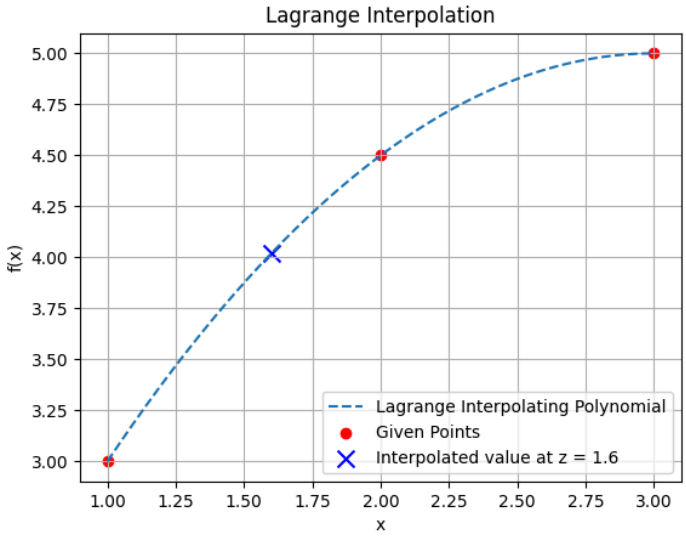
Natasha Piedrabuena
Numerical Computation
Lagrange Interpolation

In this report, Lagrange Interpolation, a polynomial interpolation method, was implemented to estimate the values of a function given a discrete set of known points. Lagrange Interpolation constructs a polynomial that passes exactly through each of these points by using a combination of Lagrange basis polynomials. Unlike root-finding methods, which aim to locate zeros of functions, interpolation techniques focus on approximating the function's behavior over a range by fitting it to known values. The user can input a set of data points (x-coordinates and their corresponding y-coordinates), and the interpolation algorithm calculates the polynomial that fits these points. Additionally, the user specifies a target point at which the interpolated value is calculated. This approach provides an effective way to estimate unknown function values when the underlying functional form is not available but the data points are.

Problem A)



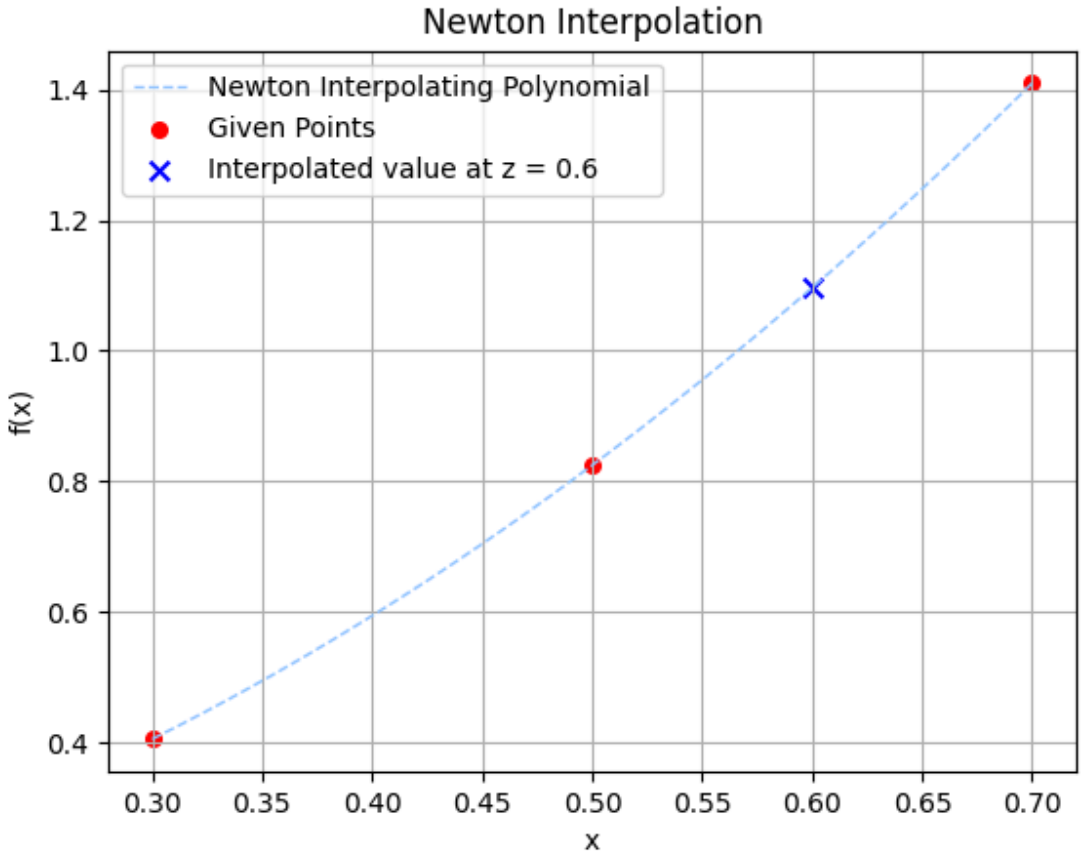
For problem A, a lagrange interpolation function is constructed for the x coordinates, y coordinate and the interpolated data point. The code was constructed using the sudo code provided in lecture 8.

```
Lagrange Interpolation
Enter the x-coordinates (comma-separated): 1,2,3
Enter the corresponding y-coordinates (comma-separated): 3,4.5,5
Enter the point at which you want to interpolate: 1.6

Lagrange Interpolation Results
----------------------------------

x-coordinates: [1.0, 2.0, 3.0]
y-coordinates: [3.0, 4.5, 5.0]
Interpolated value at 1.6: 4.0200000000000005
```

Then I chose these data points to evaluate the system before moving on to the next problem. I chose to use x [1, 2, 3] and f(x) [3,4,5]. The Interpolated value ~ 4.02 and the graphical representation can be seen in the first figure.
Problem B)

For problem B, I asked chatgpt to create a lagrange interpolation function with the description of lecture 8 and problem question. There isn't much difference between mines and the generated AI.

