MATPOWER Interior Point Solver
MIPS 1.2.2
User's Manual

Ray D. Zimmerman Hongye Wang

December 16, 2016

Contents

List of Tables

1 Introduction

1.1 Background

MATPOWER **I**nterior **P**oint **S**olver (MIPS) is a package of MATLAB language M-files[1] for solving non-linear programming problems (NLPs) using a primal dual interior point method. The MIPS project page can be found at:

https://github.com/MATPOWER/mips

MIPS is based on code written in C language [1] by Hongye Wang as a graduate student at Cornell University for optimal power flow applications [2, 3]. It was later ported to the MATLAB language by Ray D. Zimmerman of PSERC[2] at Cornell University for use in MATPOWER [4, 5].

Up until version 6 of MATPOWER, MIPS was distributed only as an integrated part of MATPOWER. After the release of MATPOWER 6, MIPS was split out into a separate project, though it is still included with MATPOWER as its default AC optimal power flow solver.

[1] Also compatible with GNU Octave [6].
[2] http://pserc.org/

1.2 License and Terms of Use

The code in MIPS is distributed under the 3-clause BSD license[3] [7]. The full text of the license can be found in the `LICENSE` file at the top level of the distribution or at https://github.com/MATPOWER/mips/blob/master/LICENSE and reads as follows.

[3] Versions 1.0 through 1.1 of MIPS were distributed under version 3.0 of the GNU General Public License (GPL) [8] with an exception added to clarify our intention to allow MIPS to interface with MATLAB as well as any other MATLAB code or MEX-files a user may have installed, regardless of their licensing terms. The full text of the GPL can be found at http://www.gnu.org/licenses/gpl-3.0.txt.

1.3 Citing MIPS

While not required by the terms of the license, we do request that publications derived from the use of MIPS explicitly acknowledge that fact by citing reference [3].

> 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.
> http://dx.doi.org/10.1109/TPWRS.2010.2051168

1.4 MIPS Development

Following the release of MIPS 1.2.2 (with MATPOWER 6.0), the MIPS project moved to an open development paradigm, hosted on the MIPS GitHub project page:

https://github.com/MATPOWER/mips

The MIPS GitHub project hosts the public Git code repository as well as a public issue tracker for handling bug reports, patches, and other issues and contributions. There are separate GitHub hosted repositories and issue trackers for MATPOWER, MOST, MIPS and the testing framework used by all of them, MP-Test, all available from https://github.com/MATPOWER/.

2 Getting Started

2.1 System Requirements

To use MIPS 1.2.2 you will need:

- MATLAB® version 7 (R14) or later[4], or
- GNU Octave version 3.4 or later[5]
- MP-Test, for running the MIPS test suite.[6]

For the hardware requirements, please refer to the system requirements for the version of MATLAB[7] or Octave that you are using.

[4] MATLAB is available from The MathWorks, Inc. (http://www.mathworks.com/). MATLAB is a registered trademark of The MathWorks, Inc.

[5] GNU Octave [6] is free software, available online at http://www.gnu.org/software/octave/. MIPS 1.2.2 may work on earlier versions of Octave, but it has not been tested on versions prior to 3.4.

[6] MP-Test is available at https://github.com/MATPOWER/mptest.

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

In this manual, references to MATLAB usually apply to Octave as well.

2.2 Installation

Installation and use of MIPS requires familiarity with the basic operation of MATLAB or Octave, including setting up your MATLAB path.

Step 1: Clone the repository or download and extract the zip file of the MIPS distribution from the MIPS project page[8] to the location of your choice. The files in the resulting `mips` or `mipsXXX` directory, where `XXX` depends on the version of MIPS, should not need to be modified, so it is recommended that they be kept separate from your own code. We will use <*MIPS*> to denote the path to this directory.

Step 2: Add the following directories to your MATLAB or Octave path:

- <*MIPS*>`/lib` – core MIPS functions
- <*MIPS*>`/lib/t` – test scripts for MIPS

Step 3: At the MATLAB prompt, type `test_mips` to run the test suite and verify that MIPS is properly installed and functioning.[9] The result should resemble the following:

```
>> test_mips
t_mplinsolve....ok
t_mips..........ok
t_qps_mips......ok
All tests successful (136 of 136)
Elapsed time 0.11 seconds.
```

2.3 Documentation

There are two primary sources of documentation for MIPS. The first is this manual, which gives an overview of the capabilities and structure of MIPS and describes the formulations behind the code. It can be found in your MIPS distribution at <*MIPS*>`/docs/MIPS-manual.pdf` and the latest version is always available at: `https://github.com/MATPOWER/mips/blob/master/MIPS-manual.pdf`.

[8] `https://github.com/MATPOWER/mips`

[9] The tests require a functioning installation of MP-Test.

And second is the built-in `help` command. As with the built-in functions and toolbox routines in MATLAB and Octave, you can type `help` followed by the name of a command or M-file to get help on that particular function. All of the M-files in MIPS have such documentation and this should be considered the main reference for the calling options for each function. See Appendix A for a list of MIPS functions.

