Optimization Methods and Software, Vol.11&12, pp.385-396, 1999

User's Guide to LIPSOL
Linear-programming Interior Point Solvers v0.4 *

Yin Zhang
Department of Computational and Applied Mathematics
Rice University, Houston, Texas 77005, U.S.A.

1 Introduction

1.1 What is LIPSOL?

LIPSOL stands for Linear programming Interior-Point SOLvers. It is a free, Matlab-based software package for solving linear programs by interior-Point methods. It requires Matlab version 4.0 or later to run. The current release of LIPSOL is for 32-bit UNIX platforms.

LIPSOL is designed to solve relatively large problems. It utilizes Matlab's sparse-matrix data-structure and Application Program Interface facility, and at the same time takes advantages of existing, efficient Fortran codes for solving large, sparse, symmetric positive definite linear systems. Specifically, LIPSOL constructs MEX-files from two Fortran packages: a sparse Cholesky factorization package developed by Esmond Ng and Barry Peyton at ORNL and a multiple minimum-degree ordering package by Joseph Liu at University of Waterloo [1].

Built in the high-level programming environment of Matlab, LIPSOL enjoys a much greater degree of simplicity and versatility than codes in Fortran or C language. On the other hand, utilizing efficient Fortran codes for computationally intensive tasks, LIPSOL also has adequate speed for solving moderately large-scale problems even in the presence of overhead

*This work was supported in part by DOE DE-FG02-93ER25171.
[1]We thank these authors for making their codes available to us.

induced from Matlab. LIPSOL has been extensively tested on the Netlib set of linear programs and has effectively solved all 95 Netlib problems.

LIPSOL is free software and comes with no warranty. All files written by this author are copyrighted under the terms of the GNU General Public License as published by the Free Software Foundation.

1.2 What isn't LIPSOL?

Still under development, the current version of LIPSOL has not yet reached the final stage of a production code. In particular, it still lacks a sophisticated post-solution processing procedure to give solution information in terms of the original model. It is known that interior-point methods give solutions in the interior of the solution set whenever it is not a singleton set. LIPSOL does not convert an interior solution to a vertex one and does not have a sensitivity analysis functionality.

1.3 Who should use LIPSOL?

LIPSOL provides another economical means to solve linear programs. For anyone who has access to Matlab but not to a desired commercial linear programming package, LIPSOL may fill the gap.

LIPSOL is specially convenient for numerical experimentation, thus useful for researchers. LIPSOL can read a What-You-See-Is-What-You-Get linear program format that may make it useful in teaching courses that involve solving linear programs.

2 Installation

2.1 Where to get LIPSOL?

A complete LIPSOL package distribution version 0.41 should included with the CD-Rom of this volume. A LIPSOL distribution is a ompressed tar-file named `lipsol*.tar.gz`, where the asterisk is replaced by a particular version number.

One can obtain the latest distribution from the LIPSOL home page on the World Wide Web:

```
http://www.caam.rice.edu/~zhang/lipsol/
```

LIPSOL is also retrievable by anonymous FTP from:

```
ftp.caam.rice.edu:pub/people/zhang/lipsol/
```

When getting the compressed file through FTP, it is often necessary to set the transmission mode to binary.

2.2 How to install LIPSOL?

The installation procedure on UNIX systems is as follows:

1. Get the file `lipsol*.tar.gz`.

2. Move this file to a directory of yours (`$HOME/matlab` is suggested). From there, issue the command: "`gzip -dc lipsol*.tar.gz | tar xvf -`". Here you must have the GNU compression program `gzip` installed on your system. You may use "`zcat`" or "`gunzip -c`" in place of "`gzip -dc`".

3. You will now have a new directory called `lipsol`. Go to the directory `lipsol/src`. You may need to modify the script `Install` to change the `MATLAB` variable to the actual path in which Matlab is installed on your system before proceeding to the next step.

4. Do installation by issuing the command `Install`.

If everything goes as it should, LIPSOL is all set and ready to go.

A special note: In the file `lipsol/src/mps2mat.f`, a number of array-size parameters are set to values that may be too large for systems with small memory. Consider to change these values to smaller values if you receive complaints while running `mps2mat`.

3 Running LIPSOL

3.1 A quick start

In the `lipsol` directory, simply type: `lipsol` at the system prompt. Once at the Matlab prompt ">>", once again enter `lipsol`. LIPSOL will prompt you for a problem name and then go ahead to solve the problem.

For example, to run the problem "recipe", you would do

```
>> lipsol
Enter problem name: recipe
```

Alternatively, you could enter `lipsol('recipe')` or `lipsol recipe` or at once.

