Name : _____

Homework Project #4

Math 365, Spring 2016

Due Wednesday March 9th

- **Problem 2.** Please publish in a separate file. Remember, I don't want to time my machine but your machine.
- **Problem 3.** There is no code to turn in, but please turn in your hand written work, either to me in class, in my mail box, or scan it in and submit it via your Dropbox folder.
- **Problems 1 and 4**. Include these problems in a single file so I can publish them to a single PDF document.

For all plots, be sure to use the `markersize` property to adjust the size of the data points. Also, add axes labels and a title, and where appropriate, a legend.

1. (**Curve fitting**) The temperature dependence of the reaction rate coefficient of a chemical reaction is often modeled by the Arrhenius equation

$$k = A \exp(-E_a/(RT)) \tag{1}$$

where k is the reaction rate, A is the pre-exponential factor, E_a is the activation energy, R is the universal gas constant, and T is the absolute temperature (K°). Experimental data for a particular reaction yield the following results.

$T(K^\circ)$	773	786	797	810	810	820	834
k	1.63	2.95	4.19	8.13	8.19	14.9	22.2

   (a) Use `polyfit` to get a least-squares fit of this data to obtain values for A and E_a for the reaction. Take $R = 8314 J/kg/K^\circ$. Write out the two values A and E_a that you find to the file `reaction_coeffs.out`.

   (b) Plot the data points (k_i, T_i). On the same plot, plot the best-fit curve of your data. Add a legend, axes labels and a title to your plot.

   (c) What is the reaction rate of this process at $T = 813K^\circ$? Use `fprintf` and a format string to print out your answer. Use floating point notation and show 2 significant figures after the decimal places.

2. (**Timing the linear solve using backslash verses the Matlab 'inverse'**) For this problem, you will fit timing data to a curve and compare the results of using the back-slash operator vs. using the inverse function `inv` to solve a linear system.

   - As in previous homeworks, use `logspace` and `round` to create an array of N values between some `nmin` and `nmax`. Take for example, `nmin` equal to 500 and `nmax` equal to the value you computed on Homework #2.

   (a) (**Create timing data**) Time two different ways to solve the dense linear system $A\mathbf{x} = \mathbf{b}$. The first way is to use the backslash operator, and the second way is to use the `inv` operator, i. e. `inv(A)*b`. Get timing data for each N value in a range of N values between `nmin` and `nmax`. Create a log-log plot of your timing data verses N for the two different methods.

(b) (**Fit data to a curve**) Assume that you can express timing data as a function of matrix size N according to the model

$$t = CN^p$$

where t is the time for the operation you are timing. Use `polyfit` to determine the parameters C and p that fit the model in the least squares sense. Plot the best-fit curves on your log-log plots.

(c) (**Report your results**) Using `fprintf`, print to two decimals places the value of p that you get for each operation. Be sure to add any descriptive text.

(d) (**Use your results to predict time required for large linear solves**) Use your best-fit data to estimate the time (in minutes) it would take to solve a dense $50,000 \times 50,000$ linear system. Print out the time for each solution method, as well as a ratio of the backslash operator time to the `inv` time. Which method is better to use as matrices get large? Compare this to the time it would take to solve the same-size sparse linear system you used in Homework # 2. How much faster is the sparse solve?

(e) **Compare to a sparse linear solve**. Time the sparse linear solve for the $(1, -2, 1)$ matrix used in previous homework and estimate the convergence rate p for the sparse solver. Add a `loglog` plot of the data to your graph for the other two solvers.

For this problem, you will be graded in part on how well you have formatted your results. Here are some tips :

- When printing out the exponents value `p`, only use 2 digits after the decimal.
- Print out your timing results in minutes; probably 1 decimal place after the decimal is sufficient.
- Use descriptive text in your `fprintf` statement to indicate what exactly you are printing out.

3. (**Lagrange polynomials**) The Lagrange polynomials simplify the task of polymomial interpolation because they provide an *explicit* formula for a polynomial that interpolates a set of data points (x_i, y_i), $i = 0, 1, \ldots, n$. They are also numerically more stable, because they don't require that we invert the ill-conditioned Vandermonde system.

Below are a few pencil and paper exercises to get you more familiar with Lagrange interpolants.

(a) Construct the Lagrange polynomials $\ell_0(x)$, $\ell_1(x)$ and $\ell_2(x)$ for the data $(-1, 3)$, $(0, 2)$, and $(1, 5)$. Verify that the three polynomials you found satisfy

$$\ell_j(x_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Do the Lagrange polynomials depend on both x_i and y_i? Or only the x_i values?