3 MIPS – MATPOWER Interior Point Solver

MIPS, that is, the **MATPOWER** **I**nterior **P**oint **S**olver, is a primal-dual interior point solver implemented in pure MATLAB code, derived from the MEX implementation of the algorithms included in TSPOPF [1] and described in [2, 3].

This solver has application to general nonlinear optimization problems of the following form:

$$\min_x f(x) \tag{3.1}$$

subject to

$$g(x) = 0 \tag{3.2}$$
$$h(x) \leq 0 \tag{3.3}$$
$$l \leq Ax \leq u \tag{3.4}$$
$$x_{\min} \leq x \leq x_{\max} \tag{3.5}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$, $g\colon \mathbb{R}^n \to \mathbb{R}^m$ and $h\colon \mathbb{R}^n \to \mathbb{R}^p$.

The solver is implemented by the `mips` function, which can be called as follows,

```
[x, f, exitflag, output, lambda] = ...
    mips(f_fcn, x0, A, l, u, xmin, xmax, gh_fcn, hess_fcn, opt);
```

where the input and output arguments are described in Tables 3-1 and 3-2, respectively. Alternatively, the input arguments can be packaged as fields in a `problem` struct and passed in as a single argument, where all fields except `f_fcn` and `x0` are optional.

```
[x, f, exitflag, output, lambda] = mips(problem);
```

The calling syntax is nearly identical to that used by `fmincon` from MATLAB's Optimization Toolbox. The primary difference is that the linear constraints are specified in terms of a single doubly-bounded linear function ($l \leq Ax \leq u$) as opposed to separate equality constrained ($A_{eq}x = b_{eq}$) and upper bounded ($Ax \leq b$) functions. Internally, equality constraints are handled explicitly and determined at run-time based on the values of l and u.

The user-defined functions for evaluating the objective function, constraints and Hessian are identical to those required by `fmincon`, with one exception described below for the Hessian evaluation function. Specifically, `f_fcn` should return `f` as the scalar objective function value $f(x)$, `df` as an $n \times 1$ vector equal to ∇f and, unless

Table 3-1: Input Arguments for `mips`[†]

name	description
`f_fcn`	Handle to a function that evaluates the objective function, its gradients and Hessian[‡] for a given value of x. Calling syntax for this function: `[f, df, d2f] = f_fcn(x)`
`x0`	Starting value of optimization vector x.
`A`, `l`, `u`	Define the optional linear constraints $l \leq Ax \leq u$. Default values for the elements of `l` and `u` are `-Inf` and `Inf`, respectively.
`xmin`, `xmax`	Optional lower and upper bounds on the x variables, defaults are `-Inf` and `Inf`, respectively.
`gh_fcn`	Handle to function that evaluates the optional nonlinear constraints and their gradients for a given value of x. Calling syntax for this function is: `[h, g, dh, dg] = gh_fcn(x)`
`hess_fcn`	Handle to function that computes the Hessian[‡]of the Lagrangian for given values of x, λ and μ, where λ and μ are the multipliers on the equality and inequality constraints, g and h, respectively. The calling syntax for this function is: `Lxx = hess_fcn(x, lam, cost_mult)`, where $\lambda =$ `lam.eqnonlin`, $\mu =$ `lam.ineqnonlin` and `cost_mult` is a parameter used to scale the objective function
`opt`	Optional options structure with fields, all of which are also optional, described in Table 3-3.
`problem`	Alternative, single argument input struct with fields corresponding to arguments above.

[†] All inputs are optional except `f_fcn` and `x0`.
[‡] If `gh_fcn` is provided then `hess_fcn` is also required. Specifically, if there are nonlinear constraints, the Hessian information must provided by the `hess_fcn` function and it need not be computed in `f_fcn`.

`gh_fcn` is provided and the Hessian is computed by `hess_fcn`, `d2f` as an $n \times n$ matrix equal to the Hessian $\frac{\partial^2 f}{\partial x^2}$. Similarly, the constraint evaluation function `gh_fcn` must return the $m \times 1$ vector of nonlinear equality constraint violations $g(x)$, the $p \times 1$ vector of nonlinear inequality constraint violations $h(x)$ along with their gradients in `dg` and `dh`. Here `dg` is an $n \times m$ matrix whose j^{th} column is ∇g_j and `dh` is $n \times p$, with j^{th} column equal to ∇h_j. Finally, for cases with nonlinear constraints, `hess_fcn` returns the $n \times n$ Hessian $\frac{\partial^2 \mathcal{L}}{\partial x^2}$ of the Lagrangian function

$$\mathcal{L}(x, \lambda, \mu, \sigma) = \sigma f(x) + \lambda^{\mathsf{T}} g(x) + \mu^{\mathsf{T}} h(x) \tag{3.6}$$

for given values of the multipliers λ and μ, where σ is the `cost_mult` scale factor for the objective function. Unlike `fmincon`, `mips` passes this scale factor to the Hessian evaluation function in the 3$^{\text{rd}}$ argument.

The use of `nargout` in `f_fcn` and `gh_fcn` is recommended so that the gradients and Hessian are only computed when required.

