AMSC/CMSC 662 Homework 2 , Fall 2011
Due: 9:30am Tuesday, September 27.

1. (10 points) In his PhD research at UMD, Sungwoo Park was working with block diagonal matrices (**BDmatrices**) where all of the diagonal blocks are square; for example,

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 8 & 8 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 8 & 2 & 3 & 0 \\ 0 & 0 & 0 & 4 & 5 & 7 & 6 & 0 \\ 0 & 0 & 0 & 7 & 9 & 1 & 8 & 0 \\ 0 & 0 & 0 & 2 & 1 & 8 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}.$$

When these matrices are very large, it is quite inefficient to store all of the zeros, so Sungwoo decided to use MATLAB's **cell arrays**. For example, the matrix $\boldsymbol{A}$ above is stored as

$$\boldsymbol{A}\{1\} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 8 & 8 & 8 \end{bmatrix}, \qquad \boldsymbol{A}\{2\} = \begin{bmatrix} 6 & 8 & 2 & 3 \\ 4 & 5 & 7 & 6 \\ 7 & 9 & 1 & 8 \\ 2 & 1 & 8 & 5 \end{bmatrix}, \qquad \boldsymbol{A}\{3\} = \begin{bmatrix} 9 \end{bmatrix}.$$

It would be nice to make working with such matrices as easy as working with the usual MATLAB matrices, and this can be done using **classes**, which are user-defined data structures along with rules for operating on them. To illustrate this **object oriented programming (OOP)** approach, we define a set of operations on data that is in BDmatrix format.

For example, we would like to be able to multiply two BDmatrices, $\boldsymbol{A}$ and $\boldsymbol{B}$, with compatible dimensions, just by writing `C = A * B` in MATLAB.[1]

Sungwoo didn't quite do that, but he wrote a MATLAB function that we will call `BDopr`, so that he could multiply the two matrices by typing
`C = BDopr('*', A, B)`, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are BDmatrices.

Similarly, we could compute solution of the linear system $\boldsymbol{AX} = \boldsymbol{B}$, where $\boldsymbol{A}$ is a BDmatrix and $\boldsymbol{X}$ and $\boldsymbol{B}$ are matrices with the same number of rows as $\boldsymbol{A}$ and $p \geq 1$ columns, by typing `X = BDopr('\',A,B)`.

Write and test your own (well-documented) version of `BDopr`.

Error checking: Use MATLAB's `error` function to report if dimensions are incompatible or if the input string does not equal '*' or '\'.

Note: If you think you need to compute any matrix inverses in order to solve the linear system, review the subject of linear system solving in a good numerical analysis book and learn the algorithm that Matlab uses for '\'. If you still have questions, please ask me or Tyler for help.

[1] We could do this using MATLAB's OOP tools, introduced at `http://www.mathworks.com/help/techdoc/matlab_oop/ug_intropage.html`, but we will not use these tools in this homework. We will take a much more elementary approach.

2. (10 points) Write a function in MATLAB

```
function [Roots,PossibleRoots] = AllSolve(f,a,b,L,tol)
```

to find all solutions $x \in [a, b]$ to the equation $f(x) = 0$, as we discussed in class. The function `f` should take a single variable `x` as its argument and should return the value $f(x)$. The variable L is a bound on the Lipschitz constant for f, so we know that for all $z, y \in [a, b]$,

$$|f(z) - f(y)| \leq L|z - y| .$$

Your algorithm should work as follows:

- Begin by making a **stack** with space for 100 entries. (Use MATLAB's `zeros` function to initialize a MATLAB array to hold it.)
- Push the single entry $[a, b, fa, fb]$ onto the stack, where $fa = f(a)$ and $fb = f(b)$.
- While the stack is not empty,
  - Pop the top entry off the stack to get values for c, d, fc, and fd.
  - If $|d - c| < tol$
    - If fc and fd have opposite signs, insert $[c, d]$ as the next row of `Roots`.
    - Else insert $[c, d]$ as the next row of `PossibleRoots`.
  - Else if the Lipschitz condition shows that it is possible that there is a point $x \in [c, d]$ satisfying $f(x) = 0$, then
    - Let $m = (c + d)/2$ be the midpoint of the interval.
    - If there is room on the stack,
      - Push $[c, m, fc, f(m)]$ onto the stack.
      - Push $[m, d, f(m), fd]$ onto the stack.
    - Else if there is no room on the stack, call `error` and quit.
- End while.

In writing your program, remember that evaluating $f(x)$ might be very expensive (several seconds or minutes), so don't use more function evaluations than necessary.

Submission instructions: Send Tyler email, subject `Hmwk 2`, with two plain-text attachments: `BDopr.m` and `AllSolve.m`.

He will grade your programs for documentation, clarity, and correctness of results on test problems that he has chosen.

Your programs should not display any information unless there is an error.