CS 290H: Sparse matrix algorithms // Homework 3

Assigned November 2, 2004

Due Sunday, November 21

1. [15 points] Suppose A is an $n \times n$ matrix, possibly unsymmetric, that has an LU factorization without any pivoting, $A = LU$. Consider the algorithm described in class lecture 10 (28 Oct), slide 6, but without the pivot step:

```
for j = 1 : n do
    solve [L1 0 ; L2 I] * [Uj ; Lj] = Aj;
    scale Lj by dividing by Ujj;
end;
```

where $L1$ is $(j-1) \times (j-1)$, $L2$ is $(n-j+1) \times (j-1)$, Aj is column j of A, and Uj and Lj are appropriate parts of column j of L and U. Make this algorithm precise (by specifying all the matrix dimensions, signs, and so on), and then prove that it does in fact compute L and U such that L is lower triangular with ones on the diagonal, U is upper triangular, and $A = LU$.

2. [20 points] Let A be a 7×7 matrix whose nonzero structure is

$$\begin{pmatrix} x & 0 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & 0 & 0 \\ 0 & x & x & 0 & x & 0 & 0 \\ x & 0 & 0 & 0 & 0 & x & 0 \\ 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & x & 0 & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & 0 & 0 & x \end{pmatrix}$$

For example, A might be generated by the Matlab statement

```
A = sparse([1 4 3 6 7 3 1 2 2 3 5 6 4 5 7],
           [1 1 2 2 2 3 4 4 5 5 5 5 6 6 7], 1, 7, 7);
```

2(a) [4 points] Draw the column intersection graph $G_\cap(A)$.

2(b) [4 points] Draw the filled column intersection graph $G_\cap^+(A)$.

2(c) [4 points] Draw the column elimination tree $T_\cap(A)$.

2(d) [4 points] Find a permutation vector p (i.e. a vector of length 7 that contains each of the integers 1 through 7 once) such that the matrix $A(p,:)$ has an entirely nonzero diagonal.

2(e) [4 points] Draw the directed graph $G(A(p,:))$.

3. [20 points]

3(a) [10 points] Let A be an arbitrary nonsingular matrix, and let $PA = LU$ be an LU factorization of A with partial pivoting. Prove that $u_{ij} \neq 0$ implies that (i, j) is an edge of the filled column intersection graph $G_\cap^+(A)$.

3(b) [10 points] Find a nonzero pattern that has the properties that (1) there exists at least one nonsingular matrix with that nonzero pattern and (2) for *every* nonsingular matrix A with that nonzero pattern, there exist some i and j such that (2a) $i < j$, (2b) (i, j) is an edge of the filled column intersection graph $G_\cap^+(A)$, and (2c) in the LU factorization with partial pivoting $PA = LU$, the element u_{ij} is zero. (In other words, for any particular matrix, the upper bound on the structure of U in the previous problem may not be strict.)

[Optional, extra credit] Let A be an arbitrary nonsingular matrix.

(a) Show that the LU factorization of A with partial pivoting, $PA = LU$, can be rewritten as $A = P_1 L_1 P_2 L_2 \cdots P_n L_n U$, where each L_j is a "Gauss transform" (that is, a lower triangular matrix with ones on the diagonal whose only off-diagonal nonzeros are in column j), and each P_j is a transposition (that is, a permutation that interchanges two rows and leaves all the other rows fixed).

(b) Let

$$\bar{L} = I + \sum_j (L_j - I),$$

where I is the $n \times n$ identity matrix. (Thus $\bar{L}$ is a lower triangular matrix each of whose columns has the nonzeros from one of the L_j.) Prove that the collection of nonzero values in $\bar{L}$ is the same as in L, though they may occur in different locations.

(c) Prove that $\bar{l}_{ij} \neq 0$ implies that (i, j) is an edge of the filled column intersection graph $G_\cap^+(A)$.

4. [45 points] Write a Matlab mexFunction `[L, U] = lunopiv (A)` that factors a square sparse matrix A as L times U, where L is lower triangular with unit diagonal and U is upper triangular. You may assume that A does have an LU factorization – if a zero is encountered on the diagonal, you should go ahead and divide by it, producing a factorization that contains Inf values.

Given any matrix A in which partial pivoting does not interchange rows (that is, a matrix for which Matlab's `[L, U, P] = lu(A)` returns P equal to the identity matrix—for example, this happens if A is diagonally dominant), the result of your code should be the same as the first two outputs of Matlab's `lu`, at least to within roughly machine epsilon times the largest element in L, U, or A. One way to verify that you're getting the right answer is to use your routine to solve a linear system $Ax = b$ for a random right-hand side b. To do this, you may either use Matlab's backslash operator to solve the sparse triangular systems $Ly = b$ and $Ux = y$, or you may use your own triangular solve routine to do this. It is a good idea to verify that `norm(A*x-b)` is small just to make sure everything is working.

You should use a left-looking column algorithm like the one we've discussed in class. You will probably build on your sparse triangular solve code from Homework 2. As in Homework 2, you may use Matlab to help get the indices of each column of the output matrices into sorted order by wrapping your routine in a Matlab m-file that transposes your results (twice) to sort them. Your algorithm should be pretty much the same as GP without pivoting, using a SPA and depth-first search to do the sparse-sparse triangular solve at each column. You may use your choice of data structures for the SPA.

To test your code, generate random diagonally dominant matrices A by

```
A = sprand(n, n, d/n);
A = A + diag(sum(abs(A))) + speye(n);
```

for values of $d = 1$ and $d = 5$, and n as large as you can. You should try your code both on A as generated above, and on the result of permuting A symmetrically by minimum degree,

```
p = symamd(A | A');
A = A(p, p);
```

(Symmetric minimum degree is appropriate, even though the matrix is nonsymmetric, because you're not pivoting.) You should verify that your code is giving the right answer as described above, but for your timing results you should only measure the factorization time, not the time to generate A or to do the triangular solves.

Turn in all your code, and a plot that shows both your code's run time and the run time of Matlab's `lu`, and also a Matlab transcript of the session that creates the plot and verifies the output.

I won't give extra credit for the fastest code this time, since different codes will prefer different matrices, but I will take performance into account in the grade.

[**Optional, extra credit**] Add partial pivoting to your code: keep track of an integer permutation vector, and return it as a third output. Thus your code should be callable as

```
[L, U, p] = lupiv(A);
```

and the result should be that $A(p, :) = LU$. Now your code should work for any square nonsingular matrix. Try your code on matrices of the form

```
A = sprand(grid5(k));
A = A(:, colamd(A));
```

for k as large as you can. (This time you use column minimum degree because you are doing partial pivoting.) Compare your results to Matlab's `[L, U, P] = lu(A)`. Your L and U should be (close to) the same as Matlab's. Matlab returns a sparse permutation matrix instead of a permutation vector, but you ought to be able to identify your p in the result of `[i, j, a] = find(P)` for Matlab's P.

[**Optional, extra credit**] Install Tim Davis's Matlab/Java interface to the Florida Sparse Matrix Collection, from http://www.cise.ufl.edu/research/sparse/mat/, and experiment with your code with some of the matrices from the collection. If you've implemented partial pivoting (the first extra credit), you should be able to factor any of the square matrices. If not, you'll have to stick to diagonally dominant or symmetric positive definite ones, or else modify the diagonal elements so that pivoting isn't necessary.