Mathematics 372 – Numerical Linear Algebra
Lab2/Problem Set 3 – *due:* Friday, February 9

General Information

In this lab/problem set we will study the *Cholesky* decomposition of positive-definite symmetric matrices, using MATLAB. A particular item of interest in one of the later questions will be the relation of the "sparsity pattern" of A (that is, the locations of zero and nonzero entries in the matrix) and the sparsity pattern of the Cholesky factor R.

MATLAB Practice

The command for computing Cholesky factorizations in MATLAB is `chol` (logically enough!). To use it, you can enter the matrix A first as we discussed before, then enter a command of the form

```
chol(A)
```

The output will be the Cholesky factor R. For example enter the matrix

$$A = \begin{pmatrix} 16 & 4 & 8 & 4 \\ 4 & 10 & 8 & 4 \\ 8 & 8 & 12 & 10 \\ 4 & 4 & 10 & 12 \end{pmatrix}, \tag{1}$$

then compute the Cholesky factor and call the result R. Of course we should have $A = R^t R$ (the Cholesky factor is the "matrix square root"). To check that the result is correct, you can compute $R^t R$ in MATLAB with

```
R' * R
```

(the "prime" is transpose in MATLAB, $*$ is matrix multiplication as we saw before). Try it and compare with the original matrix A. The matrix (1) is simpler than many because the Cholesky factor is also an integer matrix. That is not always true since as we know the computation of the entries in R involves taking square roots. To see a more typical example, let's add a 4 identity matrix I to this A:

```
AA = A + eye(4)
```

(someone had a sense of humor here!). Then compute the Cholesky factor and check that it is correct. Call your computed Cholesky factor RR.

The Cholesky factorization is only defined for positive definite symmetric matrices. If you enter a matrix that is symmetric but not positive definite, then MATLAB will produce an error message. For instance try `chol(B)` for

$$B = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

B is not positive definite (why not?)

Something more subtle happens if you enter a matrix that is not symmetric. In that case, MATLAB doesn't actually check to see whether the matrix is symmetric, it just *assumes that it is* and works with the entries above the diagonal as in the formulas for Cholesky factorization that we wrote down in class. For instance, try

$$T = \begin{pmatrix} 12 & 1 \\ 2 & 11 \end{pmatrix}$$

and compute `chol(T)`. If the output is R, what matrix is $R' * R$ here?

Recall that we got into the whole "matrix factorization game" and the idea of the Cholesky factorization by way of the observation that solving systems of linear equations for upper- or lower-triangular coefficient matrices is *especially simple* (no row-reductions are necessary if we use backward or forward substitution). For instance suppose we want to solve $Ax = b$ for some b. Since $A = R^t R$, as we know, this can be done in two steps. By associativity of matrix multiplication, $R^t(Rx) = b$ First we solve the system $R^t y = b$, then we solve $Rx = y$ for x. Carry out this process in MATLAB for the system $AAx = b$ with AA as above, and a vector b of your choice. Check your final answer.

Finally, for some of the problems below, we will need to introduce the idea of *sparse* matrices. A somewhat imprecise, but suggestive, way to understand this idea is that a sparse matrix is one where some "large" proportion of the total entries are *zeroes*. One could also quantify this, of course, by computing the actual proportion of zero entries in the matrix. Matrices with "almost all" entries nonzero are called "dense" matrices. MATLAB has an interesting command called `spy` that produces a *plot* indicating the sparsity structure of a matrix. For instance, enter

```
A = [1 0 -1; 2 3 0; -1 -1 2]
spy(A)
```

What does the plot mean? Of course this is more interesting for "big" matrices (where we might not even want to try to list all the entries!) For example, there are a number of "builtin" big example matrices in MATLAB, including one 479×479 matrix called `west0479`. To access it and generate the spy plot, enter

```
load west0479
spy(west0479)
```

Even though we don't want to try to see the whole 479×479 matrix, we can visualize where it has nonzero entries and see some things about it. For instance, is `west0479` a symmetric matrix? The number $nnz = 1887$ at the bottom of the plot is the actual number of *nonzero* entries. What fraction of the total entries in this matrix are zero?

If a matrix is very sparse in this intuitive sense, then storing the whole matrix (with all the zeroes explicitly filled in) is an inefficient use of the computer memory (especially if the matrix is *big*, say 10000×10000). So an alternate way to think about storing the information in a matrix is to give the total numbers of rows and columns, then *list only the*

nonzero entries, together with their row and column numbers. If a pair of a row number and a column number *doesn't* appear, then that entry is automatically *zero*. A matrix stored in this way is properly called a matrix with a *sparse representation*, but we will also often refer to it as a *sparse matrix* (even if the proportion of zero entries is not that large). A moment's thought will convince you that it is possible to compute matrix sums, products, etc. using this sort of representation of matrices (though it would be awkward by hand!). And MATLAB has two different ways of dealing with matrices:

- as *dense* matrices (all the matrices we have entered manually to this point have been dense), or
- as *sparse* (i.e. sparsely represented) matrices (`west0479` above is a sparse matrix). For instance try entering the command

```
west0479
```

Two commands `sparse` and `dense` convert from one representation to the other (and are inverse functions of one another!). With A the 3×3 matrix above

```
AS = sparse(A)
```

The output is the sparse representation. Note that there are exactly seven items in the list (each item being a pair of a row and column number, together with the corresponding entry in A). The command `dense(AS)` returns to the dense representation.

