Operation Counts for Gaussian Elimination
Introduction to Conditioning of Linear Systems

Problem 3 is to be handed in, written up clearly on separate sheets, on Wednesday 10/22.

Here are some Matlab commands that you will be using in this lab.

Random matrices and vectors: `randn(4)` returns a 4×4 matrix with random entries. `randn(5,1)` returns a vector of length 5 (i.e., a 5×1 matrix) with random entries.

LU decomposition: `[L,U]=lu(A)` does the LU decomposition of A. It uses pivoting so L will be a unit lower triangular matrix with the rows switched around and U an upper triangular matrix. If A is an $n \times n$ matrix and b a vector of length n, then `A\b` returns the solution to $Ax = b$ computed using the LU decomposition.

Cholesky decomposition: if A is a positive definite matrix then `U=chol(A)` is the upper triangular matrix with $U^T U$ (or `U'*U` in Matlab notation) equal to A. Try it.

Counting operations: `flops` returns the number of floating point operations used (each addition, subtraction, multiplication, and division counts as one operation). `flops(0)` resets the counter to 0, so, for example, "`flops(0); lu(A); ops=flops`" would set `ops` to the number of operations needed to compute the LU decomposition of A.

Plotting symbols: If you want to plot some points, but not connect the points with line segments, you can specify a symbol like `'o'` or `'*'`. For example, the commands `n=1:5; plot(n,n.^2,'o',n,sqrt(n),'*')` will plot the squares of the numbers 1 to 5 with o's and their square-roots with stars. (Try it.)

Vector norms: If x is a vector, the norm of x, which is usually denoted $\|x\|$, is its length: $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$. Matlab has the command `norm(x)` to compute this.

1. We showed in class that computing doing Gaussian elimination of an $n \times n$ matrix, or, what is the same thing, computing its LU decomposition, requires $n^3/3 + O(n^2)$ additions and multiplications. In this exercise you are to check that Matlab's routine does indeed cost about $2n^3/3$ operations (counting both additions and multiplications).

a) As a warm-up create a random 50×50 matrix and count how many flops to do its LU decomposition. Compare the result to $\frac{2}{3}50^3$.

b) Now make a loop `for i=1:10`, and inside the loop set n equal to $10i$ and compute and store the number of flops to do an LU decomposition of a random $n \times n$ matrix (only count the flops in the LU decomposition, not in creating the random matrix). On a single plot, using o's and stars, plot the number of flops and the estimate $\frac{2}{3}n^3$ of the number of flops, both as functions of n.

2. a) Let's try Matlab's Cholesky decomposition routine. First make a 4×4 positive definite matrix. You can do this with `A=randn(4); A=A*A'` (because $A * A^T$ is symmetric

positive definite for any matrix A). Do `U=chol(A)` to get the upper triangular Cholesky factor. Check that $U^T U = A$.

b) As in part b of the previous problem, plot the number of flops needed to do the Cholesky decomposition for a matrix of size n compares to the predicted value of $n^3/3$ (counting additions and multiplications).

3. In this problem we will explore the sensitivity, or condition, of the problem of solving $Ax = b$. That is, we will consider how making a small changes in b affects the solution x. If c is an approximation of b, we can measure the absolute error with $\|b - c\|$ and the relative error with $\|b - c\|/\|b\|$.

a) Create a random 4×4 matrix A and a random vector b of length 4. Create another vector c near b by adding a small random vector to b: `c=b+0.01*randn(4,1)`. (This should give you a c which differs from b by roughly 1%.) For your b and c what is the absolute error and relative error? Now solve $Ax = b$ with the \ operation. Also solve $Ay = c$. What are the absolute and relative errors in y as an approximation of x? Do this experiment 5 times, with different random matrices each time, and give a table of results. (There should be 4 columns in your table: the absolute and relative error in b and the absolute and relative error in the corresponding x.) Briefly describe what you find. Are the relative errors in x much larger, much smaller, or about the same size as those in b?

b) Do the same experiment but with much smallers errors, say a random change to b with a factor of 0.00001 instead of 0.01. Again make a table and describe what you see.

c) Do the same experiment, but with much larger matrices, say `A=100*randn(4)`. Again make a table and describe what you see.

d) There are matrices A for which the solution of $Ax = b$ can be very sensitive to changes in A. These are matrices that are in some sense nearly singular, so solving $Ax = b$ is nearly undefined. A simple example is the positive definite matrix

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}.$$

For this matrix and $b = (32, 23, 33, 31)^T$, what is the (exact) solution of $Ax = b$? Now approximate b with $(32.1, 22.9, 33.1, 30.9)^T$. Then what is the exact solution? What are the absolute and relative errors in b and in the corresponding x? Try adding random errors of various sizes to b, compare the sizes of relative errors in b to those of the relative errors in x that they cause. **Note**: the above phenomena is due to the matrix A not the particular method used to solve $Ax = b$.

e) What is the determinant of the matrix A in part d? (If you don't know the determinant command in Matlab, do `lookfor determinant`.)