(b) Write down the Vandermonde matrix V for the same set of data. Obtain an exact expression for the inverse V^{-1} of this matrix, using either Gauss-Jordon elimination, or using the `inv` function in Matlab. In either case, verify by hand that $VV^{-1} = I$, where I is the 3×3 identity matrix.

(c) Show that the coefficients v_{ij} of $\ell_j(x) = v_{2,j}x^2 + v_{1,j}x + v_{0,j}$ appear in the columns of the inverse of the Vandermonde matrix.

(d) Show that the Lagrange interpolating formula gives exactly the same polynomial as the Vandermonde matrix method. Use the inverse matrix V^{-1} you found above to show this. How are the Lagrange interpolating polynomials and the columns of the matrix V^{-1} related?

4. (**Barycentric Formula**) The Lagrange Formula is a convenient way to write down an explicit formula for a polynomial interpolant. The best way to evaluate a polynomial using this form, however, is to use the *Barycentric Formula*. This problem asks you to implement the Barycentric Formula.

   You are asked to approximate the following function using as a 54 degree polynomial (this is not a typo!).

$$f(x) = |x| + \frac{x}{2} - x^2, \qquad x = x(t) = -\cos(t), \qquad t \in [0, \pi]$$

   See Figure 1 for a plot of the function $f(x)$.

   To get the best approximation, we will use "Chebyschev" nodes on the interval $[-1, 1]$. These are defined as

$$x_j = -\cos(t_j), \qquad j = 0, 1, \ldots, 54$$

   where the values t_j are equally spaced points in $[0, \pi]$. **Hint:** Use `t = linspace(0,pi,N+1)`, for $N = 54$.

   (a) Plot the true solution over the interval $[-1, 1]$. Add to your plot the data points you generate above.

   (b) Compute the 54 degree interpolant of the function above by using `polyfit` and `polyval`. Plot the polynomial that you get over the interval $[-1, 1]$. You should observe that the interpolant obtained this way does a very poor job of approximating the true function.

   (c) Write a function that computes the barycentric weights. Your function depends only on `xdata` and will look like this :

```
function w = baryweights(xdata)
     % Compute the weights ...
end
```

   Note : The fastest method for computing the weights can be done in one line of Matlab code!

   (d) Write a function that evaluates the approximating polynomial at a vector of `x` values. Your function will look like this:

```
function y = baryval(xdata,ydata,weights,x)
     % Evaluate the resulting polynomial.
end
```

   (e) Use your implementation of the Barycentric formula to construct an interpolant for $f(x)$, given above. Plot the resulting interpolant, along with the data points. You should see that the Barycentric form gives a near perfect (in the eye-norm, at least) approximation to the true function.

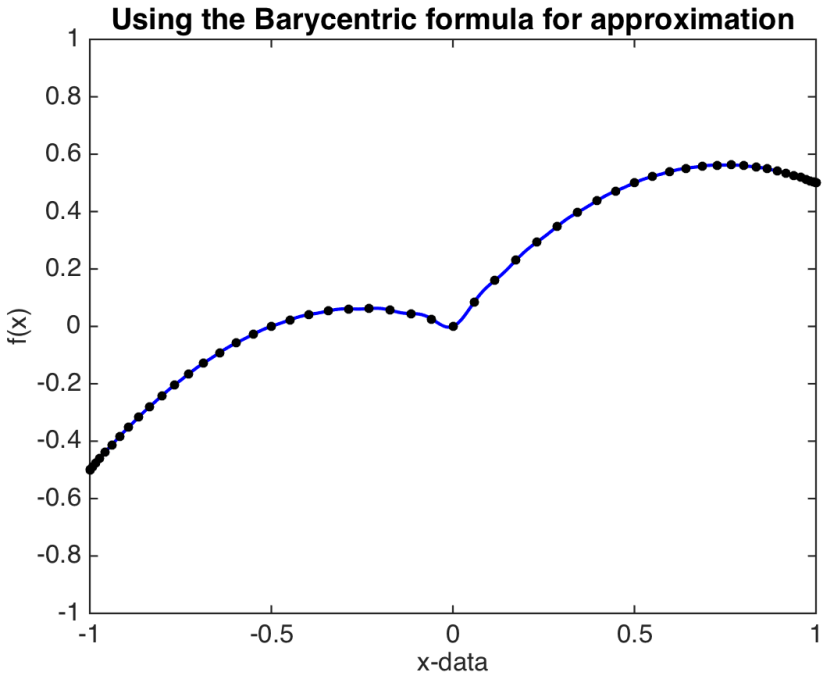

Figure 1: Problem 4 : Function approximation at 55 Chebyschev nodes.