Table 3-2: Output Arguments for `mips`

name	description
`x`	solution vector
`f`	final objective function value
`exitflag`	exit flag 1 – first order optimality conditions satisfied 0 – maximum number of iterations reached -1 – numerically failed
`output`	output struct with fields
	`iterations` number of iterations performed
	`hist` struct array with trajectories of the following: `feascond`, `gradcond`, `compcond`, `costcond`, `gamma`, `stepsize`, `obj`, `alphap`, `alphad`
	`message` exit message
`lambda`	struct containing the Langrange and Kuhn-Tucker multipliers on the constraints, with fields:
	`eqnonlin` nonlinear equality constraints
	`ineqnonlin` nonlinear inequality constraints
	`mu_l` lower (left-hand) limit on linear constraints
	`mu_u` upper (right-hand) limit on linear constraints
	`lower` lower bound on optimization variables
	`upper` upper bound on optimization variables

3.1 Example 1

The following code, included as `mips_example1.m` in <*MIPS*>`lib/t`, shows a simple example of using `mips` to solve a 2-dimensional unconstrained optimization of Rosenbrock's "banana" function[10]

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \tag{3.7}$$

First, create a MATLAB function that will evaluate the objective function, its gradients and Hessian, for a given value of x. In this case, the coefficient of the first term is defined as a paramter `a`.

[10] http://en.wikipedia.org/wiki/Rosenbrock_function

Table 3-3: Options for `mips`[†]

name	default	description
`opt.verbose`	0	controls level of progress output displayed 0 – print no progress info 1 – print a little progress info 2 – print a lot of progress info 3 – print all progress info
`opt.feastol`	10^{-6}	termination tolerance for feasibility condition
`opt.gradtol`	10^{-6}	termination tolerance for gradient condition
`opt.comptol`	10^{-6}	termination tolerance for complementarity condition
`opt.costtol`	10^{-6}	termination tolerance for cost condition
`opt.max_it`	150	maximum number of iterations
`opt.step_control`	0	set to 1 to enable step-size control
`opt.sc.red_it`	20	max number of step-size reductions if step-control is on
`opt.cost_mult`	1	cost multiplier used to scale the objective function for improved conditioning. Note: This value is also passed as the 3$^{\text{rd}}$ argument to the Hessian evaluation function so that it can appropriately scale the objective function term in the Hessian of the Lagrangian.
`opt.xi`	0.99995	ξ constant used in α updates in (3.46) and (3.47)
`opt.sigma`	0.1	centering parameter σ used in γ update in (3.52)
`opt.z0`	1	used to initialize elements of slack variable Z
`opt.alpha_min`	10^{-8}	algorithm returns "Numerically Failed" if the α_p or α_d from (3.46) and (3.47) become smaller than this value
`opt.rho_min`	0.95	lower bound on ρ_t corresponding to $1-\eta$ in Fig. 5 in [2]
`opt.rho_max`	1.05	upper bound on ρ_t corresponding to $1+\eta$ in Fig. 5 in [2]
`opt.mu_threshold`	10^{-5}	Kuhn-Tucker multipliers smaller than this value for non-binding constraints are forced to zero
`opt.max_stepsize`	10^{10}	algorithm returns "Numerically Failed" if the 2-norm of the Newton step $\begin{bmatrix} \Delta X \\ \Delta\lambda \end{bmatrix}$ from (3.45) exceeds this value

```
function [f, df, d2f] = banana(x, a)
f = a*(x(2)-x(1)^2)^2+(1-x(1))^2;
if nargout > 1          %% gradient is required
    df = [  4*a*(x(1)^3 - x(1)*x(2)) + 2*x(1)-2;
            2*a*(x(2) - x(1)^2)                    ];
    if nargout > 2      %% Hessian is required
        d2f = 4*a*[ 3*x(1)^2 - x(2) + 1/(2*a),  -x(1);
                    -x(1)                       1/2 ];
    end
end
```

Then, create a handle to the function, defining the value of the paramter `a` to be 100, set up the starting value of x, and call the `mips` function to solve it.

```
>> f_fcn = @(x)banana(x, 100);
>> x0 = [-1.9; 2];
>> [x, f] = mips(f_fcn, x0)

x =

     1
     1


f =

     0
```

3.2 Example 2

The second example[11] solves the following 3-dimensional constrained optimization, printing the details of the solver's progress:

$$\min_x f(x) = -x_1x_2 - x_2x_3 \tag{3.8}$$

subject to

$$x_1^2 - x_2^2 + x_3^2 - 2 \leq 0 \tag{3.9}$$
$$x_1^2 + x_2^2 + x_3^2 - 10 \leq 0. \tag{3.10}$$

First, create a MATLAB function to evaluate the objective function and its gradients,[12]

[11] From http://en.wikipedia.org/wiki/Nonlinear_programming#3-dimensional_example.