Please note that a *problem name* is not a file name, and should not contain any dot. In the above example, the actual file storing the problem "recipe" may be `recipe.mat.gz`.

3.2 Run LIPSOL from any directory

To be able to run LIPSOL from any directory, you need to first set the environment variable `LIPSOL`. Assume that you have installed LIPSOL in the directory `$HOME/matlab/lipsol`. If you are using *csh*, then put the following line in your `$HOME/.cshrc` file:

```
setenv LIPSOL $HOME/matlab/lipsol
```

If you are using *sh* (Bourne shell), then put the following line in your `$HOME/.profile` file

```
LIPSOL=$HOME/matlab/lipsol; export LIPSOL
```

Then copy the file `lipsol` in the `$HOME/matlab/lipsol` directory to a directory on your binary search path, most likely `$HOME/bin`, and delete the second line (containing `setenv LIPSOL $cwd`) from the copied file. Next time you will be able to invoke LIPSOL from any directory by typing `lipsol`.

3.3 Run LIPSOL in the background

To run LIPSOL in the background, go to the directory `lipsol/batch` and issue the command

```
runbg [logfile] &
```

where the `logfile` name is optional. This would invoke LIPSOL in the background to run a list of problems as listed in the file `probs/prob.NAME`, where `NAME` is a string specified in the file `prob.rc`.

For example, if you want to solve two problems, `juliet` and `romeo`, in the background, then you can create a file called, say, `prob.shakespeare` in the directory `probs`, which contains two lines:

```
juliet
romeo
```

In addition, in the file `prob.rc`, you specify by one word `shakespeare` that the problems to be solved are in the file `probs/prob.shakespeare`.

One can also change LIPSOL configuration parameter values by specifying those parameter values in the file `config.rc` (the default values being: 16 8 0). See the description for the command `lipconfig` for more details on LIPSOL configuration.

3.4 Where are the problems located?

LIPSOL supports three linear program input formats: MAT, MPS and LPP, which will be described in detail later. Accordingly, LIPSOL has three default directories for storing problems: `matdir`, `mpsdir` and `lppdir`. In general, a MAT-file should have a file extension `.mat` and be stored in the directory `matdir`; similarly for MPS and LPP-files. However, one does not need to use these default directories and can store problems in any directory as described below.

LIPSOL can also read problems from directories other than the above three, as long as they are specified in the configuration function `lipconfig.m` in the directory `lipsol/solvers/lipcomm` as entries of the string vector PROBDIR. The first entry of PROBDIR is an empty string defined as:

```
PROBDIR=str2mat([]);
```

Do not alter it. If you want LIPSOL to access problems stored in the directory, say, `/usr/users/myname/myprobs`, then add an entry to PROBDIR as below

```
PROBDIR=str2mat(PROBDIR,'/usr/users/myname/myprobs');
```

which follows the first entry. Of course, you can add more than one directories this way. To ensure proper access, the full path of a directory should be specified like in the above example.

If there exist multiple files with the same problem name, for example, at least two files exist out of `foo.mat`, `foo.mps` and `foo.lpp`, LIPSOL will choose one to solve according to the order of MAT, MPS and LPP.

All problem files can be either uncompressed or compressed by either `gzip` or the UNIX `compress`.

4 Input formats

LIPSOL can read linear programs in 3 formats: MAT, MPS and LPP.

4.1 MAT-format

A MAT-file should consist of the following 7 quantities:

```
A b c lbounds ubounds BIG NAME
```

representing the linear program

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & lbounds \le x \le ubounds \end{array} \tag{1}$$

NAME is a string storing the name of a linear program, and BIG is a large number so that $ubounds(i) = BIG$ if there exists no upper-bound for the i-th variable $x(i)$.

If you have generated the seven quantities by whatever means, you can save them into a MAT-file using Matlab's `save` command.

4.2 MPS-format

MPS-format is a de facto industrial standard for describing linear programs, especially the large ones. However, the MPS format is not convenient for small problems. For more details, see the Linear Programming FAQ (frequently asked questions), maintained by Robert Fourer at Northwestern University, which can be found at the Web site:

```
http://www-c.mcs.anl.gov/home/otc/Guide/faq/
```

4.3 LPP-format

LPP stands for LP-Plain. It is a What-You-See-Is-What-You-Get format useful for small problems and classroom use. Here is an LPP-file called, say, `example.lpp`:

```
%This is a simple example.
NAME: example
objective:
  min 2x + 5y - 2.5z
constraints:
  1) x + {sin(pi/4)/4} z < 5
  2) y + z > {exp(2)} x
bounds:
  1) -2 < x < 2
  2) y > 1
  3) z < 3
```

An LPP-file consists of 4 groups of objects: name, objective, constraints and bounds. These markers must be present even when one group is empty (e.g., no bounds exist other than zero-lower bounds). LPP format is case-insensitive, but the following restrictions apply:

1. A colon ":" must appear before a problem name.
2. Either "min" or "max" should precede the expression of an objective.
3. A number and a right parentheses ")" must precede a constraint or a bound.

