MATLAB in Numerical Linear Algebra Research

Edmond Chow

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Themes

Efficiency

- MATLAB can now work on “large problems”
- It’s good to have some knowledge of what’s under the hood
  - Sparse matrices
  - Reading sparse matrix data

Expediency

- Interaction between MATLAB and the rest of your work environment
  - Passing arguments to functions
  - Calling C/Fortran from MATLAB, and vice-versa
Creating sparse matrices

Sparse matrices are stored column-by-column. Creating a sparse matrix by inserting nonzeros is very slow. Use `sparse` instead. However, the ordering also matters:

```matlab
>> n=1000;
>> a=sprandn(n,n,.5);
>> [i j k]=find(a);
>> tic;b=sparse(i,j,k,n,n);toc % column based
elapsed_time =
  0.0957
>> tic;b=sparse(j,i,k,n,n);toc % row based
elapsed_time =
  0.6152
```

Similarly, traverse your sparse matrix by columns.
Reading sparse matrices stored in files

- MATLAB’s binary MAT-files
- Harwell-Boeing format
- Matrix-Market format

Reading a wind-tunnel matrix, 200K rows, 10M nonzeros, a 413 Mbyte file in coordinate format:

```matlab
>> tic; [i j k] = textread('pwtk', '%f %f %f'); toc
elapsed_time =
   1.0126e+03
>> tic; load pwtk; toc
elapsed_time =
    182.4648
>> fid = fopen('pwtk', 'r');
>> tic; M = fscanf(fid, '%f'); toc
elapsed_time =
    99.3915
>> M = reshape(M, 3, num_nonzeros);
>> M = spconvert(M');
```

Loading the binary version takes 6 seconds.
Number of nonzeros in sparse matrix addition

\[ \text{nzmax}(A+B) = \text{nnz}(A) + \text{nnz}(B) \]

\[
>> \text{a}=\text{sprandn}(1000,1000,.5);
>> \text{nnz(a)}
\]
\[
\text{ans} =
\]
\[
393537
\]
\[
>> \text{b}=\text{a}+\text{a};
>> \text{nnz(b)}
\]
\[
\text{ans} =
\]
\[
787074
\]
\[
>> \text{b}=\text{b}+\text{sparse(0)};
>> \text{nnz(b)}
\]
\[
\text{ans} =
\]
\[
393537
\]
Number of nonzeros in sparse matrix multiplication

\[ \text{nzmax}(A) < 1.5 \text{ nnz}(A) ? \]

```matlab
>> \text{nnz}(b)
ans =
    72879
>> \text{nzmax}(b)
ans =
    83802
>> b=b*b;
>> \text{nnz}(b)
ans =
    203123
>> \text{nzmax}(b)
ans =
    275711
>> b=b*b;
>> \text{nnz}(b)
ans =
    720208
>> \text{nzmax}(b)
ans =
    3153350
```
Interfaces for Conjugate Gradient, I

function x = cg(A, b, M, tol, maxit, x0, printout)

- Trailing input arguments can be optional (default values used instead)
- In the function, nargin is the number of input arguments; nargout is also available
- Matrix M is an approximation to matrix A
Interfaces for Conjugate Gradient, II

function x = cg(Afun, b, Mfun, params)

- Afun and Mfun can be functions:
  cg(‘amult’, b, ’precon’, params), with
  \( v = amult(u) \) and \( v = precon(u) \)

- Global variables can be used to pass parameters to amult and precon

- params is an array storing tol and maxit
function x = cg(b, params, Afun, AfunARG, ... Mfun, MfunARG, P1, P2, P3, P4, P5, P6, P7, P8);

- AfunARG and MfunARG specify which parameters to pass to Afun and Mfun
- cg(b, params, 'amult', [1] 'precon', [2 3], A, L, U) with the functions \( v = amult(u,A) \), and \( v = precon(u,L,U) \)
- Strings containing the calls to Afun and Mfun are created and the functions are called using eval
function x = cg(Afun, b, tol, maxit, M1fun, M2fun, x0, varargin)

