Experiments with LU preconditioning

We have discussed the use of LSQR or LSMR on the least-squares problem
\[ \min \| Ax - b \|_2^2. \]
We now know that the very least we should do to help LSQR/LSMR
is to use diagonal preconditioning:
\[ \min_y \| ACy - b \|, \quad x = Cy, \]
where \( C \) is a diagonal matrix that scales the columns of \( A \) to give them unit 2-norm. This is straightforward if \( A \) is available as a sparse matrix (but not if \( A \) is an operator). In the project we first normalize the columns of \([ A \ b]\), so we may assume
that diagonal preconditioning has already been done and that we want to solve a
modified problem
\[ \min \| Ax - b \|, \quad \text{where } [A \ b] \text{ has normalized columns.} \quad (1) \]

Now suppose we compute \( PAQ = LU \), where \( P \) and \( Q \) are permutations chosen
to keep \( L \) and \( U \) sparse, and most importantly to keep \( L \) well-conditioned. \((L \text{ should have unit diagonals and bounded subdiagonals.})\) Assuming \( A \) has full column rank,
we see that \( AQU^{-1} = P^T L \) should be well-conditioned. Hence, as long as \( U \) is not
too ill-conditioned, we can solve problem (1) as follows:
\[ \min_y \| AQU^{-1}y - b \|, \quad Uw = y, \quad x = Qw. \quad (2) \]

The aim of our project is to experiment with this approach using data \( A \) and \( b \)
from the LPnetlib group of Tim Davis's University of Florida Sparse Matrix collection
(\( \text{http://www.cise.ufl.edu/research/sparse/matrices/} \)). In particular,
the problems \( \text{lp\_pilot*\_mat} \) are linear optimization models created by Professor
George Dantzig and colleagues here at Stanford during the 1980s. Our data \([A \ b]\) is
obtained by normalizing \([A^T \ c]\) from the linear programs.

Download \( \text{www.stanford.edu/class/msande318/homework/LUprecond.zip} \) and
unpack the files

\begin{verbatim}
AUinv.m
lsmr.m
lsmrLUtest.m
lp_pilot4.mat
lp_pilot_we.mat
lp_pilotnov.mat
lp_pilot JA.mat
lp_pilot.mat
lp_pilot87.mat
\end{verbatim}

(with the \( \text{lp\_pilot} \) models ordered from smallest to largest). Study \( \text{lsmrLUtest.m} \)
and observe that it proceeds as follows:
1. Perturb $A$ very slightly to make sure that no row contains only one nonzero. (See why later.)

2. Normalize the columns of $[A \ b]$.

3. Use MATLAB’s sparse QR to solve (1). $x_{QR} = A\backslash b$ would do the same thing, but the chosen method allows us to save the $R$ factor and see $\text{nnz}(R)$.

4. Use LSMR to solve (1).

5. Use MATLAB’s first sparse LU method to solve (2). This method is due to Gilbert and Peierls, and we refer to it as GP LU. It maintains the bound $L_{ij} \leq 1/\text{thresh}$. With $\text{thresh} \in [0, 1]$ and not too small (larger than 0.2 say), $L$ should be well-conditioned and LSMR should converge more quickly.

6. Display results.

Note that the private function $\text{AUinv.m}$ and the anonymous (in-line) functions

\[
\text{AUinverse} = @(x,\text{mode}) \text{AUinv}(x,\text{mode},\text{AQ},U);
\]

\[
\text{AUinverse2} = @(x,\text{mode}) \text{AUinv}(x,\text{mode},\text{AQ},U2);
\]

are used to implement the GP LU preconditioning.

**Project tasks**

The aim is to use MATLAB’s second (more recent) sparse LU to solve (2). This makes use of Tim Davis’s software UMFPACK. Like GP LU, it maintains the bound $L_{ij} \leq 1/\text{thresh}$ as long as $A$ has no singleton rows. (Hence our need for step 1 above.) Again, LSMR should converge well if $\text{thresh}$ is not too small.

1. Study the way in which GP LU preconditioning has been implemented, given that $PAQ = LU$. From (2) we should say that the right-preconditioner is $M = UQ^T$, where we would need to solve systems of the form $Mx = y$. Instead we work with a private function $\text{AUinv}$ that thinks it is forming matrix-vector products with an operator $AU^{-1}$, for some $A$ and $U$. Explain why the function handle $\text{AQUinverse} = @(x,\text{mode}) \text{AUinv}(x,\text{mode},\text{AQ},U)$; performs the required function.

2. After the first LU method, add code to $\text{lsmrLUtest.m}$ to implement a second LU preconditioning method, making use of $[L2,U2,P2,Q2] = \text{lu}(A,\text{thresh})$; to obtain a column ordering $Q2$ and triangular factor $U2$. Note that we need a new function handle $\text{AQUinverse2}$.

3. Run $\text{lsmrLUtest}('lp_pilot87',\text{thresh})$ for various values of $\text{thresh}$ and collect the statistics printed at the end. Plausible values for $\text{thresh}$ are in the range $[0.2,0.9]$, but for interest you could try other values. Summarize the results you obtain by describing any apparent trend. Because this is the largest test case, it should make trends more evident. Beware that the times given by $\text{tic}$ and $\text{toc}$ may not be especially accurate.