4. The symbols "<" and ">" should be used instead of "<=" and ">=".

5. A coefficient must be either a number or a Matlab expressions enclosed in a pair of braces.

6. the symbols "$*$" and "/" are not allowed except inside braces.

7. There must be one and only one variable per bound, without a coefficient, or a sign, or an "=" sign. For example, $2x < 1$ and $x+y > 1$ are not legitimate bounds in LPP format; $2 > z$ is legitimate, $-2 < -z$ is not; and $1 < x < 1$ is, $x = 1$ is not (but $x = 1$ can be a constraint).

Zero-lower bounds are implicitly assumed unless otherwise specified. Variables without any bound are treated as free variables. LPP format imposes essentially no restriction on spacing as long as it makes mathematical sense.

5 LIPSOL commands

5.1 List of all commands

LIPSOL commands are stored in the directory `lipsol/solvers/lipcomm`. The `Contents.m` file in that directory list all the LIPSOL commands in the current version. We copy that file below.

```
Basic command.
  lipsol  ---  reads and solves an LP problem in either
               MAT, MPS or LPP format.

Other commands.
  lpformats -- explains MPS, MAT and LPP formats.
  lipconfig -- configuration function for LIPSOL.
  listmat  --- lists MAT-files in /lipsol/matdir.
  listmps  --- lists MPS-files in /lipsol/mpsdir.
  listlpp  --- lists LPP-files in /lipsol/lppdir.
  listall  --- lists all problems reachable by LIPSOL.
  readlpp  --- reads an LPP file into workspace.
  viewlpp  --- displays the LPP model in workspace.
  savelpp  --- saves an LPP model in workspace into a file.
  sol2lpp  --- converts xsol to xlpp (solution to LPP model).
```

The command `viewlpp` and those in the list-family are simple and need no explanations. We will give further details on the rest of the commands.

5.2 lipconfig

`lipconfig` — Configuration function for LIPSOL.

Current version of LIPSOL has four configuration parameters. They are defined as global variables in `lipsol/solvers/lipcomm/lipconfig.m`:

```
global CACHE_SIZE  % cache size in kbytes, default = 16
global LOOP_LEVEL  % loop unrolling level, default = 8
global MONITOR_ON  % 0 = Off,  1 = On,     default = 0
global PROBDIR     % prob directories, default = str2mat([])
```

The first two parameters are required by the ORNL package for sparse Cholesky factorization. The third one controls whether or not turn on a graphic window to monitor the progress of LIPSOL iterations. The last parameter is a string vector designating directories from which LIPSOL can access problems, in addition to the three default directories: `matdir`, `mpsdir` and `lppdir`.

Invoking `lipconfig` without argument will set the default values for these configuration parameters.

The command `lipconfig(VALUES)` resets the first three configuration parameters to the values given in the vector VALUES of three numbers for example, [32 8 1]. Our limited experiments have shown little variations in performance by altering the first two parameters.

Additional problem directories have to be added manually to the file. For example, to add the directory `/usr/users/myname/myprobs` for LIPSOL access, add a line:

```
PROBDIR=str2mat(PROBDIR,'/usr/users/myname/myprobs');
```

The command `lipconfig([])` displays the current configuration.

5.3 lipsol

`lipsol` — Linear programming Interior-Point Solver(s).

The command `lipsol` without argument will prompt for a problem name to run.

The command `lipsol('probname')` will search for `probname.mat` (or its compressed version) first, if not found, then `probname.mps` and then `probname.lpp` (or their compressed versions). Once a file is found, LIPSOL will load and solve the problem. Therefore, if multiple files exist with the same name but different file extensions (formats), LIPSOL will choose one to solve according to the order of MAT, MPS and LPP.

If `probname = []` or `probname = ''` (i.e., an empty string), the file `default.mat` in the `/tmp` directory will be loaded if it exists.

5.4 readlpp

`readlpp` — Reading and converting an LPP-file into `default.mat`.

It reads a LPP-file, checks the syntax, and if correct, converts the problem from the LPP-format to MAT-format. It then stores the converted problem in the file `default.mat`.

