MACM 316 — Computing Assignment #2

David Pham / dhpham@sfu.ca / 301318482

Values of N and N_{ex}

I knew that I was eventually going to take $log_{10}(N)$ and log_{10}(avg time to solve mtx), so I chose to define $N = \lceil 10^k \rceil$, where $k = 1, 1.52, 2.53$, to correspond nicely with log_{10} values.

I started by defining `dense_Nex`, `tri_Nex`, and `perm_Nex` to be the number of solves for each type of matrix $[M_d]$, $[M_t]$, and $[M_p]$, respectively. After some fiddling around, I managed to find values for each `*_Nex` that would automatically scale well as N increased each loop.

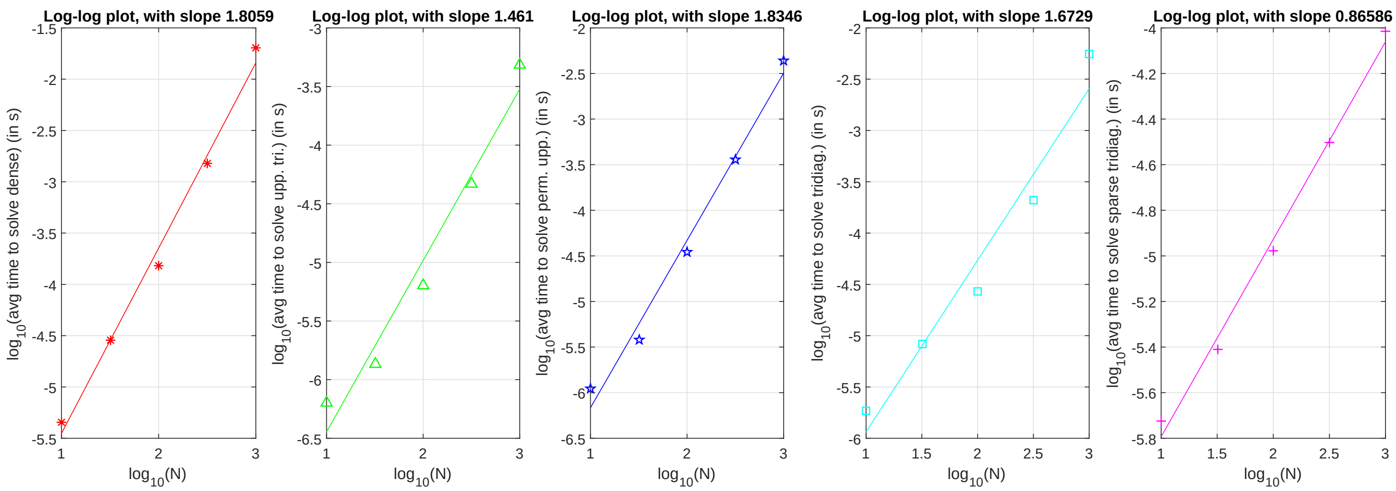

Figure 1: log-log plots for dense, upper triangular, permuted upper triangular, tridiagonal, and sparse tridiagonal matrices.

Conclusions

It seems like using *backslash* in MATLAB somehow cuts down on the computational cost as N increases. Theoretically, the number of flops in solving $[M_d]$ should increase by $O(N^3)$, giving the log-log plot a slope of 3. However, the computations have shown the slope to be ≈ 1.8 (Figure 1), which is jarring. Perhaps, since we don't generate a new random matrix for each solve, MATLAB caches the LU factorization of the Matrix somehow?

Interestingly, the slope in the log-log plot of the permuted upper triangular matrix $[M_p]$ is greater than that of the dense matrix $[M_d]$. MATLAB seems to treat them both similarly, but solving $[M_p]$ takes longer for some reason. I take this to mean that MATLAB doesn't recognize the matrix as being a permuted upper triangular matrix, and tries to solve it as a dense matrix: as MATLAB goes about row reduction, it often encounters situations where row interchanges are necessary, unlike in the dense matrix case.

