

Math221 Homework # 4 Solutions (Fall 2009)

Instructor: James W. Demmel

October 9, 2009

Question 1: 2.19.

Question 1, Part 1. First Solution: Following the hint, we assume A a singular and get a contradiction. If A is singular, there must be a nonzero vector x such that $x^T A = 0$. Suppose that the largest component in absolute value of x is x_i ; divide x by x_i so that this component is 1 and the other components of x are at most 1 in absolute value. Now look at the i -th component of $y^T = x^T A$, namely $y_i = \sum_{j=1}^n x_j A_{ji} = A_{ii} + \sum_{j \neq i} x_j A_{ji}$. By the triangle inequality $|\sum_{j \neq i} x_j A_{ji}| \leq \sum_{j \neq i} |x_j| \cdot |A_{ji}| \leq \sum_{j \neq i} |A_{ji}|$ so that $|y_i| \geq |A_{ii}| - |\sum_{j \neq i} x_j A_{ji}| \geq |A_{ii}| - \sum_{j \neq i} |A_{ji}|$, which is positive by diagonal dominance of A , so $y \neq 0$ and A cannot be singular.

Second (similar) Solution: Gershgorin's Theorem says that the eigenvalues of A lie in disks with centers at A_{ii} and radii equal to the off diagonal row sums: $\sum_{j \neq i} |A_{ij}|$. By applying Gershgorin's Theorem to A^T , which has the same eigenvalues as A , we may also use the off diagonal column sums $\sum_{j \neq i} |A_{ji}|$. Now if A is *strictly column diagonally dominant*, i.e. $|A_{ii}| > \sum_{j \neq i} |A_{ji}|$, so the center of each disk is farther away from the origin than its radius, then no disk can contain the origin, so 0 cannot be an eigenvalue, so the matrix cannot be singular.

Question 1, Part 2. We use induction as suggested. By the definition of strict column diagonal dominance, the largest entry in the first column in absolute value is on the diagonal, so column pivoting does not need to exchange rows. Now we need to show that after one step of Gaussian elimination, the Schur complement is still strictly column diagonally dominant. It will be enough to show that the first column of the Schur complement is strictly column diagonally dominant.

Write $A = \begin{bmatrix} a & c_1 & c_2 \\ b_1 & \hat{A}_{11} & \hat{A}_{12} \\ b_2 & \hat{A}_{21} & \hat{A}_{22} \end{bmatrix}$, where a, c_1, b_1 and \hat{A}_{11} are all 1-by-1, so that the first column of

the Schur complement is $\begin{bmatrix} \hat{A}_{11} - b_1 c_1 / a \\ \hat{A}_{21} - b_2 c_1 / a \end{bmatrix}$. Strict column diagonal dominance in the first column of the Schur complement requires us to prove that $|\hat{A}_{11} - b_1 c_1 / a| > \|\hat{A}_{21} - b_2 c_1 / a\|_1$. We do this as

follows:

$$\begin{aligned}
\|\hat{A}_{21} - b_2 c_1/a\|_1 &\leq \|\hat{A}_{21}\|_1 + \|b_2 c_1/a\|_1 \\
&= \|\hat{A}_{21}\|_1 + \|b_2\|_1 \cdot |c_1/a| \\
&< |\hat{A}_{11}| - |c_1| + \|b_2\|_1 \cdot |c_1/a| \\
&\quad \dots\text{by strict diagonal dominance of } A \text{ in column 2} \\
&< |\hat{A}_{11}| - |c_1| + (|a| - |b_1|) \cdot |c_1/a| \\
&\quad \dots\text{by strict diagonal dominance of } A \text{ in column 1} \\
&= |\hat{A}_{11}| - |b_1| \cdot |c_1/a| \\
&\leq |\hat{A}_{11} - b_1 \cdot c_1/a| \quad \dots\text{as desired.}
\end{aligned}$$

Question 2.

Note that multiplying any matrix B times the vector e of all ones gives a vector Be of row sums of B . Similarly $e^T B$ gives a vector of column sums of B . Now if e has some entries equal to 1 and some zero, Be will sum just those entries in each row of B where there are 1s in e , and analogously for $e^T B$. Now let e^{odd} be the vector with ones in odd locations and zeros in the even locations, and e^{even} have ones in even locations and zeros in odd locations. Then Be^{odd} is a vector containing the sum of the odd numbered entries of each row of B , and $(e^{even})^T (Be^{odd})$ adds these sums just for the even numbered rows, so that

$$(e^{even})^T (Be^{odd}) = \sum_{i \text{ even and } j \text{ odd}} B_{ij}$$

Thus we want to compute this for $B = A^{-1}$. We do not need to compute A^{-1} explicitly (indeed a point of this question is that you hardly ever need to compute A^{-1} explicitly!), instead we just

1. Factor $A = PLU$ using Gaussian elimination with partial pivoting; cost = $\frac{2n^3}{3} + O(n^2)$
2. Use this to solve $Ax = e^{odd}$ for x ; cost = $2n^2 + O(n)$
3. Compute the dot product $(e^{even})^T x$; cost = $O(n)$

for a total cost of $\frac{2n^3}{3} + O(n^2)$, several times cheaper than computing the inverse explicitly.

