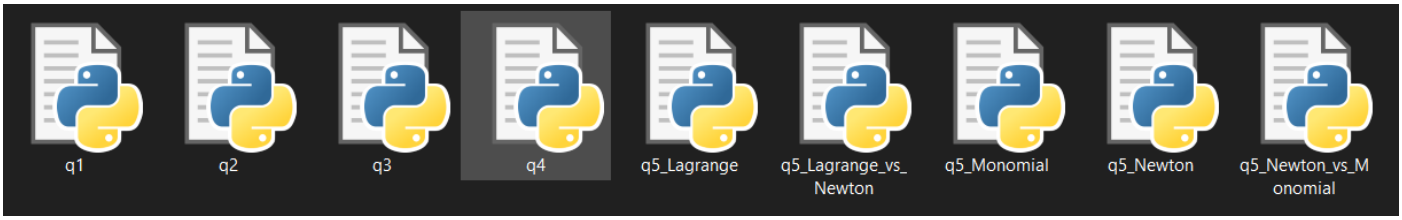

Name: Udandarao Sai Sandeep
Roll Number: 180123063
Dept.: Mathematics and Computing

Note:
Use the following command for running the python files.
#python3 filename.py (in Linux terminal)
Note: There are 9 python files in submission. The filenames are as follows:



Q1. $f(x) = e^x \quad 0 \le x \le 2$

i) $x_0 = 0, x_1 = 0.5$

The value of the function f at x = 0.25 was approximated using the Linear Lagrange Interpolation Method. The approximate value is as follows:

```
f(0.25) using linear Lagrange interpolation = 1.324360635350064
```

ii) $x_0 = 0.5, x_1 = 1$

The value of the function f at x = 0.75 was approximated using the Linear Lagrange Interpolation Method. The approximate value is as follows:

```
f(0.75) using linear Lagrange interpolation = 2.183501549579587
```

iii) $x_0 = 0, x_1 = 1, x_2 = 2$

The value of the function f at x = 0.25 was approximated using the second Lagrange Interpolating polynomial. The approximate value is as follows:

```
f(0.25) using second Lagrange interpolation = 1.1527742906760838
```

$x_0 = 0,\ x_1 = 1, x_2 = 2$

The value of the function f at x = 0.75 was approximated using the second Lagrange Interpolating polynomial. The approximate value is as follows:

```
f(0.75) using second Lagrange interpolation = 2.0119152049056064
```

iv)

Errors while using **Linear Lagrange Polynomial:**

```
Error b/w approximate and actual value at 0.25 (Linear): 0.0403352186623227
Error b/w approximate and actual value at 0.75 (Linear): 0.0665015329669120
```

Errors while using **Second Lagrange Polynomial:**

```
Error b/w approximate and actual value at 0.25 (Second): 0.1312511260116575
Error b/w approximate and actual value at 0.75 (Second): 0.1050848117070684
```

It is evident **that linear Lagrange interpolating polynomial** approximates the polynomial at x = 0.25 and at x = 0.75 with better accuracy.

We can observe that the values to be approximated are x = 0.25 and x = 0.75. Both values of x lie between **[0,1]**. In the case of Linear Lagrange Polynomial, concise end points were used ([0,0.5] for 0.25 and [0.5,1] for 0.75). At such small intervals, the function $f = e^x$ behaves approximately as the **line joining the endpoints**. Hence, the approximation in this case produced **less** errors.

However, for Second Lagrange Polynomial, the initial points considered were spread apart, and were not evenly spread with respect to the targeted points (0.25 and 0.75). Here, the interval **[0,2]** has a larger size, whereas the targeted points **x= 0.25 and x = 0.75** could easily fit in the interval [0,1]. The larger space between the selected initial points (0, 1 and 2) resulted in **larger** errors.

Q2.

i)Using Lagrange's Interpolating polynomials of first, second, and third, the value of the given function **f** was approximated at **x = 8.4**. The results are as follows:

```
x0 = 8.3
x1 = 8.6
f(8.4) using linear Lagrange interpolation = 17.87833

x0 = 8.1
x1 = 8.3
x2 = 8.6
f(8.4) using second Lagrange interpolation = 17.877129999999998

x0 = 8.1
x1 = 8.3
x2 = 8.6
x3 = 8.7
f(8.4) using third Lagrange interpolation = 17.8771425
```

ii) Using Lagrange's Interpolating polynomials of first, second, and third, the value of the given function **g** was approximated at **x =-1/3**. The results are as follows:

```
x0 = -0.5
x1 = -0.25
f(-1/3) using linear Lagrange interpolation = 0.2150416666666667

x0 = -0.75
x1 = -0.5
x2 = -0.25
f(-1/3) using second Lagrange interpolation = 0.1803055555555556

x0 = -0.75
x1 = -0.5
x2 = -0.25
x3 = 0
f(-1/3) using third Lagrange interpolation = 0.17451851851851857
```

Q3.