[12] Since the problem has nonlinear constraints and the Hessian is provided by `hess_fcn`, this function will never be called with three output arguments, so the code to compute `d2f` is actually not necessary.

```
function [f, df, d2f] = f2(x)
f = -x(1)*x(2) - x(2)*x(3);
if nargout > 1           %% gradient is required
    df = -[x(2); x(1)+x(3); x(2)];
    if nargout > 2       %% Hessian is required
        d2f = -[0 1 0; 1 0 1; 0 1 0];   %% actually not used since
    end                                 %% 'hess_fcn' is provided
end
```

one to evaluate the constraints, in this case inequalities only, and their gradients,

```
function [h, g, dh, dg] = gh2(x)
h = [ 1 -1 1; 1 1 1] * x.^2 + [-2; -10];
dh = 2 * [x(1) x(1); -x(2) x(2); x(3) x(3)];
g = []; dg = [];
```

and another to evaluate the Hessian of the Lagrangian.

```
function Lxx = hess2(x, lam, cost_mult)
if nargin < 3, cost_mult = 1; end   %% allows to be used with 'fmincon'
mu = lam.ineqnonlin;
Lxx = cost_mult * [0 -1 0; -1 0 -1; 0 -1 0] + ...
        [2*[1 1]*mu 0 0; 0 2*[-1 1]*mu 0; 0 0 2*[1 1]*mu];
```

Then create a `problem` struct with handles to these functions, a starting value for x and an option to print the solver's progress. Finally, pass this struct to `mips` to solve the problem and print some of the return values to get the output below.

```
function mips_example2
problem = struct( ...
    'f_fcn',    @(x)f2(x), ...
    'gh_fcn',   @(x)gh2(x), ...
    'hess_fcn', @(x, lam, cost_mult)hess2(x, lam, cost_mult), ...
    'x0',       [1; 1; 0], ...
    'opt',      struct('verbose', 2) ...
);
[x, f, exitflag, output, lambda] = mips(problem);
fprintf('\nf = %g   exitflag = %d\n', f, exitflag);
fprintf('\nx = \n');
fprintf('   %g\n', x);
fprintf('\nlambda.ineqnonlin =\n');
fprintf('   %g\n', lambda.ineqnonlin);
```

```
>> mips_example2
MATPOWER Interior Point Solver -- MIPS, Version 1.2.2, 16-Dec-2016
 (using built-in linear solver)
 it    objective   step size   feascond     gradcond     compcond     costcond
----  ------------ --------- ------------ ------------ ------------ ------------
  0            -1                    0          1.5            5            0
  1    -5.3250167     1.6875           0     0.894235     0.850653      2.16251
  2    -7.4708991    0.97413    0.129183   0.00936418     0.117278     0.339269
  3    -7.0553031    0.10406           0   0.00174933    0.0196518    0.0490616
  4    -7.0686267   0.034574           0   0.00041301    0.0030084   0.00165402
  5    -7.0706104  0.0065191           0  1.53531e-05  0.000337971  0.000245844
  6    -7.0710134 0.00062152           0  1.22094e-07  3.41308e-05  4.99387e-05
  7    -7.0710623 5.7217e-05           0  9.84879e-10  3.41587e-06  6.05875e-06
  8    -7.0710673 5.6761e-06           0  9.73527e-12  3.41615e-07  6.15483e-07
Converged!

f = -7.07107   exitflag = 1

x =
   1.58114
   2.23607
   1.58114

lambda.ineqnonlin =
   0
   0.707107
```

This example can be found in `mips_example2.m`. More example problems for `mips` can be found in `t_mips.m`, both in <*MIPS*>`lib/t`.

3.3 Quadratic Programming Solver

A convenience wrapper function called `qps_mips` is provided to make it trivial to set up and solve linear programming (LP) and quadratic programming (QP) problems of the following form:

$$\min_x \frac{1}{2} x^\mathsf{T} H x + c^\mathsf{T} x \tag{3.11}$$

subject to

$$l \leq Ax \leq u \tag{3.12}$$

$$x_{\min} \leq x \leq x_{\max}. \tag{3.13}$$

Instead of a function handle, the objective function is specified in terms of the paramters H and c of quadratic cost coefficients. Internally, `qps_mips` passes `mips` the handle of a function that uses these paramters to evaluate the objective function, gradients and Hessian.

The calling syntax for `qps_mips` is similar to that used by `quadprog` from the MATLAB Optimization Toolbox.

```
[x, f, exitflag, output, lambda] = qps_mips(H, c, A, l, u, xmin, xmax, x0, opt);
```

Alternatively, the input arguments can be packaged as fields in a `problem` struct and passed in as a single argument, where all fields except `H`, `c`, `A` and `l` are optional.

```
[x, f, exitflag, output, lambda] = qps_mips(problem);
```

Aside from `H` and `c`, all input and output arguments correspond exactly to the same arguments for `mips` as described in Tables 3-1 and 3-2.

As with `mips` and `fmincon`, the primary difference between the calling syntax for `qps_mips` and `quadprog` is that the linear constraints are specified in terms of a single doubly-bounded linear function ($l \leq Ax \leq u$) as opposed to separate equality constrained ($A_{eq}x = b_{eq}$) and upper bounded ($Ax \leq b$) functions.

