LUSOL Matlab Interface Tutorial

Nick Henderson

January 18, 2011

1 Introduction

This tutorial documents the installation and use of a Matlab mex interface to Michael Saunders' LUSOL. The interface, named `lusol_mex`, uses LUSOL to compute a sparse LU factorization of a matrix. The interface provides access to all solve, multiply, and update subroutines. Source code and binaries can be downloaded from

https://github.com/nwh/lusol_mex/

2 System requirements

- Matlab version 7.6 (R2008a) or newer. `lusol_mex` uses a style of object-oriented programming that was introduced in R2008a. Earlier versions of Matlab are not supported.
- Compatible C compiler. Each version of Matlab is compatible with a certain compiler. Refer to the list on Mathwork's website.
- Compatible fortran compiler. This need not be the compiler specified in the Matlab list. The fortran compiler is used to for code that will be linked to the mex routine. It is required that the fortran compiler be "compatible" with the C compiler that mex uses.

`lusol_mex` has been successfully built under a few different scenarios:

- Mac OS X 10.6, Matlab R2009a, `gcc` from Xcode, and `g95` or `gfortran`.
- Ubuntu 10.10 64-bit, Matlab R2010b, `gcc-4.3`, and `gfortran-4.3`.

3 Building

1. Run `mex -setup` to generate `$HOME/.matlab/[VER]/mexopts.sh`.
2. Modify `$HOME/.matlab/[VER]/mexopts.sh` to point to the correct compiler version. The change must be made in the section corresponding to your system architecture. For example, variables for 64-bit linux are set in the `glnxa64` section.
3. Modify the `makefile` in the `lusol_mex/` directory.
4. Run `make`.
5. Make sure the `lusol_mex/` directory is added to your matlab path.

4 Testing

There is a small set of test cases in the `lusol_mex` directory. These require the Matlab xUnit Test Framework to run. See:

http://www.mathworks.com/matlabcentral/fileexchange/22846-matlab-xunit-test-framework

With the testing framework installed, the command `runtests` will execute all tests in `lusol_test.m`.

5 Design

`lusol_mex` makes use of object-oriented programming in Matlab for efficiency and ease of use. Here is an example that factorizes a random sparse matrix:

```
>> rand('twister',123);
>> A = sprand(30,30,.2);
>> mylu = lusol(A);
```

The symbol `mylu` now refers to an object in the Matlab workspace. It contains all of the data for the sparse factorization and provides access to methods for solving $Ax = b$, computing products Ax, and performing stable sparse updates.

```
>> % solve Ax=b
>> b = ones(30,1);
>> x = mylu.solve(b);
>>
```

```
>> % multiply Ax
>> b2 = mylu.mulA(x);
>>
>> % check the result
>> norm(b-b2)

ans =

   1.1285e-14
```

Direct access to the data is not provided and usually not needed. It is possible to obtain sparse L and U factors with the methods:

```
>> L = mylu.L0();
>> U = mylu.U();
```

The method to obtain the L factor is named `L0` to indicate that it returns the initial L factor. LUSOL stores updated to the L factor in product form. The U factor is always maintained. Due to the structure of LUSOL data, the `L0` and `U` methods are required to copy and manipulate data.

6 Usage

The best source of information on the usage of `lusol_mex` is with Matlab's `help` and `doc` commands. Try:

```
>> help lusol
>> % or
>> doc lusol
```

The first thing to do is to create a `lusol` object. This requires a matrix A and possibly some options:

```
>> mylu = lusol(A);
>> mylu = lusol(A,options);
>> mylu = lusol(A,'pivot','TRP','Ltol1','5.0');
```

The first command instantiates the `lusol` object and factorizes A. Matrix A can be scalar (`A = 1`), however it may not be empty (`A = []`). The second command sets parameters using the `options` struct. The third command sets parameters using the key-value format.

6.1 Options

See `help lusol.luset` for details. The best documentation for the input parameters is the fortran code in the file `lusol.f`.

To create an options structure with defaults:

```
>> options = lusol.luset();
```

Options may then be changed by modifying the structure fields. You can also change the default options using key-value pairs in the parameter list to `lusol.luset`. For example:

```
>> options = lusol.luset('pivot','TRP');
```

Note that `lusol.luset` is a "static" method. It can be called without creating a `lusol` object.

6.2 Factorize

The `factorize` method can be used to factorize a (new) matrix after a `lusol` object has already been created. Reallocation will only occur if the new matrix requires more storage or if the parameter `nzinit` is set larger. Example:

```
>> % mylu is already in the workspace
>> [info nsing depcol] = mylu.factorize(A,options);
```

The output:

`info` a status flag

`nsing` number of apparent singularities

`depcol` logical index indicating dependent columns

6.3 Solve

The `solve` method allows solves with A, A^T, L, L^T, U, U^T. See `help lusol.solve`. The relevant methods are:

- `solveA`
- `solveAt`
- `solveL`

- `solveLt`
- `solveU`
- `solveUt`

These methods all call `solve` with the correct mode.

6.4 Multiply

The `mul` method allows products with A, A^T, L, L^T, U, U^T. See `help lusol.mul`. The relevant methods are:

- `mulA`
- `mulAt`
- `mulL`
- `mulLt`
- `mulU`
- `mulUt`

These methods all call `mul` with the correct mode.

6.5 Update

`lusol_mex` provides access to all update subroutines in LUSOL:

`repcol` replace a column

`reprow` replace a row

`addcol` add a row

`addrow` add a column

`delcol` delete a row

`delrow` delete a column

`r1mod` rank 1 update