Name : _____

Homework Project #3

Math 365, Fall 2015

Due Wednesday, October 9th

You are encouraged (but not required) to work together in teams of two on this assignment. Please turn in only one homework per team, and be sure that both names are on the assignment.

1. Consider the following system of equations :

$$u_{j-1} - 2u_j + u_{j+1} = -4h^2\pi^2 \sin(2\pi x_j), \qquad j = 1, \ldots N-1$$

where $x_j = jh$ and $h = 1/N$ and $u_0 = u_N = 0$.

   (a) Represent this linear system as a *tridiagonal* system. Use `spdiags` to create the tridiagonal matrix and use the backslash operator to solve the system for the unknown values u_j. Plot the solution for $N = 100$. Add axes labels and a title.

   (b) On the same graph, plot the exact solution $u_{exact}(x) = \sin(2\pi x)$. This problem solves the differential equation $u''(x) = -4\pi^2 \sin(2\pi x)$ using a *finite difference approximation* to $u''(x)$.

   (c) Use **condest** to estimate the condition number of the resulting linear system for the range of values $N = 100$ to $N = 5 \times 10^5$ (use `logspace`). (Note : Because we are storing the matrix in sparse format, N can be much larger than what you computed in your last homework!). What happens to the conditioning of this system as N gets large? Plot a `loglog` plot of the condition number as a function of N. On the same graph, plot N^2 verses N. What does this second plot say about the scaling of the condition number with N?

   (d) For each N in the range of values you used in part 1c, compute an error as

$$e_N = \frac{\| u - u_{exact} \|_\infty}{\| u_{exact} \|_\infty}$$

   using the `norm` function with a second argument of `Inf`. Plot this error on a `loglog` plot as a function of N. On the same plot, also include a plot of N^{-2}, $\mathbf{u}\kappa(A)$ and $\mathbf{u}N^2$, where $\mathbf{u}$ is machine epsilon computed using the Matlab keyword `eps`, and $\kappa(A)$ is the condition number of A. What can you say about scaling of the error and the condition number? Describe a potential downside to solving $u''(x) = f(x)$ using a finite difference approximation.

2. (**Controlling the temperature in a rod**) Using the model that we developed in Homework #2, we want to control the temperature at two points in the interior of a metal rod by adjusting the temperatures we can impose at the endpoints.

   (a) To start this investigation, we write a function that takes two end point temperature values and returns the temperature field. To do this, we can "wrap" our code from homework #2 (call this "legacy code"!) inside of a function like this :

```
function [T,x] = solve_temp(a,b)
     % Input  : (a,b) Temp. values at left and right end points
     % Output : T, vector of values T at x=0.01, x=0.02, ..., x=0.99.
     % Output : x, vector x = [0.01, 0.02, 0.03, ..., 0.99].

     % "Wrap" code from homework #2, including physical parameters.
end
```

Just as in Homework #2, this function will compute the temperature at the points $x_1 = 0.01$, $x_2 = 0.02$, ..., $x_{99} = 0.99$. The new twist is that the endpoint values are now *variable*, given now by $T_0 = a$ and $T_N = b$. With the exception of the flame, which should be centered at $L/2$ (not $3L/5$), all parameters and problem description are exactly as described in Homework #2. Do not include any plotting commands in `solve_temp`.

Check your solution : Call your function using the temperature end point values $T_0 = 10$ and $T_N = 25$. You should get :

```
[T,x] = solve_temp(10,25);
fprintf('%24.16f\n',T([25 75]));   % Temperature at x=0.25 and x=0.75
```

Your results should be very close to :

```
 69.1391828407954989
 76.6391828407959821
```

(b) We want to find boundary conditions that lead to specified interior temperatures at $T(0.25)$ and $T(0.75)$. To do this, we will use the fact (easy to show) that the relation between the endpoint boundary conditions and the interior temperature values T_{25} and T_{75} is given by the *linear* system

$$\begin{bmatrix} T_{25} \\ T_{75} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{1}$$

You could use this linear system to solve for unknown boundary temperatures (a, b), given desired temperature values T_{25} and T_{75}.

The only problem with this formulation is that we don't know any of the six coefficients a_{11}, a_{12}, a_{21}, a_{22}, F_1 or F_2. So your next task is to figure out how to find these coefficients. You do this by making clever choices of input values (a, b) to your `solve_temp` function, and use the return values to determine the unknown six coefficients. Here is the sketch of a function you can write to construct the linear system in (1).

```
function [A,F] = construct_system()
   % Construct coefficient matrix A and vector F

   % Call 'map_bc' with clever choices of (a,b) to
   % find six unknown coefficients in A and F.

   % This 'nested' function will be useful!
   function [T25,T75] = map_bc(a,b)
      [T,x] = solve_temp(a,b);
      T25 = T(25);
      T75 = T(75);
   end
end
```

The function `construct_system` has no arguments, but returns the 2×2 matrix A and the 2×1 vector F needed to define the linear system in (1).

