CS 290N/219: Sparse matrix algorithms: Homework 3

Assigned October 19, 2009

Due by class Wednesday, October 28

1. [20 points] Let G be the graph of the n-vertex model problem, that is, a k-by-k grid graph with $n = k^2$ vertices. Prove that there is some constant $c > 0$ such that for *every* elimination ordering on G, the filled graph G^+ contains a complete subgraph with at least $c\sqrt{n}$ vertices. (A *complete subgraph* is a set of vertices such that every pair is joined by an edge.)

Hint: Suppose you're given an ordering for the vertices of G. Think of playing the graph game in the given order, and consider the first time that you've either marked all the vertices in any single row of the entire grid or else marked all the vertices in any single column.

2. [40 points] (see Davis problem 6.13). An *incomplete LU factorization* is an approximate factorization $A \approx LU$, in which L and U are lower and upper triangular matrices whose product is "approximately" A in some sense, but L and U have fewer nonzeros than the actual LU factors of A. It is useful as a preconditioner for iterative methods. There are many ways to define and compute a so-called ILU factorization; we'll study some of them in the second part of the course.

For this problem, you are to write a mex-file that implements the following version of ILU. I suggest that you start with Davis's `cs_lu` code, though can write your code from scratch if you prefer. Your ILU routine should take two additional parameters `droptol` and `diagtol`, both of which are real (`double`) numbers in the range 0 to 1. The parameter `droptol` controls how much fill is discarded during the factorization, and thus how sparse the approximate factors are; the parameter `diagtol` controls how close to zero the diagonal elements of U are allowed to be.

The ILU code will compute one column of the factors at a time, just like `cs_lu`, with three differences:

- There is no partial pivoting, that is, no exchanging rows.
- After each column of L and U has been computed (but before the column of L is scaled by the diagonal element u_{jj}), let η be the norm of that computed column. All fill entries in that column that are smaller in magnitude than the "local drop tolerance," which is $\texttt{droptol} \cdot \eta$, are "dropped" from L or U. Notice that only fill entries are dropped—nonzeros of L and U that correspond to nonzeros of the original matrix A are kept no matter how small they are. The other exception to the dropping rule is the diagonal element u_{jj} of U, which is also kept even if it is too small.
- If $|u_{jj}| < \texttt{diagtol} \cdot \eta$, then u_{jj} is replaced by $\pm(\texttt{diagtol})^{1/2} \cdot \eta$. Like the dropping, this replacement happens before the column of L is scaled by u_{jj}.

Your code should also return the number of diagonal elements that were adjusted because of `diagtol`.

Setting `droptol = diagtol = 0` produces the complete factorization $A = LU$ without pivoting, if it exists. Setting `droptol = 1` produces what is sometimes called an "ILU0" factorization, in which all fill elements are dropped.

Verify and experiment with your code on various matrices in Matlab. You can compare your code to Matlab's `luinc()` function, but notice that they don't do quite the same thing. I will post some suggestions of matrices to experiment with as well.

You can use your ILU code for preconditioning experiments in the second part of the course.