Name : ______________________________

Homework Project #3

Math 365, Spring 2016

Due Wednesday, February 24th

1. (**Approximating derivatives of a function**) In Calculus I, you may have learned that we can approximate a derivative to a function using a formula such as

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

where h is a small value. This formula approximates the slope of a tangent line to the function $f(x)$ by finding the slope between the two nearby points $(x-h, f(x-h))$ and $(x+h, f(x+h))$. It is not hard to develop a similar formula for the approximation to the second derivative. A common approximation to $f''(x)$ is given by

$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2}$$

These formulas are commonly known as *finite difference approximations.*

For this problem, we will apply the above formulas to the function $f(x) = \cos(x)$.

   (a) Create anonymous functions for the exact values of $f(x)$, $f'(x)$ and $f''(x)$. Compute the derivatives of $f(x)$ by hand and code them into your anonymous functions.

   (b) Create an array of h values $h_i = 2^{-i}$, $i = 0, 1, 2, \ldots, N$ where N is chosen so that $1 + 2^{-N} > 1$ but $1 + 2^{-(N+1)} = 1$ (in floating point arithmetic).

   (c) Compute vectors containing approximations to the first and second derivatives at the point $x = 1$ using the formulas above for the vector of values of h_i.

   (d) Compute a vector of errors for each approximation by comparing your approximate values with the exact values $f'(1)$ and $f''(1)$.

   (e) Plot your errors on a `loglog` plot. Add a title, axis labels and a legend. Use the command `set(gca,'xdir','reverse')` so that the h values get smaller going from left to right on your plot. Your final graph should look like the one shown in Figure 1.

   (f) Answer this question : "Why do the errors decrease to a point, but then start to grow again for smaller values of h?"

2. (**Solving a differential equation**) In this problem, we will use finite difference approximation for the second derivative above to solve the differential equation

$$u''(x) = -4\pi^2 \sin(2\pi x)$$

on the interval $[0, 1]$, subject to $u(0) = u(1) = 0$.

To do this, we consider the following system of equations.

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

where the x_j are equally spaced points in the interval $[0, 1]$, with interval spacing $h = 1/N$. For boundary conditions, we assume $u_0 = u_N = 0$.

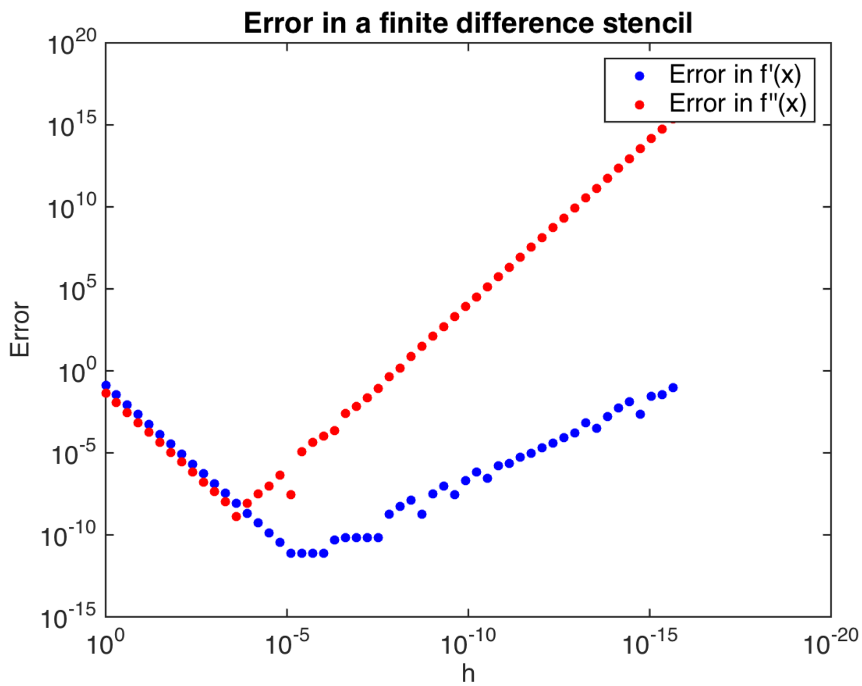

Figure 1: Graph for Problem 1.

(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. On the same graph, plot the exact solution $u_{exact}(x) = \sin(2\pi x)$. Your approximate solution and the exact solution should be very close.

(b) Create an array of N values in the range $N = 100$ to $N = 5 \times 10^5$ (use `logspace`). Then, for each N, create a sparse $N \times N$ matrix A used above to solve the differential equation. (Note : Because we are storing the matrix in sparse format, N can be much larger than what you computed in your last homework). Then,

i. For each matrix A, compute the *condition number* $\kappa(A)$ of A. Store this value in an array. **Hint**: Use the Matlab function `condest`.

ii. For each matrix A, solve the differential equation as you did in (2a). Compute the error in your approximation as

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

using the `norm` function with a second argument of `Inf`. Store this error in an array.