Check your solution : You should get $a_{11} = 0.75$ and $F_1 = 55.3891828407956552$.

(c) Put everything together to write your `prob2` function.

```
function prob2()

close all;

[A,F] = construct_system()
```

```
write_file(A,'A.out');
write_file(F,'F.out');

probci(A,F)
probcii(A,F)

end
```

where `probci` and `probcii` are two Matlab functions you will write to solve the problems below.

i. You want the temperature at $x = 0.25$ to be $35C^\circ$ and the temperature at $x = 0.75$ to be $70C^\circ$. Determine the temperature that you should impose at the left and right endpoints of the rod to achieve this temperature profile.
Plot the points (x_{25}, T_{25}) and (x_{75}, T_{75}), and, on the same graph, plot the temperature curve you obtained using the left and right end point temperatures you found.
Save your results using
`write_file([a, T25, T75, b],'temp1.out');`
where (a, b) are the left and right temperature values you found.

ii. You can only hold the left end of the rod at $10C^\circ$, but you are willing let the temperature at $x = 0.25$ to be $40C^\circ$. How hot will rod be at $x = 0.75$ and the right end point? Will it be any where near the $70C\circ$ you need for your experiment?
On a second graph, plot this new temperature you found, and indicate the two temperatures at $x = 0.75$ and $x = L = 1$ with markers on the plot. Save your four temperature values (a, T_{25}, T_{75}, b) to the file `temp2.out`. Add axis labels, and a title to your plots.

3. (**Function approximation using polynomials**) Data interpolation is an important problem in scientific computing. One way to interpolate a set of $n + 1$ data points is to find an n^{th} degree polynomial that *interpolate* the data points. In Matlab, there are at least two different ways to do this. For example, you can set up a Vandermonde matrix system using `vander`, or use the `polyfit` command. We will learn other methods in the next few homework assignments.

   Download the data sets `poly1.dat`, `poly2.dat` and `poly3.dat` from the course website. These data sets contains diffferent discrete "samplings" of the function $f(x) = e^{-5x^2}\sin(4\pi x)$ in the interval $[-1, 1]$.

   (a) Using the data in `poly1.dat`, convince your self that both `vander` and `polyfit` produce the same set of polynomial coefficients. Provide evidence, either graphically, or by some other means that convinces a potential colleague that these two routines are essentially the same.

   (b) Write a function that takes as input a set of data points, and a function handle from which these points were sampled, and produces a plot of the interpolating polynomial, the data points, and original function.
   Use this function to create three plots from the three data sets you downloaded. What can you say about the different "samplings" of data? Do some seem to lend themselves better to polynomial interpolation than others?
   In your plots, be sure to include the following items.
   - Use the Matlab command `markersize` to see the data points clearly.
   - Use the `legend` command to add a legend to each plot showing the interpolating polynomial, the original function and the data points.
   - Finally, use either the `axis` command or the `ylim` command to restrict the y-axis to show the interesting parts of the graph. Note that the interpolating polynomial may have values that are quite large, relative to the original function, and so it might not always be possible to show the entire interpolating polynomial and still see anything interesting of the original function.

   See Figure 1 for a sample plot.

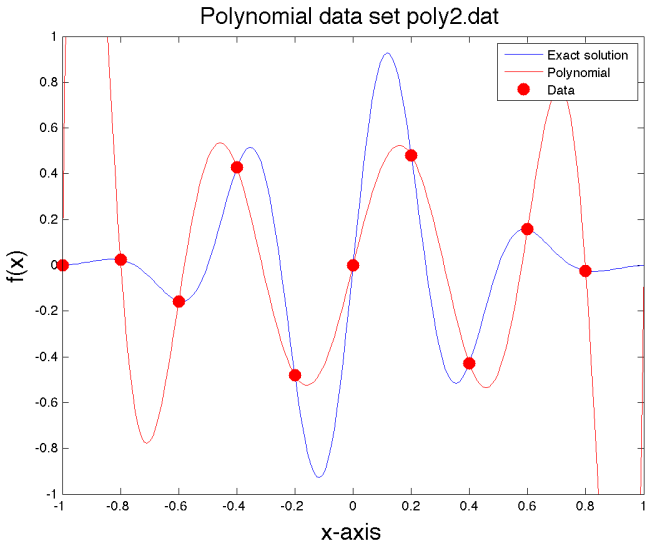

Figure 1: Example of polynomial interpolation for Problem 3.