Question 3.

Question 3, Part 1. Note that the recursive routine RLUint has just two integers as arguments: i (where (i,i) is the top left corner of the submatrix on which the algorithm is to be applied), and m (the number of columns of this submatrix). The matrix is updated in-place, just as with non-recursive LU, and so is accessed as a global variable called Ac (a copy of the original input matrix A). The only output argument is an integer called err , which is zero if the routine executes normally (the usual case), and positive if a zero pivot is encountered (only possible if the matrix is very close to singular). In particular RLUint has no matrix inputs or outputs, because that means whole submatrices would be copied in memory whenever RLUint is called or returns, which is not good since our goal is minimizing data movement! RLUint deals with odd m and avoids division by 0, which makes it slightly more complicated than the simplest possible solution.

```

% Recursive LU with partial pivoting
function [L,U,P]=RLU(A)
% On input
% Assume if A is N by M, then N >= M
% On output, same as Matlab's lu:
% L is unit lower triangular,
% U is upper triangular,
% P is a permutation matrix
% P*A = L*U
global Ac PermLU N ;
[N,M]=size(A); Ac = A; PermLU = [1:N]; m=M;
% Call recursive LU on copy Ac of input matrix
err = RLuint(1,m); % the name is short for "RLU internal"
if (err > 0)
    disp(['Zero pivot in column ',int2str(err)])
end
L = tril(Ac,-1)+eye(N); U = triu(Ac,0);
P = eye(N); P = P(PermLU,:);
return

function err = RLuint(i,m)
% Perform GEPP recursively in place on submatrix Ac(i:N, i:i+m-1)where
% Ac is a copy of the original input matrix, initialized before the first
% call to RLuint
% N is the number of rows of the original input matrix
% PermLU is a vector of row indices that records permutations;
% it is initialized to [1:N] before the first call to RLuint;
% whenever rows i and j of Ac are interchanged, entries i and j
% of PermLU are also interchanged
global Ac PermLU N ;
if (m==1) % just one column to factor
    [Y,I]=max(abs(Ac(i:N,i))); % find largest entry in column
    if (I(1) ~= 1), % if largest entry not at top
        PermLU([i,i-1+I(1)]) = PermLU([i-1+I(1),i]); % update permutation
        Ac([i,i-1+I(1)],:) = Ac([i-1+I(1),i],:); % swap rows of matrix
    end
    err = 0;
    if (Ac(i,i) == 0) % error: indicate zero pivot in column i
        err = i;
    elseif (i < N) % usual case: compute column i of L
        Ac(i+1:N,i) = Ac(i+1:N,i)*(1/Ac(i,i));
    end
else % more than one column
% divide m columns into left and right halves; deal with m being odd
mL = floor(m/2); mR = ceil(m/2);

```

```

% do LU on left half of matrix
err = RLUint(i,mL);
if (err == 0), % provided no zero pivot encountered, continue
% Update U
Ac(i:i+mL-1,i+mL:i+m-1) = ...
    (tril(Ac(i:i+mL-1,i:i+mL-1),-1)+eye(mL))\Ac(i:i+mL-1,i+mL:i+m-1);
% Update Schur Complement
Ac(i+mL:N,i+mL:i+m-1) = Ac(i+mL:N,i+mL:i+m-1) - ...
    Ac(i+mL:N,i:i+mL-1)*Ac(i:i+mL-1,i+mL:i+m-1);
% Do LU on right half of matrix
err = RLUint(i+mL,mR);
end
end
return

```

To test this routine run the following two tests repeatedly for several values of n .
The first test is

```

A = randn(n,n); [L,U,P]=RLU(A); ...
[norm(triu(L-eye(n),0),1),norm(tril(U,-1),1),norm(P*P'-eye(n),1),norm(P*A-L*U,1)/(n*eps)]

```

The output should consist of 4 numbers: the first three should be exactly zero to confirm that L is unit lower triangular, U is upper triangular, and P is orthogonal (i.e. it contains one copy of each row of the identity matrix). The last number should $O(1)$, confirming that $\|PA - LU\|_1 = O(n\epsilon)$.