MIPS also includes another wrapper function `qps_matpower` that provides a consistent interface for all of the QP and LP solvers it has available. This interface is identical to that used by `qps_mips` with the exception of the structure of the `opt` input argument. The solver is chosen according to the value of `opt.alg`. See the help for `qps_matpower` for details.

Several examples of using `qps_matpower` to solve LP and QP problems can be found in `t_qps_matpower.m`.

3.4 Primal-Dual Interior Point Algorithm

This section provides some details on the primal-dual interior point algorithm used by MIPS and described in [2, 3].

3.4.1 Notation

For a scalar function $f\colon \mathbb{R}^n \to \mathbb{R}$ of a real vector $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^{\mathsf{T}}$, we use the following notation for the first derivatives (transpose of the gradient):

$$f_X = \frac{\partial f}{\partial X} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}. \tag{3.14}$$

The matrix of second partial derivatives, the Hessian of f, is:

$$f_{XX} = \frac{\partial^2 f}{\partial X^2} = \frac{\partial}{\partial X}\left(\frac{\partial f}{\partial X}\right)^{\mathsf{T}} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{3.15}$$

For a vector function $F\colon \mathbb{R}^n \to \mathbb{R}^m$ of a vector X, where

$$F(X) = \begin{bmatrix} f_1(X) & f_2(X) & \cdots & f_m(X) \end{bmatrix}^{\mathsf{T}} \tag{3.16}$$

the first derivatives form the Jacobian matrix, where row i is the transpose of the gradient of f_i

$$F_X = \frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{3.17}$$

In these derivations, the full 3-dimensional set of second partial derivatives of F will not be computed. Instead a matrix of partial derivatives will be formed by computing the Jacobian of the vector function obtained by multiplying the transpose of the Jacobian of F by a vector λ, using the following notation

$$F_{XX}(\lambda) = \frac{\partial}{\partial X}\left({F_X}^{\mathsf{T}}\lambda\right). \tag{3.18}$$

Please note also that $[A]$ is used to denote a diagonal matrix with vector A on the diagonal and e is a vector of all ones.

3.4.2 Problem Formulation and Lagrangian

The primal-dual interior point method used by MIPS solves a problem of the form:

$$\min_X f(X) \tag{3.19}$$

subject to

$$G(X) = 0 \tag{3.20}$$

$$H(X) \leq 0 \tag{3.21}$$

where the linear constraints and variable bounds from (3.4) and (3.5) have been incorporated into $G(X)$ and $H(X)$. The approach taken involves converting the n_i

inequality constraints into equality constraints using a barrier function and vector of positive slack variables Z.

$$\min_X \left[f(X) - \gamma \sum_{m=1}^{n_i} \ln(Z_m) \right] \tag{3.22}$$

subject to

$$G(X) = 0 \tag{3.23}$$
$$H(X) + Z = 0 \tag{3.24}$$
$$Z > 0 \tag{3.25}$$

As the parameter of perturbation γ approaches zero, the solution to this problem approaches that of the original problem.

For a given value of γ, the Lagrangian for this equality constrained problem is

$$\mathcal{L}^\gamma(X, Z, \lambda, \mu) = f(X) + \lambda^\mathsf{T} G(X) + \mu^\mathsf{T}(H(X) + Z) - \gamma \sum_{m=1}^{n_i} \ln(Z_m). \tag{3.26}$$

Taking the partial derivatives with respect to each of the variables yields:

$$\mathcal{L}^\gamma_X(X, Z, \lambda, \mu) = f_X + \lambda^\mathsf{T} G_X + \mu^\mathsf{T} H_X \tag{3.27}$$
$$\mathcal{L}^\gamma_Z(X, Z, \lambda, \mu) = \mu^\mathsf{T} - \gamma e^\mathsf{T} [Z]^{-1} \tag{3.28}$$
$$\mathcal{L}^\gamma_\lambda(X, Z, \lambda, \mu) = G^\mathsf{T}(X) \tag{3.29}$$
$$\mathcal{L}^\gamma_\mu(X, Z, \lambda, \mu) = H^\mathsf{T}(X) + Z^\mathsf{T}. \tag{3.30}$$

And the Hessian of the Lagrangian with respect to X is given by

$$\mathcal{L}^\gamma_{XX}(X, Z, \lambda, \mu) = f_{XX} + G_{XX}(\lambda) + H_{XX}(\mu). \tag{3.31}$$

3.4.3 First Order Optimality Conditions

The first order optimality (Karush-Kuhn-Tucker) conditions for this problem are satisfied when the partial derivatives of the Lagrangian above are all set to zero:

$$F(X, Z, \lambda, \mu) = 0 \tag{3.32}$$
$$Z > 0 \tag{3.33}$$
$$\mu > 0 \tag{3.34}$$

where

$$F(X,Z,\lambda,\mu) = \begin{bmatrix} \mathcal{L}_X^{\gamma\,\mathsf{T}} \\ [\mu]\,Z - \gamma e \\ G(X) \\ H(X) + Z \end{bmatrix} = \begin{bmatrix} f_X{}^{\mathsf{T}} + G_X{}^{\mathsf{T}}\lambda + H_X{}^{\mathsf{T}}\mu \\ [\mu]\,Z - \gamma e \\ G(X) \\ H(X) + Z \end{bmatrix}. \tag{3.35}$$