4. Experiment with the other data sets in turn. Again describe any apparent trend.
function lsmrLUtest(name,thresh)

% function lsmrLUtest('lp_pilot4',thresh)
% loads file lp_pilot4.mat from the LPnetlib group of Tim Davis's
% University of Florida Sparse Matrix collection
% (http://www.cise.ufl.edu/research/sparse/matrices/),
% sets A=A' and b=c, normalizes the columns of [A b], and
% solves min ||Ax-b|| by various means. "thresh" is an input
% to Matlab's sparse LU functions. It should be in the range [0,1]
% and not too small: say thresh >= 0.2.
%
% 24 May 2007: First version for testing LU preconditioning of LSQR.
% Michael Saunders, class MS&E 318.
% 02 Jun 2012: Use LSMR in place of LSQR.
% 03 Jun 2012: Method 1 uses sparse QR as benchmark.
% Method 2 uses Matlab's original sparse LU.
% Method 3 uses sparse LU from UMFPACK2. This
% requires Tim Davis's SuiteSparse software:
% http://www.cise.ufl.edu/research/sparse/SuiteSparse/
% 03 Jun 2012: Guard against singular preconditioner by checking if
% condest(R)==Inf, which would mean cond(U) = Inf.
% 05 Jun 2012: Perturb A so that no rows have just one nonzero.
% Method 3 now uses the UMFPACK factors [L,U,P,Q] = lu(A)
% avoids because L is bounded when A has no singleton rows.
% 06 Jun 2012: Method 3 is omitted here. To be added as projectLU assignment.

load(name);
A = Problem.A;
c = Problem.aux.c;
A = A';
b = c;
[m,n] = size(A);

%% Prevent A from having any singleton rows.
lenr = sum(A~=0,2); % Count nonzeros in each row of A
lenr = full(lenr);
S = find(lenr==1); % Set of singleton rows
A(S,n) = A(S,n) + eps; % Perturb A in last column
lenr = sum(A~=0,2); % Count nonzeros in each row of A
lenr = full(lenr);
S = find(lenr==1); % Set of singleton rows
A(S,n-1)= A(S,n-1) + eps; % Perturb A in column n-1

%% Normalize the columns of [A b]. Error if b=0 or A'b=0.
cnorms = sqrt(sum(A.*A,1));
C = diag(sparse(1./cnorms));
A = A*C;
b = b/norm(b);
if (norm(b,inf) < 100*eps), error(' b = 0'), end
if (norm(A'*b,inf) < 100*eps), error('A''*b = 0'), end

%% The problem is now min ||Ax - b||
% where the columns of [A b] have been normalized.
% Solve with sparse QR. Error if R is singular.
t = tic;
q = colamd(A); % xQR = A\b uses sparse QR to do
[c,R] = qr(A(:,q),b,0); % essentially this
if (condest(R) == Inf), error('Singular problem'), end
xQR = R\c;
xQR(q) = xQR;
itn0 = 0;
e0 = 0;
t0 = toc(t);

%% Solve min ||Ax - b|| as it stands
fprintf(fout,'\n\nSolve LS problem (with columns already normalized)\n');
t = tic;
daamp = 0;
lambda = 0;
atol = 1e-10;
btol = atol;
conlim = 1e+10;
itnlim = 10000;
localSize = 0; % 0 = no local reorthogonalization
show = true;

[x1, istop1, itn1, normr, normAr, normA, condA, normx]...
  = lsmr(A, b, lambda, atol, btol, conlim, itnlim, localSize, show);
e1 = norm(x1-xQR,inf);
t1 = toc(t);

%% Solve with LU preconditioning (using original Matlab sparse LU)
fprintf('

 Solve with LU preconditioning (inv(U) = right-preconditioner)\n')
t = tic;
q = colamd(A); % The operator is (AQ)\U
AQ = A(:,q);
[L,U,P] = lu(AQ,thresh);
Lmax = max(max(abs(L)));
Lmax = full(Lmax);
AQUinverse = @(x,mode) AUinv(x,mode,AQ,U);

[y2, istop2, itn2, normr, normAr, normA, condA, normx]...
  = lsmr(AQUinverse, b, lambda, atol, btol, conlim, itnlim, localSize, show);
x2 = U'y2;
x2(q) = x2;
e2 = norm(x2-xQR,inf);
t2 = toc(t);

%% Display results
fprintf('
 lsmrLUtest(''%s''), name)
fprintf('
 m , n %8g %10g',m,n)
fprintf('
 nnz(A), nnz(R) %8g %10g',nnz(A),nnz(R))
fprintf('
 nnz(L), nnz(U) GP %8g %10g',nnz(L),nnz(U))
fprintf('
 nnz(L), nnz(U) UMFPACK %8g %10g',nnz(L2),nnz(U2))

Method itns time error thresh Lmax
QR %8g %10.2f %10.1e
lsmr %8g %10.2f %10.1e
GP LU preconditioning %8g %10.2f %10.1e %8.3f %9.2e

return

%%%% Private function
function y = AUinv(x,mode,A,U)

% Helper function for running LSQR/LSMR with the
% matrix operator A*inv(U).
%
% 10 May 2007: First version for testing with lsqrSOL.m.
% 03 Jun 2012: Experimented with mode 2.
if mode==1
  y = A*(U'\x);
else
  y = (x'*A)/U; % This still seems fastest
  y = U'\(A'*x); % This seems slover, even though
  % A'*x is better than (x'*A)' nowadays
  % opts.UT = true; % Doesn't work for sparse U
  % opts.TRANSA = true;
  % y = linsolve(U,A'*x,opts);
end