MACM 316 – Computing Assignment #3

Due Date: Monday October 23 at 11:00pm.

Instructions: Upload to Crowdmark a single PDF file that consists of only two pages: page #1 is your report (containing all discussions, data and figures), and page #2 is a listing of your Matlab code. The assignment is due at **11:00pm sharp!** If Crowdmark indicates that your submission was late, then you will be assigned a grade of zero, with no exceptions – so don't leave submitting until the last minute. When this CA is released, you should receive an e-mail containing a link for uploading your assignment, so make sure you save this message.

- Carefully review the **"Guidelines for Computing Assignments"** posted on Canvas.
- If you have questions about this assignment then you can obtain help in the computational workshops, tutorials, or the "Computing Assignment" discussion group on Canvas.

CA3 – Gaussian Elimination for Random Matrices

In this computing assignment, you will study the behaviour of floating-point errors when the Gaussian elimination (GE) algorithm with partial pivoting (as implemented in Matlab's "backslash" command) is applied to random matrices of widely differing sizes. You have already seen in class that when solving systems with small matrices, the errors remain small. However, even for state-of-the-art implementations of GE, these errors can become significant when the matrix is very large. The purpose of CA3 is to answer the question: ***How large is "large"?***

To construct a random matrix system for which the exact solution (or at least a pretty good approximation of it) is known, we'll make use of a simple trick:

- Construct the $N \times N$ matrix A with random entries chosen from the interval $[-1, 1]$.
- Let $x = [1, 1, \ldots, 1]^T$ be an N-vector of ones – this is our exact solution.
- Compute the right hand side corresponding to any particular choice of A to be $b = Ax$.

Then, we compute the approximate solution $y = A \setminus b$ using Matlab's backslash operator. To measure the error between x and y, let

$$\delta = \max_{i=1,\ldots,N} |x_i - y_i|$$

which is the maximum component-wise difference between the two N-vectors[†].

Because A is a random matrix, no single A gives a good measure of the actual error in GE; instead, we must run this calculation for a number of trials with different realizations of A in order to get a sense for the representative error. For this purpose, let M be the number of trials and suppose that the solution error for the k^{th} trial is $\delta^{(k)}$. Then we can define the mean or average error as

$$E_N = \frac{1}{M}\left(\delta^{(1)} + \delta^{(2)} + \cdots + \delta^{(M)}\right)$$

which should be a better measure of error for an "average" random matrix A.

To help you get started, I have posted the simple code `GERandom.m` on Canvas, which performs the computations just described for given values of M and N. The code determines the mean error E_N over M random trials and then generates a scatter plot showing the error in each trial. Note that this plot uses a log scale for error (using Matlab's `semilogy` command) so that large changes in the error magnitude are easier to see. Experiment with this code and observe how increasing N leads to an increase in the mean error E_N.

[†]You'll see later in section 3b that this is just the vector max-norm or inf-norm, denoted $\delta = \|x - y\|_\infty$.

This leads us to the goal of CA3 which is ...

> ***GOAL:** To determine the size of the matrix N^* for which the mean error in Gaussian elimination reaches $E_{N^*} \approx 1$. In other words, how large does a random matrix A have to be before the round-off errors in GE are the same order of magnitude as the solution x? Or, when does relative error exceed 100%!*

In practice, your computer will not have the processing power to find N^* directly because these matrices are simply too huge. Instead, you should compute the mean error E_N for a sequence of more "moderate" sized N values and plot the data. You will then observe that your data points lie along a predictable curve, which you may then extrapolate to even larger values of N to estimate roughly where the curve hits the value $E_N \approx 1$. Here is a detailed list of what is required in your 1-page report:

(a) Modify the code `GERandom.m` to turn it into a function with this calling sequence

```
function EN = GERandom(N, M)
```

which should make your later calculations much easier. You can also remove any unneeded output or plotting statements.

(b) Use your modified code to experiment with different N and M, choosing values that are as large as possible without forcing your code to take too long to execute (limit yourself to 5–10 minutes of clock time at most). You need to strike a balance: you want your number of trials M to be large enough to get an accurate estimate of the average error, but you also want N as large as possible to obtain an accurate extrapolation. Carefully justify your final choice of N and M.

(c) Plot your E_N versus N for different values of N ranging between 5 and the maximum value you chose in (b). Use a log-log scale, which in Matlab can be done in two ways:

```
loglog(N, EN)     or     plot(log10(N), log10(EN))
```

where `N` and `EN` are vectors containing your computed data points.

(d) Observe that your data lies roughly along a straight line! This suggests seeking a linear function that fits your log-log data, or in other words to find constants p_1 and p_2 so that $\log_{10}(E_N) \approx p_1 \log_{10}(N) + p_2$. You can do this manually or else run the following Matlab command

```
p = polyfit(log10(N), log10(EN), 1)
```

which fits a polynomial of degree 1 to your log-log data, returning a 2-vector `p` that contains the coefficients of the linear fit[‡]. Add this line to your data plot and comment on how accurately it approximates the data.

(e) Estimate N^* by extrapolating your linear function to the point where it hits the value $E_N = 1$ (or $\log_{10} E_N = 0$).

[‡]You'll learn more in section 4c about the algorithm implemented in `polyfit` to perform this curve fitting.