The command `readlpp` without argument will prompt for a problem name and then read the problem from the directory `lipsol/lppdir`, if it exists.

The command `readlpp('filename')` will read a problem in the specified file, Here is an example:

```
readlpp('~/matlab/lipsol/lppdir/example.lpp');
```

Note that the path of the file is required.

5.5 savelpp

`savelpp` — Saving an LP model in workspace into a file.

This command should be called after `readlpp` has been called at least once before.

`savelpp` will first ask for permission to save the `default.mat` file into a file called `lipsol/matdir/NAME.mat`, where `NAME` is a problem name extracted from an LPP-file which will be in workspace after `readlpp` is called. If your answer is affirmative and `NAME` exists, the file will be created.

If your answer is negative, `savelpp` will prompt for a `name` and then save the current model into an LPP-file `lipsol/lppdir/name.lpp`; and if no name is given, the model is saved into `lipsol/lppdir/NAME.lpp`, where again `NAME` comes from `readlpp`.

5.6 sol2lpp

`sol2lpp` — Returning a solution of an LPP model.

This command should be called after `lipsol` has solved an LPP-model. It processes the solution returned by `lipsol`, deletes slack variables, displays and returns a solution to the original LPP model in terms of the original variables.

6 Algorithm

A detailed description on the algorithm implemented in LIPSOL is given in a paper by the author [4], along with extensive numerical test results. In

this section, we will only present a simplified illustration for the algorithm used in LIPSOL. We consider the following standard-form of linear program:

$$\min\{c^T x : Ax = b, \;\; x \geq 0\},$$

where $A \in \mathbf{R}^{m \times n}$, $m < n$. Its dual linear program is

$$\max\{b^T y : A^T y + s = c, \;\; s \geq 0\}.$$

The optimality conditions for this linear program are

$$F(z) = \begin{pmatrix} Ax - b \\ A^T y + s - c \\ xs \end{pmatrix} = 0, \;\; (x, s) \geq 0,$$

where the variable $z = (x, y, s)$ contains both the primal and the dual variables and xs denotes the element-wise multiplication of the vectors x and s.

The algorithm is a primal-dual algorithm, meaning that both the primal and the dual programs are solved simultaneously. It can be considered as a Newton-like method applied to the linear-quadratic system $F(z) = 0$ in the above optimality conditions, while keeping the iterates (x^k, s^k) positive (thus the name interior-point method). The algorithm can be schematically described as follows.

Let us consider just one iteration, suppressing the iteration count k for simplicity, at iterate $z = (x, y, s)$ where $(x, s) > 0$. We first compute the so-called prediction direction

$$\Delta z_1 = -[F(z)']^{-1} F(z),$$

which is nothing but Newton's direction; then the so-called corrector direction

$$\Delta z_2 = -[F(z)']^{-1}(F(z + \Delta z_1) - \mu \hat{e})$$

where $\mu > 0$ is called the centering parameter which has to be chosen carefully, and $\hat{e}$ is a zero-one vector with ones corresponding to the nonlinear equations xs in $F(z)$. We combine the two directions with a step-length parameter α and update z to obtain the new iterate

$$z^+ = z + \alpha(\Delta z_1 + \Delta z_2)$$

where the step-length parameter α is chosen so that $z^+ = (x^+, y^+, s^+)$ satisfies $(x^+, s^+) > 0$. The algorithm then repeats these steps at the new iterate until convergence.

The above algorithmic scheme is a variant of the predictor-corrector algorithm proposed by Mehrotra [2] (see also [1, 5]). Conceptually, it is a

relatively simple algorithm though an actual implementation can be much more complicated, especially for large, sparse problems. For more details on Mehrotra's predictor-corrector algorithm and other primal-dual interior-point algorithms, including their theory and implementation, we refer interested users to a recent book by Stephen Wright [3] and many related papers cited in that book.

References

[1] I. J. Lustig, R.E. Marsten, and D.F. Shanno. "On implementing Mehrotra's predictor-corrector interior point method for linear programming," *SIAM J. Optimization* 2 (1992) 435-449.

[2] S. Mehrotra. "On the implementation of a primal-dual interior point method," *SIAM J. Optimization* 2 (1992) 575-601.

[3] S. Wright. *"Primal-Dual Interior-Point Methods."* SIAM, 1997.

[4] Y. Zhang. "Solving large-scale linear programs by interior-point methods Under the MATLAB Environment." To appear in *Optimization Methods and Software*.

[5] Y. Zhang, and D. Zhang. "On polynomiality of the Mehrotra-type predictor-corrector interior-point algorithms," *Mathematical Programming* 68 (1995) 303-318.