CS 290N / CS 219: Sparse Matrix Algorithms // Homework 1

Due by class time Wednesday, October 7

Please turn in all your homework on paper, either in class or in the homework box in the Computer Science Department office. For programs, turn in (1) code listing; (2) sample output and any plots, showing thorough testing; (3) a transcript of a Matlab session showing how it's used.

Problem 1. **[10 points]** Let A and B be two n-by-n matrices. Show that the number of nonzero scalar multiplications required to compute $C = AB$ is (using Matlab notation)

$$\sum_{i=1}^{n} \texttt{nnz}(\texttt{A}(:,\texttt{i})) * \texttt{nnz}(\texttt{B}(\texttt{i},:)).$$

Problem 2. **[30 points]** Let G be the 9-vertex "grid graph" corresponding to the two-dimensional model problem on a 3-by-3 mesh. Suppose the vertices of G are numbered 1 through 9 by rows from upper left to bottom right.

2(a) **[5 points]** Draw the sequence of ten so-called "elimination graphs" (beginning with G and ending with the empty graph) that result from playing the vertex elimination game on G with the given vertex numbering.

2(b) **[5 points]** Draw the 9-vertex "filled graph" G^+ consisting of G plus fill edges.

2(c) **[10 points]** In Matlab, create a 9-by-9 symmetric, positive definite matrix A that has G as its nonzero structure (that is, such that $G(A) = G$). Use the Matlab functions `spy()` and `chol()` to visualize the nonzero structure of A and of its Cholesky factor, verifying that the nonzero structure of the Cholesky factor is $G^+(A)$. Print your `spy` plots and turn them in with your homework.

2(d) **[10 points]** Find a way to renumber the vertices so that there is less fill. That is, find a permutation p such that $G^+(A(p,p))$ has fewer edges than $G^+(A)$. Draw the filled graph $G^+(A(p,p))$, and the print the `spy` plots of `A(p,p)` and its Cholesky factor.

Problem 3. **[30 points]** You may do the programming assignments for this course in C or Fortran; I recommend C. In each case, you will set your code up with an interface so that it can be called from Matlab as a "mexFunction". This will let you use Matlab to debug your code, and it will let me use Matlab to test it. This warmup assignment is just for you to learn how to write a Matlab interface using sparse matrices.

Write a Matlab mexFunction, `sparseprint(A)`, which prints the contents of a sparse matrix `A` as a list of nonzero entries like "entry (7,8): 13.4".

Your routine can assume the matrix is real, but should not assume that it's square. Test it from Matlab with several matrices you make up. See the Matlab functions `sprand` or `sprandn` to generate random sparse test matrices. Or download the Matlab interface to the UF Sparse Matrix Collection, at http://www.cise.ufl.edu/research/sparse/matrices.

For documentation on mexFunctions, open up a help window in Matlab, and look in the *External Interfaces/API* section, under the *Matlab* section. The C mexFunction syntax is quite a bit nicer than the Fortran. All mexFunctions have the routine name `mexFunction`.

You'll need to use the following Matlab "mx" and "mex" routines:

`mxGetM`	returns the number of rows of a matrix
`mxGetN`	returns the number of columns of a matrix
`mxGetJc`	returns a pointer to the column pointer array (`Ap`)
`mxGetIr`	returns a pointer to the row indices (`Ai`)
`mxGetPr`	returns a pointer to the numerical values (`Ax`)

There is a more complex but very well-written example of mexFunctions for sparse matrices in Tim Davis's LDL code, which is at http://www.cise.ufl.edu/research/sparse/ldl. LDL computes a sparse Cholesky factorization; you don't need to understand how that works (yet), but you can see from that code how a C program can get access to a Matlab sparse matrix.

I recommend the following strategy for writing mexFunctions. Suppose the name of the mexFunction is `foobar`. Put all Matlab-related work in a file called `foobarmex.c` (or `foobarmex.f` for Fortran). The routine name in this file is `mexFunction`. All calls to `mx*` and `mex*` routines should be placed here. This routine then calls a routine in the file `foobar.c` or `foobar.f` that does the actual work. There should be no references to `mx*` or `mex*` routines in the `foobar.c` file, with the exception of calls to Matlab functions that have ANSI C counterparts (such as `malloc`, `free`, and `printf`).

Why bother? Because this makes the `foobar.c` file a stand-alone routine that can then be used in a stand-alone C program (or Fortran, as the case may be), without using Matlab. OK, for such a simple routine as `sparseprint`, this structure is overkill. But it's a useful habit anyway. Tim's `LDL` program follows this structure.