$[M_3]$ still requires some row reduction, whereas $[M_t]$ requires only back-subsitution. This bit of reduction is still enough to put tridiagonal matrices behind upper triangular matrices, and the time it takes to solve $[M_3]$ is somewhere between the time it takes to solve a dense matrix and an upper triangular matrix. Out of these matrices, $[M_{3s}]$ is the fastest type of matrix to solve: its slope is ≈ 0.87, making it even faster than $O(n)$.

```
%
%  ca2_demo.m -- timing exercise (macm316, hl -- 13 jan 2019)
%
% Purpose:      This script serves as a demo for students to build on
 in
%               completing computing assignment 2. This script builds
 three types
%               of NxN matricies: dense, upper triangular, and
 permuted upper
%               triangular. It then performs a matrix solve with each,
 Nex number
%               of times. The time it takes for Nex number of solves
 is used to
%               estimate the time of one solve .
%
% Instructions: Start by running the script once, and see the
%               output for the esimtated times. Note: the choice of
 Nex here may
%               not give accurate results for all three matrix types.
 Next, copy
%               and paste this code into your own Matlab
%               file. Follow the assignment sheet for further
 instructions to
%               complete your report.

%  experimental parameters

n = 3;
N = zeros(2*n-1, 1);

dense_Nex = ones(2*n-1, 1);
avg_dense_time = zeros(2*n-1, 1);

tri_Nex = ones(2*n-1, 1);
avg_tri_time = zeros(2*n-1, 1);

perm_Nex = ones(2*n-1, 1);
avg_perm_time = zeros(2*n-1, 1);

tridiag_Nex = ones(2*n-1, 1);
avg_tridiag_time = zeros(2*n-1, 1);

sparse_tridiag_Nex = ones(2*n-1, 1);
avg_sparse_tridiag_time = zeros(2*n-1, 1);

i=1;
for k = 1:0.5:n
    % Fill N with values of 10^1, 10^1.5, ... 10^3.5, 10^4
    N(i) = ceil(10^k);

    % size exists only for convenient terminal output
    size = N(i)
```

```
   %  three matrix types

   %  dense matrix (no zeros)
   Md = randn(N(i),N(i));

   %  upper triangular
   Mt = triu(Md);

   %  randomly row-exchanged upper triangular (these are tricky array
commands,
   %  but if you run a small sample, it is clear they do the right
thing)
   idx=randperm(N(i));
   Mp = Mt(idx,:);

   %Tri-diagonal and sparse tri-diagonal
   M3 = diag(diag(Md))+diag(diag(Md,-1),-1)+diag(diag(Md,1),1);
   M3s=sparse(M3);

   %  exact solution of all ones
   x = ones(N(i),1);

   %  right-side vectors
   bd = Md*x;
   bt = Mt*x;
   bp = bt(idx);
   b3dg = M3*x;
   b3dgs = M3s*x;

   %  define # of experiments per matrix type
   dense_Nex(i) = 4440*ceil(10^(n-k));
   tri_Nex(i) = 100000*ceil(10^(n-k));
   perm_Nex(i) = 20000*ceil(10^(n-k));
   tridiag_Nex(i) = 5600*ceil(10^(n-k));
   sparse_tridiag_Nex(i) = 100000*ceil(10^(n-k));

   %  dense test
   tic
   for jj = 1:dense_Nex(i)
       xd = Md\bd;
   end
   dense_time=toc;

   %  upper tri test
   tic
   for jj = 1:tri_Nex(i)
       xt = Mt\bt;
   end
   tri_time=toc;

   % permuted upper tri test
   tic
   for jj = 1:perm_Nex(i)
```

```
        xp = Mp\bp;
    end
    perm_tri_time=toc;

    %  tridiagonal test
    tic
    for jj = 1:tridiag_Nex(i)
        x3dg = M3\b3dg;
    end
    tridiag_time=toc;

    %  sparse tridiagonal test
    tic
    for jj = 1:sparse_tridiag_Nex(i)
        x3dgs = M3s\b3dgs;
    end
    sparse_tridiag_time=toc;

    %Computing avgerage solve times
    avg_dense_time(i) = dense_time/dense_Nex(i);

    avg_tri_time(i) = tri_time/tri_Nex(i);

    avg_perm_time(i) = perm_tri_time/perm_Nex(i);

    avg_tridiag_time(i) = tridiag_time/tridiag_Nex(i);

    avg_sparse_tridiag_time(i) = sparse_tridiag_time/
sparse_tridiag_Nex(i);

    % You may find the following code helpful for displaying the
 results
    % of this demo.
    type_times = {'Dense',avg_dense_time(i), ...
                    'Upper Triangular', avg_tri_time(i), ...
                    'permuted Upper Triangular', avg_perm_time(i), ...
                    'Tridiagonal', avg_tridiag_time(i), ...
                    'sparse Tridiagonal', avg_sparse_tridiag_time(i)};
    fprintf(' \n')
    fprintf('Estimated time for a %s matrix is %f seconds.
 \n',type_times{:})

 fprintf('===========================================================================
\n')
    i = i+1;
end

%  Begin plotting log-log graph of results

%  to be used as x axis
logN = log10(N);
%  (2*n)-1 x 3 sized matrix, each column a diff. matrix type
logTimes = log10([ avg_dense_time, avg_tri_time, avg_perm_time, ...
                    avg_tridiag_time, avg_sparse_tridiag_time]);
```

```matlab
%  hold the log slope for each matrix type
p = ones(5,2);

%  hold appropriately ordered ylabels for log-log plot
logylabels = ["log_{10}(avg time to solve dense) (in s)", ...
            "log_{10}(avg time to solve upp. tri.) (in s)", ...
            "log_{10}(avg time to solve perm. upp.) (in s)", ...
            "log_{10}(avg time to solve tridiag.) (in s)", ...
            "log_{10}(avg time to solve sparse tridiag.) (in s)"];
%  colours for plot of each matrix type
plotopts = ["r*", "g^", "bp", "cs", "ms"];

x1 = [logN(1):0.01:logN(end)];

% Output
for j = 1:5
    p(j,:) = polyfit(logN, logTimes(:,j), 1);
    display(['Slope of best fit line is  : ',num2str(p(j,1))])

    % Plotting
    figure(1)

    % Plot of log data
    subplot(1,5,j)
    % plot discrete points
    plot(logN, logTimes(:,j), plotopts(j))
    hold on;
    % plot bestfit line
    y1 = polyval(p(j,:), x1);
    plotcolour = char(plotopts(j));
    plot(x1,y1,plotcolour(1))

    grid on
    xlabel('log_{10}(N)')
    ylabel(logylabels(j))
    title(['Log-log plot, with slope ', num2str(p(j,1))])
end
```

Published with MATLAB® R2018a