Math 551 Fall 2018
Lab 7

Goal: In many practical applications it is necessary to approximate a function $f(x)$ given only a finite set of function values $\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))\}$. In this lab, you will learn how to do this first by fitting a polynomial through the given data points and then by using Matlab's `interp1` command. During the lab session, your lab instructor will teach you the necessary Matlab code to complete the assignment.

What you have to submit: An M-file containing all of the commands necessary to perform the tasks described below.

Reminders About the Grading Software

Remember, the labs are graded with the help of software tools. If you do not follow the instructions, you will lose points. If the instructions ask you to assign a value to a variable, you must **use the variable name given in the instructions**. If the instructions ask you to make a variable named `x1` and instead you make a variable named `x` or `X` or `X1` or any name other than `x1`, the grading software will not find your variable and you *will* lose points. Required variables are listed in the lab instructions in a gray box like this:

Variables: `x1`, `A`, `q`

At times you will also be asked to answer questions in a comment in your M-file. You must **format your text answers correctly**. Questions that you must answer are shown in gray boxes in the lab. For example, if you see the instruction

Q7: What does the Matlab command `lu` do?

your file must contain a line similar to the following somewhere

```
% Q7: It computes the LU decomposition of a matrix.
```

The important points about the formatting are

- The answer is on a single line.
- The first characters on the line is the comment character `%`.
- The answer is labeled in the form `Q`n`:`, where n stands for the question number from the gray box.

If you do not format your answers correctly, the grading software will not find them and you *will* lose points.

Polynomials in Matlab

In Matlab, we represent polynomials through the vectors of their coefficients ordered by decreasing power. That is, a polynomial of the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

is represented as the vector $[a_n, a_{n-1}, \ldots, a_1, a_0]$. (Matlab doesn't care if this is a row or column vector.) The Matlab function `polyval` can be used to efficiently evaluate a polynomial $p(x)$ at a particular value (or many values) of x. For example, to represent the polynomial $p(x) = 3x^5 - 4x^3 + 2x + 1$, we would create a vector

```
>> p = [3, 0, -4, 0, 2, 1];
```

Notice the zeros in the second and fourth entries of the vector. Those indicate that the coefficients for the x^4 and x^2 terms are 0. Be careful about this point. If you ignore zero coefficients you will get the wrong polynomial.

In order to evaluate this polynomial at the point $x = -2$, we use the `polyval` function.

```
>> polyval(p,-2)

ans =

   -67
```

`polyval` also provides an efficient way to evaluate a polynomial at many points simultaneously. See if you can interpret the output of the following command.

```
>> polyval( [1, 0, -1], [ 2, -1; 1, 0] )

ans =

     3     0
     0    -1
```

Tasks

Consider the following set of data points on the xy-plane:

$$(-2.1, 2.2), (-1.3, 5.1), (0.1, -1), (0.5, 0.1), (1.4, 0.6), (2.0, 1.8). \qquad (\star)$$

You will need to find a polynomial of degree at most 5 that passes through these 6 points. (Does this polynomial exist? Is it unique?) Please, complete the following steps:

1. Create two row vectors `x` and `y`, containing respectively the x and y coordinates of the data points in $(\star)$.
   Variables: x, y

2. Assuming there exits a degree 5 polynomial which passes through the points in $(\star)$, write down the system of 6 linear equations representing this information, i.e. each equation in the system shows that one of the (x, y) pairs in $(\star)$ solves a certain degree 5 polynomial. (You don't need to submit this task.)

3. Create a variable holding the coefficient matrix A of the system in Task 2. Do not enter the matrix entries by hand. Instead, initialize a zero matrix of the appropriate dimension using `zeros` and then fill in the coefficients using a pair of `for` loops. So your code will look something like this

   ```
   n = length(x);
   A = zeros(n,n);
   for i=1:n
       for j=1:n
           A(i,j) = ... <---- this is the part you figure out
       end
   end
   ```

   Variables: A

4. If you have completed the previous tasks correctly, you will have generated the coefficient matrix `A` for your linear system from Task 2. Notice that the constants from the right-hand side are just the entries in the row vector `y`. Solve the system using Matlab's backslash operator. (You'll need to use `y'` as your constant vector, since `y` is a row vector.) Name the solution `p` (a good name for a variable holding polynomial coefficients).
   Variables: p

5. At this point, it would be a good idea to compare the values of `polyval(p,x)` with the vector `y` to make sure you got the right answer. To do this, add the following commands to your script.

```
% error check
err = max( abs( polyval(p,x) - y ) );
fprintf( 'max error in polyval = %e\n', err );
```

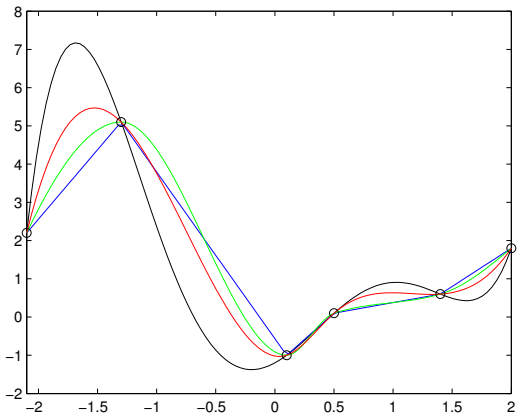
Now, when you run your script, Matlab will report the largest absolute difference between the vector `y` and the vector `polyval(p,x)`. Due to errors introduced internally by rounding, this value will almost certainly not be zero, but it should be a very small number like `4.884981e-15`, which is Matlab's representation of the number 4.884981×10^{-15}. (This step helps you check your work. There are no required output variables.)

6. Divide the interval $[-2.1, 2.0]$ into 100 equal intervals using the `linspace` command and store it to the variable `x1` (`x1` should have 101 entries). If you haven't worked with `linspace` before, see if you can figure out what the following commands do: `linspace(-1,3,2)`, `linspace(-1,3,3)` and `linspace(-1,3,11)`.
   Variables: x1
7. Evaluate the polynomial at the points of the array `x1` and store the result to the variable `y1`.
   Variables: y1

In applications with many data points, polynomial fitting is often not practical for a number of reasons. In those cases, other methods of interpolation are used. There are several methods of interpolation available in Matlab. We will consider the most commonly used linear, cubic (piecewise cubic Hermite interpolation) and (cubic) spline interpolation. Your lab instructor will explain the difference between those three.

Matlab's basic interpolation command is called `interp1`. Use the command `doc interp1` to bring up its documentation and see if you can figure out how to use it. Don't try to read every detail. It can help to scroll down to the examples and see the command in action.

7. Use `interp1` to compute the values `y2` of the polynomial at the points `x1` with 'linear' option.
   Variables: y2
8. Use `interp1` to compute the values `y3` of the polynomial at the points `x1` with 'cubic' option.
   Variables: y3
9. Use `interp1` to compute the values `y4` of the polynomial at the points `x1` with 'spline' option.
   Variables: y4
10. Plot `y1, y2, y3, y4` as functions of `x1` on the same graph using the `plot` command. Use black color for `y1`, blue for `y2`, green for `y3` and red for `y4`. Mark the initial set of points stored in `x` and `y` on this graph using black circles. The result should look something like this:



Notice the differences in the results of using the various methods of interpolation.