MATH 104A

Homework 4

General Instructions: You have to integrate all the problems that require coding and/or numerical computation in a single *jupyter* notebook. Make sure all your codes have a preamble which describes purpose of the code, all the input variables, the expected output, your name, and the date of the last time you modified it. Write your own code, individually. Do not copy codes! The solutions to the problems that do not require coding must be uploaded as a single pdf or as part of the *jupyter* notebook.

1. In Newton's form of the interpolation polynomial we need to compute the coefficients, $c_0 = f[x_0], c_1 = f[x_0, x_1], \dots, c_n = f[x_0, x_1, \dots, x_n]$. In the table of divided differences we proceed column by column and the needed coefficients are in the uppermost diagonal. A simple 1D array, c of size $n + 1$, can be used to store and compute these values. We just have to compute them from bottom to top to avoid losing values we have already computed. The following pseudocode does precisely this:

```
for $j = 0, 1 \dots, n$
    $c_j = f_j$;
end
for $k = 1, \dots n$
    for $j = n, n-1, \dots k$
        $c_j = (c_j - c_{j-1})/(x_j - x_{j-k})$;
    end
end
```

   The evaluation of the interpolation polynomial in Newton's form can be then done with the Horner-like scheme seen in class:

```
$p = c_n$
for $j = n-1, n-2, \dots, 0$
    $p = c_j + (x - x_j) * p$;
end
```

(a) Write computer codes to compute the coefficients $c_0, c_1, \ldots, c_n$ and to evaluate the corresponding interpolation polynomial at an arbitrary point x. Test your codes and turn in a run of your test.
(b) Consider the function $f(x) = e^{-x^2}$ for $x \in [-1, 1]$ and the nodes $x_j = -1 + j(2/10)$, $j = 0, 1, \ldots, 10$. Use your code(s) in (a) to evaluate $p_{10}(x)$ at the points $\bar{x}_j = -1 + j(2/100)$, $j = 0, 1, \ldots, 100$ and plot the error $f(x) - p_{10}(x)$.

2. Obtain the Hermite interpolation polynomial corresponding to the data $f(0) = 0$, $f'(0) = 0$, $f(1) = 2$, $f'(1) = 3$.

3. Find a piecewise linear function that interpolates the points $(0, 1), (1, 1), (2, 0), (3, 10)$.

4. Write a code to compute a *natural* spline $S(x)$ which interpolates a collection of given points $(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)$ where $x_0 < x_1 < x_2 < \ldots < x_n$ (do not assume they are equidistributed). Your code should have a triadiagonal solver for the resulting linear system of equations (you're not allowed to use Matlab's \ operator to solve the linear system).

5. One important application of spline interpolation is the construction of smooth curves that are not necessarily the graph of a function but that have a parametric representation $x = x(t)$ and $y = y(t)$ for $t \in [a, b]$. Hence one needs to determine two splines interpolating (t_j, x_j) and (t_j, y_j) $(j = 0, 1, \ldots n)$.

   The arc length of the curve is a natural choice for the parameter t. However, this is not known *a priori* and instead the t_j's are usually chosen as the distances of consecutive points:

$$t_0 = 0, \quad t_j = t_{j-1} + \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}, \quad j = 1, 2, \ldots n. \tag{1}$$

   Use the values in Table 1 to construct a smooth parametric representation of a curve passing through the points (x_j, y_j), $j = 0, 1, \ldots, 8$ by finding the two natural cubic splines interpolating (t_j, x_j) and (t_j, y_j), $j = 0, 1, \ldots 8$, respectively. Tabulate the coefficients of the splines and plot the resulting (parametric) curve.

j	t_j	x_j	y_j
0	0	1.50	0.75
1	0.618	0.90	0.90
2	0.935	0.60	1.00
3	1.255	0.35	0.80
4	1.636	0.20	0.45
5	1.905	0.10	0.20
6	2.317	0.50	0.10
7	2.827	1.00	0.20
8	3.330	1.50	0.25

Table 1: Table of data for Problem 5