3.4.4 Newton Step

The first order optimality conditions are solved using Newton's method. The Newton update step can be written as follows:

$$\begin{bmatrix} F_X & F_Z & F_\lambda & F_\mu \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Z \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = -F(X,Z,\lambda,\mu) \tag{3.36}$$

$$\begin{bmatrix} \mathcal{L}_{XX}^{\gamma} & 0 & G_X{}^{\mathsf{T}} & H_X{}^{\mathsf{T}} \\ 0 & [\mu] & 0 & [Z] \\ G_X & 0 & 0 & 0 \\ H_X & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Z \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = - \begin{bmatrix} \mathcal{L}_X^{\gamma\,\mathsf{T}} \\ [\mu]\,Z - \gamma e \\ G(X) \\ H(X) + Z \end{bmatrix}. \tag{3.37}$$

This set of equations can be simplified and reduced to a smaller set of equations by solving explicitly for $\Delta\mu$ in terms of ΔZ and for ΔZ in terms of ΔX. Taking the 2^{nd} row of (3.37) and solving for $\Delta\mu$ we get

$$\begin{aligned} [\mu]\,\Delta Z + [Z]\,\Delta\mu &= -[\mu]\,Z + \gamma e \\ [Z]\,\Delta\mu &= -[Z]\,\mu + \gamma e - [\mu]\,\Delta Z \\ \Delta\mu &= -\mu + [Z]^{-1}\,(\gamma e - [\mu]\,\Delta Z). \end{aligned} \tag{3.38}$$

Solving the 4^{th} row of (3.37) for ΔZ yields

$$\begin{aligned} H_X\Delta X + \Delta Z &= -H(X) - Z \\ \Delta Z &= -H(X) - Z - H_X\Delta X. \end{aligned} \tag{3.39}$$

Then, substituting (3.38) and (3.39) into the 1st row of (3.37) results in

$$\begin{aligned}
&\mathcal{L}_{XX}^{\gamma}\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda + {H_X}^{\mathsf{T}}\Delta\mu && = -{\mathcal{L}_X^{\gamma}}^{\mathsf{T}} \\
&\mathcal{L}_{XX}^{\gamma}\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda + {H_X}^{\mathsf{T}}(-\mu + [Z]^{-1}(\gamma e - [\mu]\,\Delta Z)) && = -{\mathcal{L}_X^{\gamma}}^{\mathsf{T}} \\
&\mathcal{L}_{XX}^{\gamma}\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda \\
&+ {H_X}^{\mathsf{T}}(-\mu + [Z]^{-1}(\gamma e - [\mu](-H(X) - Z - H_X\Delta X))) && = -{\mathcal{L}_X^{\gamma}}^{\mathsf{T}} \\
&\mathcal{L}_{XX}^{\gamma}\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda - {H_X}^{\mathsf{T}}\mu + {H_X}^{\mathsf{T}}[Z]^{-1}\gamma e \\
&+ {H_X}^{\mathsf{T}}[Z]^{-1}[\mu]\,H(X) + {H_X}^{\mathsf{T}}[Z]^{-1}[Z]\,\mu + {H_X}^{\mathsf{T}}[Z]^{-1}[\mu]\,H_X\Delta X && = -{\mathcal{L}_X^{\gamma}}^{\mathsf{T}} \\
&(\mathcal{L}_{XX}^{\gamma} + {H_X}^{\mathsf{T}}[Z]^{-1}[\mu]\,H_X)\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda \\
&+ {H_X}^{\mathsf{T}}[Z]^{-1}(\gamma e + [\mu]\,H(X)) && = -{\mathcal{L}_X^{\gamma}}^{\mathsf{T}} \\
&M\Delta X + {G_X}^{\mathsf{T}}\Delta\lambda && = -N \qquad (3.40)
\end{aligned}$$

where

$$\begin{aligned}
M &\equiv \mathcal{L}_{XX}^{\gamma} + {H_X}^{\mathsf{T}}[Z]^{-1}[\mu]\,H_X && (3.41)\\
&= f_{XX} + G_{XX}(\lambda) + H_{XX}(\mu) + {H_X}^{\mathsf{T}}[Z]^{-1}[\mu]\,H_X && (3.42)
\end{aligned}$$

and

$$\begin{aligned}
N &\equiv {\mathcal{L}_X^{\gamma}}^{\mathsf{T}} + {H_X}^{\mathsf{T}}[Z]^{-1}(\gamma e + [\mu]\,H(X)) && (3.43)\\
&= {f_X}^{\mathsf{T}} + {G_X}^{\mathsf{T}}\lambda + {H_X}^{\mathsf{T}}\mu + {H_X}^{\mathsf{T}}[Z]^{-1}(\gamma e + [\mu]\,H(X)). && (3.44)
\end{aligned}$$