Plot the error you computed above 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`. 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.

3. (**Function approximation using polynomials**) Often, we want to replace a complicated function (one that involves expensive function evaluations, for example) with a polynomial. The question then becomes, what is the best way to approximate a function with a polynomial? This is a topic of a specialized area of mathematics called *approximation theory*.

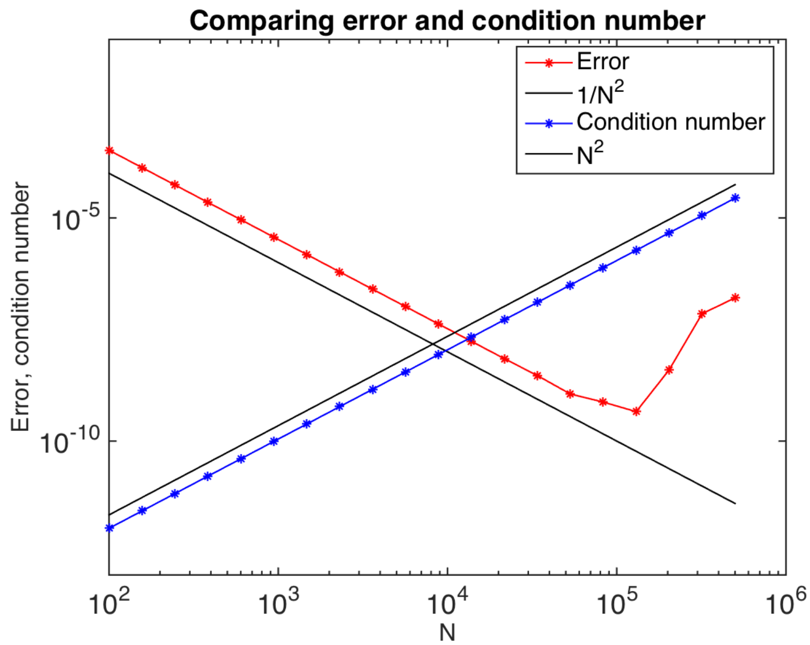

Figure 2: Plot for Problem 2.

You learned in Calculus that if we have enough derivatives of a function, we can construct a Taylor polynomial centered at a single point $x = c$. The disadvantage of this approach is that we might not have the needed derivatives (they might be too difficult to compute, for example). Another approach to creating a polynomial approximation is to use only the function value (no derivatives) at several points along an interval of interest.

In this problem, we will use this approach to find a polynomial approximation to the function

$$f(x) = \frac{1}{1 + 25x^2}$$

over the interval $[-1, 1]$. To see how different choices of points can lead to very different polynomials, you will create a polynomial using two sets of interpolation points, which you will construct below.

You will construct at set of 20 points x_j, $j = 1, 2, \ldots 20$ on the interval $[-1, 1]$. In each case, you will have $x_1 = -1$, and $x_{20} = 1$. In between -1 and 1, the points will be spaced differently, as described below. Evaluate the function $f(x)$ at the x_j to create values $y_j = f(x_j)$. Then, using the Matlab command `polyfit`, obtain the coefficients of the polynomial that interpolates (x_j, y_j). Using `polyval`, evaluate the polynomial at enough points to create a smooth curve and show how the polynomial behaves between your interpolation points.

For each of the two set of points (described below), create a plot of $f(x)$, your polynomial approximation, and the interpolation points. Each plot should show clearly that your polynomial approximation and $f(x)$ agree at the interpolation points. Add a legend, title (appropriate for the set of points used) and axis labels to each plot. Use `xlim` and `ylim` to make sure your axis limits are $[-1, 1] \times [0, 1]$.

The two sets of points x_j to use are

(a) An array of 20 equally spaced points x_j, $j = 1, 2, ...20$ on the interval $[-1, 1]$.

(b) An array x of 20 points $x_j = -\cos(t_j)$ where the t_j are equally spaced in the interval $[0, \pi]$. These are called "Chebyschev" nodes.

What can you say about the quality of each approximation? Does one do a a better job than the other?

For this problem, you will use the functions `linspace`, `polyfit`, and `polyval`.

4. (**Polynomial data fitting**) Determine the coefficients of polynomial curve fits of up to degree four for the density of saturated liquid water as a function of temperature. The data are in the `H2Odensity.dat` file on the course website.

   (a) Plot the data and the 4^{th} degree polynomial that you found.

   (b) For each polynomial $p_n(T)$ you obtain, also compute the maximum error as defined by

   $$\| e_n \|_\infty = \max_{1 \leq i \leq N} |p_n(T_i) - \rho_i|$$

   where T_i is the temperature data, ρ_i is the density data, and N is the number of data points in the data file. To compute the error, you can use the `norm` function. For example,

   ```
   err(n) = norm(pn - rho,Inf)
   ```

   where `pn` is the polynomial you found evaluated at the data points T_i and ρ_i are the measured values of density. Write the four errors you obtain to a file `polyerror.out`.