EAS 596, Fall 2018, Homework 5
Due Weds. 10/03, **5 PM**, Box outside Furnas 611

Work all problems. Only MATLAB code will be accepted. Show all work, including any M-files you have written or adapted. Make sure your work is clear and readable - if the TA cannot read what you've written, he will not grade it. All electronic work (m-files, etc.) **must** be submitted through UBLearns and submitted by the due time. Any handwritten work may be submitted by the due time. Each problem will be graded according to the following scheme: 2 points if the solution is complete and correct, 1 point if the solution is incorrect or incomplete but was using correct ideas, and 0 points if using incorrect ideas.

1. This problem explores the trade-off between the truncation error in the finite difference approximation of the derivative and floating point truncation error incurred when using finite precision arithmetic. Using a simple Taylor series, it is easy to construct a *forward difference* approximation of the derivative:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

where h is some defined step-size.

   (a) Write a MATLAB script that computes the forward finite difference approximation of the derivative of $\sin(x)$ for $x = \pi/3$. Use step sizes, h, ranging from 10^{-1} to 10^{-16}. You may find the `logspace` command useful. Plot the error, using a log-log scale, between your approximation and the exact derivative for each of your step sizes. Make sure to upload your script to UBLearns.

   (b) Based on your plot, what is the best step size to use for the finite difference step size?

   (c) Explain the behavior you see in the graph.

2. Write the following set of equations in matrix form:

$$\begin{aligned} 50 &= 5x_3 - 7x_2 \\ 4x_2 + 7x_3 + 30 &= 0 \\ x_1 - 7x_3 &= 40 - 3x_2 + 5x_1 \end{aligned}$$

   Solve the resulting system of equations, by hand, using the LU decomposition. Check your answer in MATLAB (you do not need to submit checking your answer in MATLAB).

3. Write your own MATLAB function that accepts a matrix A and computes the LU decomposition of A (without partial pivoting). Your function should return L and U. Write a MATLAB script that uses this function to solve the linear system from the previous problem to verify it's functioning correctly. Make sure to upload your function and script to UBLearns.

4. Write a function which produces the time needed to compute the LU factorization for random square matrices from a size $n = 10$ to $n = 1000$ using the function written in Problem 3 (you may find the functions "tic" and "toc" useful). Use the following MATLAB command to generate the random matrix (this ensures its strictly diagonally dominant and does not require partial pivoting): `A = rand(n,n)+n*eye(n)`, where n is the size of the linear system. Plot the time required versus the matrix size on a log-log plot. Include a line showing the long-term trend of the time required. Does the trend hold for small n? Why or why not? Be sure to *only* include the time required for the LU decomposition, and not any extraneous functions, such as the generation of the random matrix. Make sure to upload your script to UBLearns.

5. This problem illustrates how Gaussian elimination can be unstable without pivoting. Take the following matrix:

$$\mathbf{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}$$

   (a) Compute the condition number of this matrix in MATLAB using the `cond` command.

   (b) Compute the LU factorization of this matrix, by hand and *without* partial pivoting, using *exact* arithmetic.

   (c) Compute the solution to $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{b} = [1, 0]^T$, again with *exact* arithmetic.

   (d) Now repeat the factorization and solution computation using *finite-precision* arithmetic.

   (e) What will the solution be once partial pivoting is enabled using *finite-precision* arithmetic?