Combining (3.40) and the 3rd row of (3.37) results in a system of equations of reduced size:

$$\begin{bmatrix} M & {G_X}^{\mathsf{T}} \\ G_X & 0 \end{bmatrix}\begin{bmatrix} \Delta X \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -N \\ -G(X) \end{bmatrix}. \qquad (3.45)$$

The Newton update can then be computed in the following 3 steps:

1. Compute ΔX and $\Delta\lambda$ from (3.45).
2. Compute ΔZ from (3.39).
3. Compute $\Delta\mu$ from (3.38).

In order to maintain strict feasibility of the trial solution, the algorithm truncates the Newton step by scaling the primal and dual variables by α_p and α_d, respectively,

where these scale factors are computed as follows:

$$\alpha_p = \min\left(\xi \min_{\Delta Z_m < 0}\left(-\frac{Z_m}{\Delta Z_m}\right), 1\right) \tag{3.46}$$

$$\alpha_d = \min\left(\xi \min_{\Delta \mu_m < 0}\left(-\frac{\mu_m}{\Delta \mu_m}\right), 1\right) \tag{3.47}$$

resulting in the variable updates below.

$$X \leftarrow X + \alpha_p \Delta X \tag{3.48}$$

$$Z \leftarrow Z + \alpha_p \Delta Z \tag{3.49}$$

$$\lambda \leftarrow \lambda + \alpha_d \Delta \lambda \tag{3.50}$$

$$\mu \leftarrow \mu + \alpha_d \Delta \mu \tag{3.51}$$

The parameter ξ is a constant scalar with a value slightly less than one. In MIPS, ξ is set to 0.99995.

In this method, during the Newton-like iterations, the perturbation parameter γ must converge to zero in order to satisfy the first order optimality conditions of the original problem. MIPS uses the following rule to update γ at each iteration, after updating Z and μ:

$$\gamma \leftarrow \sigma \frac{Z^{\mathsf{T}} \mu}{n_i} \tag{3.52}$$

where σ is a scalar constant between 0 and 1. In MIPS, σ is set to 0.1.

Appendix A MIPS Files and Functions

This appendix lists all of the files and functions that MIPS provides. In most cases, the function is found in a MATLAB M-file in the `lib` directory of the distribution, where the `.m` extension is omitted from this listing. For more information on each, at the MATLAB prompt, simply type `help` followed by the name of the function. For documentation and other files, the filename extensions are included.

Table A-1: MIPS Files and Functions

name	description
`AUTHORS`	list of authors and contributors
`CHANGES`	MIPS change history
`CONTRIBUTING.md`	notes on how to contribute to the MIPS project
`LICENSE`	MIPS license (3-clause BSD license)
`README.md`	basic introduction to MIPS
`docs/`	
`MIPS-manual.pdf`	MIPS User's Manual
`MIPS-manual.tex`	LaTeX source for MIPS User's Manual
`lib/`	
`mips`	**M**ATPOWER **I**nterior **P**oint **S**olver – primal/dual interior point solver for NLP
`mipsver`	prints version information for MIPS
`mplinsolve`	common linear system solver interface, used by MIPS
`qps_mips`	common QP/LP solver interface to MIPS-based solver
`t/`	
`mips_example1`	implements example 1 from MIPS User's Manual
`mips_example2`	implements example 2 from MIPS User's Manual
`test_mips`	runs full MIPS test suite
`t_mips`	runs tests for MIPS NLP solver
`t_mplinsolve`	tests for `mplinsolve`
`t_qps_mips`	runs tests for `qps_mips`

Appendix B PARDISO – Parallel Sparse Direct and Multi-Recursive Iterative Linear Solvers

The PARDISO package is a thread-safe, high-performance, robust, memory efficient and easy to use software for solving large sparse symmetric and non-symmetric linear systems of equations on shared-memory and distributed-memory multiprocessor systems [9, 10]. More information is available at `http://www.pardiso-project.org`.

When the MATLAB interface to PARDISO is installed, PARDISO's solvers can be used to replace the built-in \ operator for solving for the Newton update step in MIPS by setting the `linsolver` option equal to `'PARDISO'`. The `mplinsolve` function can also be called directly to solve $Ax = b$ problems via PARDISO or the built-in solver, depending on the arguments supplied. This interface also gives access to the full range of PARDISO's options. For details, see `help mplinsolve` and the PARDISO User's Manual at `http://www.pardiso-project.org/manual/manual.pdf`.

Appendix C Release History

The full release history can be found in `CHANGES.md` or online at https://github.com/MATPOWER/mips/blob/master/CHANGES.md.

C.1 Version 1.2.2 – released Dec 16, 2016

Documentation found in the MIPS User's Manual[13] or in Appendix A of the MATPOWER 6.0 User's Manual, available online.[14]

New Open Development Model

- MIPS development has moved to GitHub! The code repository is now publicly available to clone and submit pull requests.[15]
- Public issue tracker for reporting bugs, submitting patches, etc.[16]

Other Changes

- Renamed from MATLAB Interior Point Solver to **M**ATPOWER **I**nterior **P**oint **S**olver.
- Remove dependence of `t_mpsolve()` on presence of `have_fcn()` (from MATPOWER) to detect PARDISO installation. .

