CS412 Spring Semester 2011

Homework Assignment #2

Due Thursday March 15th 2012, in class

Sum of all problems : 120%, Maximum possible score : 100%.

The following problems will introduce you to the `MATLAB` functions `poly`, `polyval` and `error`, while experimenting with Newton interpolation.

1. [40%] In this problem you will write a `MATLAB` function that computes the coefficients of each polynomial function $n_k(x)$ used in the Newton interpolation method.

   Consider $N+1$ values along the x-axis, denoted by $x_0, x_1, x_2, \ldots, x_{N-1}, x_N$. We define the *Newton polynomials* $n_0(x), n_1(x), \ldots, n_N(x)$ as follows:

   $$n_k(x) = (x - x_0)(x - x_1)\cdots(x - x_{k-2})(x - x_{k-1}) = \prod_{i=0}^{i=k-1}(x - x_i)$$

   [For the special case $k = 0$ we define $n_0(x) = 1$.]

   For this problem, you must generate each n_k in the more standard form:

   $$n_k(x) = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} + a_k x^k \tag{1}$$

   Write a `MATLAB` function `NewtonPolynomial(x,k)` which acts as follows:

   - Takes as a first parameter the vector `x=[`x_0`,`x_1`,`x_2`,...,`x_N`]` containing all the x-coordinates of the data points we wish to interpolate
   - Takes as the 2nd parameter the integer `k` ($0 \leq$ `k` $\leq N$), which selects which specific polynomial $n_k(x)$ we want to generate.
   - Returns a row vector with $N + 1$ entries

     `p=[`a_N`,`a_{N-1}`,`a_{N-2}`...,`a_2`,`a_1`,`a_0`]`

     containing the coefficients of $n_k(x)$ as used in equation (1). Note that since n_k is only a k-degree polynomial, the first $N - k$ entries of `p` would be zero. For example, if $N = 5$ and $k = 3$ the result of `newton_polynomial(x,3)` would be of the form:

     `p=[0,0,`a_3`,`a_2`,`a_1`,`a_0`]`

     This particular convention makes it easier to add two or more polynomials (of possibly different degree) by simply adding their coefficient vectors `p`.

If the parameter `k` is not an integer between 0 and N, the function should abort with the error message `'Index out of bounds'`, using the `MATLAB` command `error`.

Turn in your code, and also test your implementation by using `x0=1,x1=2, x2=3,x3=5`, and report the coefficients you generated for `k=0,1,2,3`.

Hint: You may use the `MATLAB` function `poly(u)` which takes as argument a vector `u=[`u_1`,`u_2`,`$\ldots$`,`u_m`]` and returns another vector `c=[`c_m`,`$\ldots$`,`c_1`,`c_0`]` with the coefficients of the polynomial:

$$c_m x^m + \cdots + c_1 x + c_0 = (x - u_1)(x - u_2)\cdots(x - u_m)$$

You can use function `poly` to generate the coefficients of $n_k(x)$ defined above. Of course, you would need to take some extra steps to re-format these coefficients according to the convention described above (i.e., as a vector of $N + 1$ entries, with leading zeros).

2. [40%] Write a `MATLAB` function `DDTable(x,y)` which acts as follows:

   - Takes as a first parameter the vector `x=[`x_0`,`x_1`,`x_2`,...,`x_N`]` containing all the x-coordinates of the data points we wish to interpolate
   - Takes as a second parameter the vector `y=[`y_0`,`y_1`,`y_2`,...,`y_N`]` containing all the y-coordinates of the data points we wish to interpolate
   - Returns a lower triangular $(N + 1) \times (N + 1)$ matrix which contains the divided difference table:

$$\mathtt{T} = \begin{pmatrix} f[x_0] & & & & \\ f[x_1] & f[x_0, x_1] & & & \\ f[x_2] & f[x_1, x_2] & f[x_0, x_1, x_2] & & \\ \vdots & \vdots & \vdots & \ddots & \\ f[x_N] & f[x_{N-1}, x_N] & f[x_{N-2}, x_{N-1}, x_N] & \cdots & f[x_0, x_1, \ldots, x_N] \end{pmatrix}$$

   Test your implementation with

   `x=[-1,0,2,3]`

   `y=[-10,-4,2,14]`

   Turn in your code, and a printout of the matrix that is produced for these inputs.

   To help you *debug* your code, you are given the following `MATLAB` file:

   `http://pages.cs.wisc.edu/~cs412-1/hw/DividedDifference.m`

   This implements a function `DividedDifference(x,y)` which computes the divided difference value $f[x_0, x_1, \ldots, x_N]$ using a recursive approach.

3. [40%] Using the code you wrote for the previous problems, write yet another function `NewtonInterpolation(x,y)` which operates as follows:

   - Takes as a first parameter the vector x=[x_0,x_1,x_2,...,x_N] containing all the x-coordinates of the data points we wish to interpolate
   - Takes as a second parameter the vector y=[y_0,y_1,y_2,...,y_N] containing all the y-coordinates of the data points we wish to interpolate
   - Returns a row vector with $N+1$ entries

     p=[a_N,a_{N-1},a_{N-2}...,a_2,a_1,a_0]

     corresponding to the coefficients of an N degree polynomial

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

     that interpolates through the points $(x_0, y_0), (x_1, y_1) \ldots, (x_N, y_N)$.

   This polynomial should be computed using Newton interpolation.

   Remember that, if we have alredy computed the Newton polynomials $n_0(x), n_1(x), \ldots, n_N(x)$ and the divided difference table, then we have

   $$\begin{aligned} p(x) &= f[x_0]n_0(x) \\ &\quad + f[x_0, x_1]n_1(x) \\ &\quad + f[x_0, x_1, x_2]n_2(x) \\ &\quad \vdots \\ &\quad + f[x_0, x_1, \ldots, x_N]n_N(x) \end{aligned}$$

   Turn in your code and test your implementation by computing the coefficients of the cubic polynomial that interpolates the four data points (this is the same test as the previous question):

   $$(-1, -10) \quad (0, -4) \quad (2, 2) \quad (3, 14)$$

   Plot the generated interpolating polynomial in the interval $[-1, 3]$. You will need to compute a more dense set of (x, y) values in order to generate a smooth plot. The `MATLAB` function polyval(p,x_*) can be used for this purpose: when passed a vector `p` containing the coefficients of $p(x)$ as described above, and an additional value x_*, it will return the corresponding value $y_* = p(x_*)$. Include a printout of this smooth interpolating curve with your solutions.