- \textit{Afun}, \textit{M1fun}, and \textit{M2fun} each receive all the arguments
- \texttt{cg('amult', b, tol, maxit, 'Mleft', 'Mright', x0, A, L, U)} requires the functions \( v = \text{amult}(u,A,L,U) \), \( v = \text{Mleft}(u,A,L,U) \), and \( v = \text{Mright}(u,A,L,U) \)
- \texttt{varargin} is a cell array; varargout is also available
Using structures to help implement Multigrid

- Structures and cell arrays can be used to pass complex sets of parameters to functions
- Structure can change from call to call
- Also useful when many functions need similar information

```plaintext
lev.a = a;
lev.level = 1;

% set up prolongators and coarse grid operators
lev = setup(lev, nlevels);

% one V-cycle
x = zeros(n,1);
delta = solve(lev, rhs-a*x, nlevels);
x = x + delta;
```
setup **function**

    function lev = setup(lev, nlevels);

    if (lev.level == nlevels)
        [lev.lexact lev.uexact] = lu(lev.a);
        return;
    end

    lev.p = prolongator(lev.a);
    lev.next.a = lev.p' * lev.a * lev.p;
    lev.next.level = lev.level + 1;
    lev.next = setup(lev.next, nlevels);
solve function

function v = solve(lev, rhs, nlevels)

    if (lev.level == nlevels)
        v = lev.uexact \ (lev.lexact \ rhs);
        return;
    end

    % pre-smoothing
    x = zeros(n,1);
    x = smooth(lev, x, rhs, 1);

    % compute new rhs (for the error equation)
    rhs2 = lev.p' * (rhs - lev.a * x);

    % solve the error equation recursively
    x2 = solve(lev.next, rhs2, nlevels);

    % correct
    x = x + lev.p * x2;

    % post-smoothing
    v = smooth(lev, x, rhs, 1);
int FGMRES(const mxArray *A, double *b, double *x, int dim, double tol, 
        int max_iter, int print, char *precon, int narg, const mxArray *ap[]);

sol = fgmres(a, rhs, guess, dim, tol, max_iter, print, 'lusol', L, U);

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    int n, dim, max_iter, iter, print;
    double tol;
    char precon_name[101];
    char *precon = precon_name;

    n = mxGetN(prhs[0]); /* num cols in matrix */
    dim = (int) *mxGetPr(prhs[3]);
    tol = (double) *mxGetPr(prhs[4]);
    max_iter = (int) *mxGetPr(prhs[5]);
    print = (int) *mxGetPr(prhs[6]);
    mxGetString(prhs[7], precon, 100);

    /* create solution vector, and copy the guess into it */
    plhs[0] = mxCreateDoubleMatrix(n, 1, mxREAL);
    memcpy(mxGetPr(plhs[0]), mxGetPr(prhs[2]), n*sizeof(double));

    iter = FGMRES(prhs[0], mxGetPr(prhs[1]), mxGetPr(plhs[0]), 
                   dim, tol, max_iter, print, precon, nrhs-8, &prhs[8]);
}
Calling MATLAB from MEX-files

```c
/* v = mat * u */
static void Matvec(int n, const mxArray *mat, double *u, double *v,
                    mxArray *u_array)
{
    mxArray *rhs[2];
    mxArray *lhs[1];

    rhs[0] = (mxArray *) mat;
    rhs[1] = (mxArray *) u_array; /* u_array structure is reused */
    mxSetPr(u_array, u);

    mexCallMATLAB(1, lhs, 2, rhs, "*");

    memcpy(v, mxGetPr(lhs[0]), n*sizeof(double));
    mxDestroyArray(lhs[0]);
}
```
Miscellaneous stuff

- Make sure column indices are ordered: $a = \left( a' \right)'$
- Force a row or column vector to be a column vector: $v(:, )$
- Replication: $a=5; \ a(\text{ones}(5,5))$
- To invert a permutation vector $p$, of length $n$: $r(p) = 1:n$
- Strictly lower triangular matrix: $\text{tril}(a, -1)$
- Sub-functions
- The clear function
Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.