The second test is

```

A=randn(n,n); [Lr,Ur,Pr]=RLU(A); [L,U,P]=lu(A); ...
[norm(Lr-L,1)/(n*eps),norm(Ur-U,1)/(n*eps*norm(U,1)),norm(Pr-P,1)]

```

The output should consist of 3 numbers, comparing the output of RLU and the output of Matlab's lu. Barring ties in pivot choice that are decided differently, the output numbers should be $O(1)$ (confirming that the two L matrices agree to within $O(n\epsilon)\|L\|$), $O(1)$ (confirming that the two U matrices agree to within $O(n\epsilon)\|U\|$), and 0 (confirming that the identical pivots were chosen).

Question 3, Part 2. The goal is to prove that this algorithm does the same number of arithmetic operations as non-recursive GEPP (in fact, assuming an algorithm like Strassen is not used for the matrix multiplies or triangular solves, it does *the same* arithmetic operations, except for additions possibly getting done in a different order). To confirm this hypothesis we need to write down the number of arithmetic operations of conventional GEPP (implemented with 3 loops), but on a rectangular matrix with n rows and $m \leq n$ columns:

$$\begin{aligned}
 GEPP(n,m) &= \sum_{i=1}^m (n-i) && \dots \text{ divide by the pivot in column } i \\
 &+ \sum_{i=1}^m 2(m-i)(n-i) && \dots \text{ update the Schur complement} \\
 &= m^2(n - \frac{m}{3}) + O(nm)
 \end{aligned}$$

The recurrence for the number of arithmetic operations of the recursive routine applied to an n -by- m matrix (and assuming m is a power of 2 for simplicity) is

$$\begin{aligned} A(n, m) &= A(n, \frac{m}{2}) + A(n - \frac{m}{2}, \frac{m}{2}) && \dots \text{two recursive calls} \\ &+ (\frac{m}{2})^3 && \dots \text{compute U} \\ &+ 2(n - \frac{m}{2})(\frac{m}{2})^2 && \dots \text{update Schur complement} \end{aligned}$$

We need to confirm that $A(n, m) = GEPP(n, m)$ satisfies this recurrence, up to terms of $O(nm)$. In other words, we just need to confirm that

$$m^2(n - \frac{m}{3}) = (\frac{m}{2})^2(n - \frac{m}{3 \cdot 2}) + (\frac{m}{2})^2(n - \frac{m}{2} - \frac{m}{3 \cdot 2}) + (\frac{m}{2})^3 + 2(n - \frac{m}{2})(\frac{m}{2})^2$$

Question 3, Part 2. To keep it simple, we restrict ourselves to showing that there is a constant c_{LU} such that $c_{LU} \cdot nm^2/\sqrt{M}$ is an upper bound on the solution of a simplified recurrence for the number of words $W_{LU}(n, m)$ moved by a call to RLUint on a matrix of size n -by- m :

$$\begin{aligned} W_{LU}(n, m) &= W_{LU}(n, \frac{m}{2}) + W_{LU}(n - \frac{m}{2}, \frac{m}{2}) && \dots \text{two recursive calls to RLUint} \\ &+ c_{TR} \cdot (\frac{m}{2})^3/\sqrt{M} && \dots \text{words moved by call to triangular solve} \\ &+ c_{MM} \cdot (n - \frac{m}{2})(\frac{m}{2})^2/\sqrt{M} && \dots \text{words moved by call to matmul} \end{aligned}$$

where c_{TR} and c_{MM} are some constants. Now we try to pick c_{LU} so that $c_{LU} \cdot nm^2/\sqrt{M} \geq W_{LU}(n, m)$ by choosing it to satisfy

$$\begin{aligned} c_{LU} \cdot nm^2/\sqrt{M} &\geq c_{LU} \cdot n(\frac{m}{2})^2/\sqrt{M} + c_{LU} \cdot n(\frac{m}{2})^2/\sqrt{M} \\ &+ c_{TR} \cdot n(\frac{m}{2})^2/\sqrt{M} \\ &+ c_{MM} \cdot n(\frac{m}{2})^2/\sqrt{M} \\ &= (c_{LU} \cdot (\frac{1}{4} + \frac{1}{4}) + \frac{c_{TR}}{4} + \frac{c_{MM}}{4})nm^2/\sqrt{M} \end{aligned}$$

In other words, we need to pick c_{LU} to satisfy

$$c_{LU} \geq \frac{c_{LU}}{2} + \frac{c_{TR}}{4} + \frac{c_{MM}}{4}$$

Thus we may pick $c_{LU} = (c_{TR} + c_{MM})/2$.

Question 3, Extra Credit. For a recursive implementation of matmul, see the class notes or the reference for the answer to the extra credit question in last week's homework, which also describes how to do recursive triangular solve.