Problem C)
Ex. First Run (first three data points)



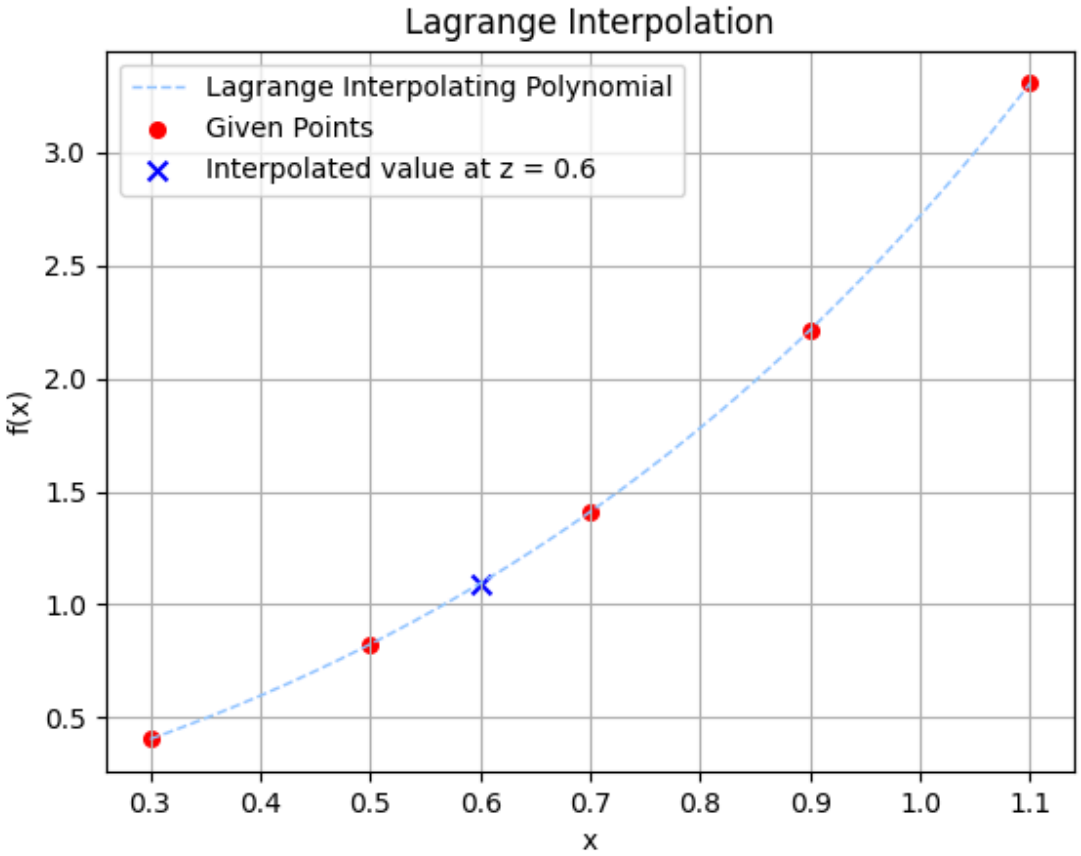
For problem C, I had to turn to the internet to get some references on how to implement Newton's interpolation method [1]. Majority of the implementation is from my original code on Problem D. The screenshots are in the Screenshot directory and each problem as a subdirectory.

```
Newton's Interpolation
Enter the x-coordinates (comma-separated): 0.3,0.5,0.7,0.9,1.1
Enter the corresponding y-coordinates (comma-separated): 0.404958,0.824361,1.40963,2.21364,3.30458
Enter the point at which you want to interpolate: 0.6
Interpolated value at 0.6 with 'First Run' points: 1.0962622499999999

Interpolated value at 0.6 with 'Second Run' points: 1.0929575625

Interpolated value at 0.6 with 'Third Run' points: 1.0939146875

Interpolated value at 0.6 with 'Fourth Run' points: 1.093316484375
```

Newton's interpolation method gave the same results as the lagrange and barely any difference between them. However theoretically the newton's interpolation should give a more accurate result compared to the lagrange. The 1.09327 is the approximation expected, and with the newton interpolation it's the same values as the Lagrange interpolation.

When asking it to chatgpt the output didn't seem to be exactly right even though I provided the equation and data point. However the program changed the data point. It can be manually tweaked to show the right data points.

Problem D)
Ex. All data points



For all four stopping points for the set data points it was implemented. For this problem I decided to show in a graph to ensure the program is running correctly. These graphs are located in the screenshot/problem_d.

```
Lagrange Interpolation
Enter the x-coordinates (comma-separated): 0.3,0.5,0.7,0.9,1.1
Enter the corresponding y-coordinates (comma-separated): 0.404958,0.824361,1.40963,2.21364,3.30458
Enter the point at which you want to interpolate: 0.6
Interpolated value at 0.6 with 'First Run' points: 1.0962622499999999

Interpolated value at 0.6 with 'Second Run' points: 1.0929575625

Interpolated value at 0.6 with 'Third Run' points: 1.0939146875

Interpolated value at 0.6 with 'Forth Run' points: 1.093316484375

Would you like to continue:(Y/N) n
```

The 1.09327 is the approximation expected, however for my runs they were a bit off. The forth run was closer to the expected result.

[1] Python Numerical Methods. (n.d.). *Newton's polynomial interpolation*. UC Berkeley. Retrieved November 7, 2024, from https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter17.05-Newtons-Polynomial-Interpolation.html