Computer Assignment 1

Due Friday, October 4

You may work alone or in teams of two people. Each team must submit a single printed solution. Solutions must contain your *relevant* MATLAB input and output (do not include commands that didn't work), and text that indicates what your commands are doing and interprets your results. (You may find one of the following commands useful in preparing your solutions: `publish`, `notebook`, or `diary`; see MATLAB's online help for details.) Organization and clarity count.

.....

MATLAB is first and foremost a platform for doing linear algebra, and one of its most basic operators is the backslash ("\"), which solves linear systems. Generally speaking, to solve the system $A\mathbf{x} = \mathbf{b}$, type `x = A\b`. (Type `help slash` or `doc mldivide` for more information and other possibilities.) This gives essentially the same answer as typing `x = inv(A)*b` or `x = A^(-1)*b`, but using the backslash finds the answer more efficiently, without computing the inverse. In both cases, MATLAB uses Gaussian elimination, but with some variations that make the computation more efficient and accurate.

In your homework (problem 2.2.11) you'll learn why the number of arithmetic operations required to solve a system of n linear equations in n unknowns using Gaussian elimination is, for large n, roughly proportional to n^3. The goal of this problem is to see how true this is in practice with MATLAB, and also to see how much longer it takes to solve a linear system by inverting the matrix (`inv(A)*b`) versus the more direct approach (`A\b`).

The values of n you will be using are in the thousands, and it is neither feasible nor desirable to use a system of equations that you make up yourself. Instead, let MATLAB randomly choose a system to solve. You can get a random n by n matrix by typing `A = randn(n)`, and a random column vector by typing `b = randn(n,1)`. (Try this for a small value of n to see what the result looks like, but for large n it is best to follow these commands with a semicolon to suppress the output.)

There are two ways to measure (more accurately than using your watch) how long a computation takes to run in MATLAB: with the commands `tic` and `toc`, and with the command `cputime`. Each of the following command lines will display the amount of time taken to compute `A\b`:

```
tic; A\b; toc
start = cputime; A\b; cputime - start
```

The first command counts the actual time elapsed, while the second command counts the processor time used (which can be more than elapsed time if your computer has mutiple processors/cores). Neither method will give you the exact same answer each time you use it; use whichever method gives you more consistent answers from one try to the next.

(a) Find a value of n so that solving an n by n system using `A\b` takes consistently between 0.5 and 1 seconds on your computer. Since the timing that MATLAB reports can vary, see how long `A\b` takes for each of three consecutive runs, and do the same for `inv(A)*b`. Also the time taken may depend on the specific random matrix `A` and vector `b` you generated; try different `A` and `b` and see if the timing changes significantly. If so, you may need to average several results to get a consistent answer.

(b) Multiply n by $\sqrt[3]{2} \approx 1.26$ (then round to the nearest integer) and repeat part (a) on the same computer. Do this again and again, until n reaches four times its value in part (a). This will give you a total of seven values of n, including the one from part (a). According to the n^3 rule, each value of n should take about twice as long as the previous value, so be patient as n increases. If your computer runs out of memory, type `clear` or restart MATLAB and try again. If you can't avoid running out of memory, or if MATLAB takes more than a few minutes to solve the system, stop at that value of n.

(c) Organize your results into a table and discuss them. For each of the two solution methods you tried, how close did the average computation time come to doubling for each successive n? Does it grow faster or more slowly than n^3, and what do you think the reason for this is? How do the two solution methods compare? In particular, how much longer does `inv(A)*b` take than `A\b` when n is large?

(d) Is either solution method more accurate? Due to round-off error, a solution you get on a computer generally will not solve $A\mathbf{x} = \mathbf{b}$ exactly. For a reasonably large value of n, see how close $A\mathbf{x}$ is to $\mathbf{b}$ for `x = A\b` versus `x = inv(A)*b`. As before, the result may depend on the particular choice of A, so try more than one random choice of A and see whether you see a pattern.