each as follows:

Table 22: Load Bids

Load	Block 1 MW @ $/MWh	Block 2 MW @ $/MWh	Block 3 MW @ $/MWh
1	10 @ $100	10 @ $70	10 @ $60
2	10 @ $100	10 @ $50	10 @ $20
3	10 @ $100	10 @ $60	10 @ $50

The case file `t/t_auction_case.m`, used for this example, is a modified version of the 30-bus system that has 9 generators, where the last three have negative `PMIN` to model the dispatchable loads.

To solve this case using an AC optimal power flow and a last accepted offer (LAO) pricing rule, we use:

```
mkt.OPF = 'AC';
mkt.auction_type = 1;
```

and set up the problem as follows:

```
offers.P.qty = [ ...
        12 24 24;
```

```
        12 24 24;
        12 24 24;
        12 24 24;
        12 24 24;
        12 24 24 ];

offers.P.prc = [ ...
        20 50 60;
        20 40 70;
        20 42 80;
        20 44 90;
        20 46 75;
        20 48 60 ];

bids.P.qty = [ ...
        10 10 10;
        10 10 10;
        10 10 10 ];

bids.P.prc = [ ...
        100 70 60;
        100 50 20;
        100 60 50 ];

[mpc_out, co, cb, f, dispatch, success, et] = runmarket(mpc, offers, bids, mkt);
```

The resulting cleared offers and bids are:

```
>> co.P.qty

ans =

   12.0000   23.3156         0
   12.0000   24.0000         0
   12.0000   24.0000         0
   12.0000   24.0000         0
   12.0000   24.0000         0
   12.0000   24.0000         0

>> co.P.prc

ans =

   50.0000   50.0000   50.0000
   50.2406   50.2406   50.2406
   50.3368   50.3368   50.3368
```

```
   51.0242   51.0242   51.0242
   52.1697   52.1697   52.1697
   52.9832   52.9832   52.9832

>> cb.P.qty

ans =

   10.0000   10.0000   10.0000
   10.0000         0         0
   10.0000   10.0000         0

>> cb.P.prc

ans =

   51.8207   51.8207   51.8207
   54.0312   54.0312   54.0312
   55.6208   55.6208   55.6208
```

In other words, the sales by generators and purchases by loads are as shown summarized in Tables 23 and Tables 24, respectively.

Table 23: Generator Sales

Generator	Quantity Sold *MW*	Selling Price *$/MWh*
1	35.3	$50.00
2	36.0	$50.24
3	36.0	$50.34
4	36.0	$51.02
5	36.0	$52.17
6	36.0	$52.98

Table 24: Load Purchases

Load	Quantity Bought *MW*	Purchase Price *$/MWh*
1	30.0	$51.82
2	10.0	$54.03
3	20.0	$55.62

References

[1] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," *submitted to IEEE Transactions on Power Systems*

[2] F. Milano, "An open source power system analysis toolbox," *Power Systems, IEEE Transactions on*, vol. 20, no. 3, pp. 1199–1206, Aug. 2005.

[3] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-86, no. 11, pp. 1449–1460, November 1967.

[4] B. Stott and O. Alsaç, "Fast decoupled load flow," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-93, no. 3, pp. 859–869, May 1974.

[5] R. A. M. van Amerongen, "A general-purpose version of the fast decoupled load flow," *Power Systems, IEEE Transactions on*, vol. 4, no. 2, pp. 760–770, May 1989.

[6] A. F. Glimm and G. W. Stagg, "Automatic calculation of load flows," *AIEE Transactions (Power Apparatus and Systems)*, vol. 76, pp. 817–828, October 1957.

[7] A. J. Wood and B. F. Wollenberg, *Power generation, operation, and control*, 2nd ed. New York: J. Wiley & Sons, 1996.

[8] T. Guler, G. Gross, and M. Liu, "Generalized line outage distribution factors," *Power Systems, IEEE Transactions on*, vol. 22, no. 2, pp. 879–881, May 2007.

[9] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER's extensible optimal power flow architecture," *Power and Energy Society General Meeting, 2009 IEEE*, pp. 1–7, July 26–30 2009.

[10] TSPOPF: [Online]. Available: http://www.pserc.cornell.edu/tspopf/

[11] H. Wang, C. E. Murillo-Sánchez, R. D. Zimmerman, and R. J. Thomas, "On computational issues of market-based optimal power flow," *Power Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 1185–1193, August 2007.

[12] *Optimization Toolbox 4 Users's Guide*. The MathWorks, Inc., 2008. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf

[13] BPMPD_MEX: [Online]. Available: http://www.pserc.cornell.edu/bpmpd/

[14] C. Mészáros, "The efficient implementation of interior point methods for linear programming and their applications," Ph.D. dissertation, Eötvös Loránd University of Sciences, 1996.

[15] MINOPF: [Online]. Available: http://www.pserc.cornell.edu/minopf/

[16] B. A. Murtagh and M. A. Saunders, *MINOS 5.5 User's Guide*, Stanford University Systems Optimization Laboratory Technical Report SOL83-20R.