Assignment #2, due October 13th, 2017, IN CLASS
AMATH 740, CS 770, CM 750
Fall 2017

Reading:
C. Moler "Numerical Computing with Matlab" (online, free download), especially Ch. 2
Quarteroni, Sacco, Saleri "Numerical Mathematics", Ch. 3, electronic copy at the library.
Trefethen and Bau "Numerical Linear Algebra" (a useful reference on the subject)

1. Implement LU factorization with partial pivoting. Test your code on a simple problem and provide a print-out of your session. Explain how you implemented row interchanges.

2. Consider matrix
$$A = \begin{bmatrix} 0.001 & 2.000 & 3.000 \\ -1.000 & 3.712 & 4.623 \\ -2.000 & 1.072 & 5.643 \end{bmatrix}$$
   (a) Write out the detailed process of LU factorization in a computer with four digit mantissa without pivoting. In particular, give M_1, M_2, etc.
   (b) Compare with a double precision LU factorization (obtained, e.g. from your code above)
   (c) Discuss reasons for loss of accuracy in part (a).

3. Compute operation count of the partial and full pivoting to the leading order. An operation here is a comparison of two numbers.

4. Let $A = (n, n)$ be nonsingular. Show that A has an LU factorization if and only if each upper left block $A_{1:k,1:k}$ is nonsingular. Hint: Show first that the row operations of Gaussian elimination leave the determinants $A_{1:k,1:k}$ unchanged.

5. (a) Describe an algorithm for computing A^{-1} by solving n systems of equations, where n is the dimension of A.
   (b) Calculate the cost of this algorithm and argue that it is preferably not be used
   (c) Propose a modification of the algorithm that takes advantage of sparsity

6. Read about the tridiagonal algorithm in e.g. Quarteroni, Sacco, Saleri "Numerical Mathematics", Ch. 3.7.1. Such matrices often arise in numerical solution of PDEs, where a system $Ax = b$ is solved many times with different right hand sides. Argue why computing the inverse matrix is a particularly bad idea in this case.

7. Show that the bound on growth factor $\rho = \frac{max_{i,j}|u_{ij}|}{max_{i,j}|a_{ij}|}$ is given by 2^{n-1}, where n is the dimension of the matrix.

8. (bonus) Time Matlab *lu* command and your own code on a random matrix of sufficiently large size (say, n=1000) using commands tic and toc. Which one is faster? Estimate how long LU factorization should take based only on the number of algebraic operations and the clock speed of your computer. Is the measured run time about your estimate? Next, run LU factorization on a sequence of random matrices, say n=100, n=200, n=400, etc. Do run times scale according to theoretical predictions? If no, then why not?