Math 131: Numerical Analysis
Homework Number 4
Due: March 6, 2024 5:00 PM

Special Instructions:

- If instructed to "write/implement a code" or "write/implement an algorithm", you should interpret this as meaning that you need to produce a code in Jupyter/python notebook.
- Unless explicitly stated, you may not use any system/python packages or functions that implement one of the algorithms for this assignment. Standard mathematical functions (e.g. pow, abs, log, sin, exp, etc.) are allowed.
- You should include all the code source you implemented as one python notebook when turning in your assignment. Your code must run as is (and provide the correct results) to receive full credit.
- PDF files of your code, screenshots, etc. will not be graded.
- All code should be properly documented and include (**at a minimum**) 1) a summary of what the code is doing, 2) a description of all parameters that are used, and 3) a description of the output.
- The notebook itself should also be properly documented. Points will be **heavily** deducted for any notebook and/or code that is not properly documented.

1. Lagrange Polynomials: Hand calculation (20 points)

(a) Construct by hand an interpolating Lagrange polynomial of degree 3 for the following data:

Table 1: Interpolating nodes

x	-1	0	1	2
y	2	2	3	1

2. Lagrange Polynomials: Programming (30 points)

(a) **Construction:** Implement a function CompBaryWeights that takes a vector $X = (x_0, x_1, ..., x_n)$ as input and computes the barycentric weights (as defined in class and in the lecture notes) for the Lagrange Polynomial form. The function should return the vector of weights in $w = (w_0, w_1, \dots w_n)$.

The header of the function should be:

```
def  CompBaryWeights(X):
--- your code here ---
return w
```

Note that a vector of the function values $Y = (y_0, y_1, ...y_n)$ is not required during the construction of the Lagrange polynomial at this step.

(b) **Evaluation:** Implement a program that takes the vectors $X = (x_0, x_1, \dots, x_n), Y = (y_0, y_1, \dots, y_n)$, the $w = (w_0, w_1, \dots w_n)$ vector from part (a), and a vector $Z = (z_0, z_1, \dots, z_m)$ containing points at which to evaluate the polynomial. The program should return a vector containing the values of the interpolating polynomial at each of the points in Z, in other words, $P_n(Z) = (P_n(z_0), P_n(z_1), ..., P_n(z_m))$. Note that n need not necessarily equal m.

The header of the function should be:

```
def EvalLagr(Z, X, Y, w):
--- your code here ---
return PnZ
```

(c) Using the programs you wrote in part (a,b), graph $f(x), P_n(x)$ and the absolute error in approximation over the intervals given. **Hint:** You may use the function LagrInterpPolyError in the notebook Math131Hwk4-PolyInterp.ipynb to plot the values of the function $f(x)$ along with the values of an interpolating polynomial $P_n(x)$ and the absolute error $|f(x) - P_n(x)|$ on an interval $[x_0, x_n]$ using the particular function $f(x)$ and values $x_0, x_n, N = n + 1$ interpolation nodes given below. The file will contain placeholders for the calls to the two functions you will write in (a) and (b).

1. $f(x) = \frac{e^{0.01x} \cdot \sin(17x^2)}{(1+25x^2)}, \quad 0 \le x \le 1$ and $N = 5, 10, 20$
2. $f(x) = \frac{1}{1+25x^2}, \quad -1 \le x \le 1$ and $N = 10, 20, 40$

(d) Based on the plots you obtained explain the behavior of the error as N increases and provide an explanation for the difference between the errors in approximating functions in (1) and (2).