The MATLAB matrix commands we have seen to this point are set up so that either dense or sparse matrices can be supplied as the input (*as long as the same form is used for all inputs!*), and the output will be a matrix of the same type. Some other commands in the future will *only* take sparse matrix inputs.

Problems

Note: Several problems below deal with rather large matrices. You probably won't want to print them out, either in dense or in sparse representations! Recall that a semicolon at the end of a MATLAB statement does the calculation but suppresses the output.

I. Recall the systems of linear equations we obtained by discretizing the 2nd order ODE boundary value problem in Lab 1. If in the ODE

$$ay'' + by' + cy = f,$$

the constants are $a = -1$, $b = 0$, $c = 1$ (as before), then the coefficient matrix A_N is positive definite for all N.

A) Use MATLAB to compute the Cholesky factors R_N for the coefficient matrices of these systems for $N = 6, 8, 20$. Show how you could recover the approximate solutions of the boundary value problem you had in the first lab using the Cholesky factors R_N.
B) What do you notice about the locations of nonzero entries in R_N in these examples?

C) (non-lab problem) An $n \times n$ matrix $A = (a_{ij})$ is said to be *tridiagonal* if $a_{ij} = 0$ for all entries with $j > i + 1$ and also $i > j + 1$ (fancier way to say the same thing: $a_{ij} = 0$ whenever $|i - j| > 1$). The entries in the "band" consisting of the main diagonal and the upper and lower diagonals on either side of it may or may not be zero. Give a full mathematical proof using the formulas we developed for computing the Cholesky factor that if A is tridiagonal, then $R = (r_{ij})$ is an upper triangular matrix with $r_{ij} = 0$ whenver $j > i + 1$. In other words R is upper triangular *and also* tridiagonal. Suggestion: To see the ideas you need for this proof, it will probably help to work out one or two 4×4 or 5×5 examples by hand just to see which entries are actually used where in the computation. The matrix

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix}$$

should be instructive. Why do the zero entries in R end up being zero?

II. The "bucky ball" is a polyhedron with 60 vertices, 32 faces (12 pentagons and 20 hexagons), and 30 edges. If you imagine taking one and "inflating" it so the surface becomes spherical, you will see the edges forming the pattern of seams on a *soccer ball.* This shape also appears in chemistry. There is a carbon-hydrogen chemical compound called "fullerene" where carbon atoms are arranged at the vertices of one of these polyhedra, and the carbon-carbon bonds are the edges. To see a picture, look up the Wikipedia entry for "Fullerene". The names "bucky ball" and "Fullerene" refer to the architect and futurist Buckminster Fuller who proposed constructing buildings called geodesic domes with these shapes.

We can get a *matrix* from this shape in the following way. First we label the vertices $1 \ldots 60$ in some fixed way. Then we make a 60×60 matrix $B = (b_{ij})$ where

$$b_{ij} = \begin{cases} 1 & \text{if there is an edge vertex } i \text{ to vertex } j \\ 0 & \text{if there is no edge} \end{cases}$$

(There are no edges from vertex i to vertex i for any i.) This gives a pretty sparse 60×60 matrix which is supplied in MATLAB under the name `bucky` (you don't `load` this one, it's a predeclared name in MATLAB). Note: This example is also discussed in the text, but try to figure this out on your own before you look there to check your work.

A) (non-lab) From the picture of fullerene in Wikipedia above, or from your experience of soccer balls, (or if absolutely necessary by examining the entries in `bucky` directly), how many nonzero entries are there on each column or row of B? Why?

B) (non-lab) Prove that B is *not positive definite* (no matter how we label the vertices).

C) To make a positive definite matrix, we can add a positive multiple of the identity matrix I_{60}. Since `bucky` is a MATLAB *sparse* matrix, we want a *sparse* identity matrix. Compute the Cholesky factors for the matrices

$$B + 4I_{60}, \quad B + 10I_{60}, \quad B + 20I_{60}$$

and examine their `spy` plots. Does the spy plot of the Cholesky factor of $B + cI_{60}$ appear to depend on the value of c?

III. We will consider the relation between the spy plots of Cholesky factors R and the spy plots of the original positive definite matrices $A = R^t R$ in this question.

The *envelope* of a symmetric matrix A or an upper triangular matrix U is a set of ordered pairs (k, ℓ) with $\ell \geq k$ representing the locations of the *nonzero* entries in the upper triangle of the matrix. It is defined like this: For each column $j = 1, \ldots, n$ we locate the *first nonzero entry in that column* (counting down from the top of the matrix). Say this happens in location (m, j). Then the rest of the pairs representing locations in that column down to the diagonal go into the envelope: $(m-1, j), (m-2, j), \ldots, (j-1, j)$.

A) On a printout spy plot of the bucky matrix B from problem II, indicate (by hand) where the envelope of B is. Do the same on the spy plot of the Cholesky factor of $B + 4I_{60}$.
B) (non-lab) The tridiagonal examples in question I and the example from part A should make the following theorem plausible.

Theorem. *If A is positive definite symmetric and R is its Cholesky factor, then A and R have the same envelope.*

There is a proof of this in the text in Theorem 1.5.7 (page 57-58). It depends on the "bordered form" of Cholesky factorization on page 43. Your job in this part is to work through the details of that proof, and write out the complete argument in your own words. In the course of this, you will want to supply a proof for Exercise 1.4.35 in the text.