C.2 Version 1.2.1 – released Jun 1, 2016

Documentation found in Appendix A of the MATPOWER 6.0b1 User's Manual, available online.[17]

Bug Fixed

- Fixed issue where default value of `'feastol'` option was not being set correctly in `mips()` when called directly (or via `qps_mips()`) with `'feastol' = 0`.

[13] http://www.pserc.cornell.edu/matpower/docs/MIPS-manual-1.2.2.pdf
[14] http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-6.0.pdf
[15] https://github.com/MATPOWER/mips
[16] https://github.com/MATPOWER/mips/issues
[17] http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-6.0b1.pdf

C.3 Version 1.2 – released Mar 20, 2015

Documentation found in Appendix A of the MATPOWER 5.1 User's Manual, available online.[18]

New License

- Switched to the more permissive 3-clause BSD license from the previously used GNU General Public License (GPL) v3.0.

New Documentation

- Added an online function reference to the MATPOWER website at `http://www.pserc.cornell.edu/matpower/docs/ref/`.

New Features

- Added support for using PARDISO (`http://www.pardiso-project.org/`) as linear solver for computing interior-point update steps in MIPS, resulting in dramatic improvements in computation time and memory use for very large-scale problems.
- New functions:
  - `mplinsolve()` provides unified interface for linear system solvers, including PARDISO and built-in backslash operator

C.4 Version 1.1 – released Dec 17, 2014

Documentation found in Appendix A of the MATPOWER 5.0 User's Manual, available online.[19]

New Features

- Many new user-settable options.

[18] `http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-5.1.pdf`
[19] `http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-5.0.pdf`

Incompatible Changes

- The name of the `mips()` option used to specify the maximum number of step-size reductions with `step_control` on was changed from `max_red` to `sc.red_it` for consistency with other MATPOWER options.

C.5 Version 1.0.2 – released Nov 5, 2013

Documentation found in Appendix A of the MATPOWER 4.0 User's Manual, available online.[20]

Bug Fixed

- Fixed a bug in MIPS where a near-singular matrix could produce an extremely large Newton step, resulting in incorrectly satisfying the relative feasibility criterion for successful termination.

C.6 Version 1.0.1 – released Apr 30, 2012

Documentation found in Appendix A of the MATPOWER 4.0 User's Manual, available online.[21]

Bug Fixed

- Fixed fatal bug in MIPS for unconstrained, scalar problems. *Thanks to Han Na Gwon.*

C.7 Version 1.0 – released Feb 7, 2011

Documentation found in Appendix A of the MATPOWER 4.0 User's Manual, available online.[22]

Changes

- Licensed under the GNU General Public License (GPL).
- Added compatibility with GNU Octave, a free, open-source MATLAB clone.

[20] http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-4.0.pdf
[21] http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-4.0.pdf
[22] http://www.pserc.cornell.edu/matpower/docs/MATPOWER-manual-4.0.pdf

- MIPS (**M**ATPOWER **I**nterior **P**oint **S**olver), a new a pure-MATLAB implementation of the primal-dual interior point methods from the optional package TSPOPF.

References

[1] TSPOPF. [Online]. Available: http://www.pserc.cornell.edu/tspopf/. 1.1, 3

[2] 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. http://dx.doi.org/10.1109/TPWRS.2010.2051168 1.1, 3, 3-3, 3.4

[3] H. Wang, *On the Computation and Application of Multi-period Security-constrained Optimal Power Flow for Real-time Electricity Market Operations*, Ph.D. thesis, Electrical and Computer Engineering, Cornell University, May 2007. 1.1, 1.3, 3, 3.4

[4] R. D. Zimmerman and C. Murillo-Sánchez. MATPOWER User's Manual, [Online]. Available: http://www.pserc.cornell.edu/matpower/ 1.1

[5] 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," *Power Systems, IEEE Transactions on*, vol. 26, no. 1, pp. 12–19, Feb. 2011. http://dx.doi.org/10.1109/TPWRS.2010.2051168 1.1

[6] John W. Eaton, David Bateman, Søren Hauberg, Rik Wehbring (2015). *GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations.* Available: http://www.gnu.org/software/octave/doc/interpreter/. 1, 5

[7] The BSD 3-Clause License. [Online]. Available: http://opensource.org/licenses/BSD-3-Clause. 1.2

[8] GNU General Public License. [Online]. Available: http://www.gnu.org/licenses/. 3

[9] O. Shenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004. B

[10] A. Kuzmin, M. Luisier and O. Shenk, "Fast methods for computing selected elements of the Greens function in massively parallel nanoelectronic device simulations," in F. Wolf, B. Mohr and D. Mey, editors, *Euro-Par 2013 Parallel*

Processing, Vol. 8097, *Lecture Notes in Computer Science*, pp. 533-544, Springer Berlin Heidelberg, 2013. B