$f(x) = e^{-x^2}$

The coefficients of the Lagrange's Polynomial (of degree 2) corresponding to the above function f was calculated. The polynomial $P_2(x)$ is as follows:

```
x0 = -1
x1 = 0
x2 = 1
P2(x) = -0.6321205588285577x^2 + 0.0x + 1.0
```

Then, the value of **f (0.9)** and **P_2(0.9)** was compared. Error between actual and real value is as follows:

```
P2(0.9) = 0.487982
f(0.9) = 0.444858
Error b/w actual and calculated values: 0.043124
```

Theoretical bound for maximum error was calculated (theoretically) using the below formula:

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n)$$

Here, $f^{(3)}(x) = -4e^{-x^2}.x.(2x^2 - 3)$

Since $|f^{(3)}(x)| < 4$ for all x in [-1,1],

Maximum Theoretical Bound on error = max(abs(f(x)-p(x))) for all x in [-1,1] = **0.114**

Q4.

$f(x) = log_{10}(tan\,(x))$

Similar to the algorithm followed in above questions, a third-degree Lagrange polynomial approximation (**P(x)**) was constructed. The polynomial is as follows:

Note: All float point numbers have been rounded off to 4 digits.

```
x0 = 1.0
x1 = 1.05
x2 = 1.1
x3 = 1.15
P(x) = 1.4667x^3 + -4.0401x^2 + 4.6385x + -1.8726
```

Then, the value of **f (1.09)** and **P (1.09)** was compared. Error between actual and real value is as follows:

```
P(1.09) = 0.2827 (Rounded to 4 digits)
f(1.09) = 0.2826 (Rounded to 4 digits)
Error b/w actual and calculated values: 0.0001
```

Theoretical bound for maximum error was calculated (theoretically) using the below formula:

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n)$$

Maximum Theoretical Bound on error = max(abs(f(x)-p(x))) for all x in [-1,1] = **7.34×10^{-6}**

Note: The value of error obtained in this calculation is higher than the theoretical bound on maximum error. This is due to the fact that all the float values have been rounded off to 4 digits, thereby loosing accuracy.

Q5.

Three separate programs were created, each of which accepts as inputs $(x_1, ..., x_n)$, $(f_1, ..., f_n)$, returns P (f $[x_1, ..., x_n]$), where the interpolating polynomial P (f $[x_1, ..., x_n]$) is computed using

i. the Monomial basis.
ii. the Lagrange basis.
iii. the Newton basis.

Input array $(x_1, ..., x_n)$ = (1,1.2,1.4, ……, 3)
Input array $(f_1, ..., f_n)$ = (erf (1), erf (1.2), erf (1. 4), ……, erf (3))
where erf function is:

$$erf = \frac{2}{\sqrt{\pi}} \int_0^x e^{-s^2}(s)ds$$

The coefficients of the interpolating polynomial (of order 10) were calculated (through program) for all 3 bases.
Lagrange Interpolation Polynomial is:

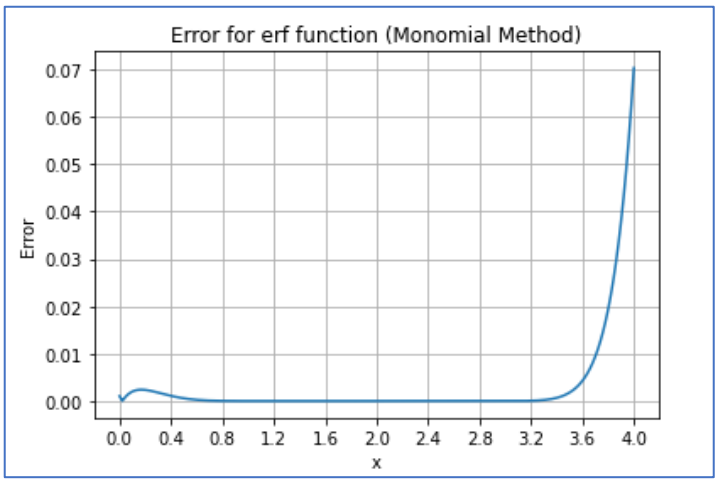
Lagrange Interpolation Polynomial is:
```
L(x) = -0.000241x^10 +0.004953x^9 -0.044174x^8 +0.221559x^7 -0.671992x^6
+1.215517x^5 -1.160967x^4 +0.397747x^3 -0.300941x^2 +1.182277x^1 -0.001038
```

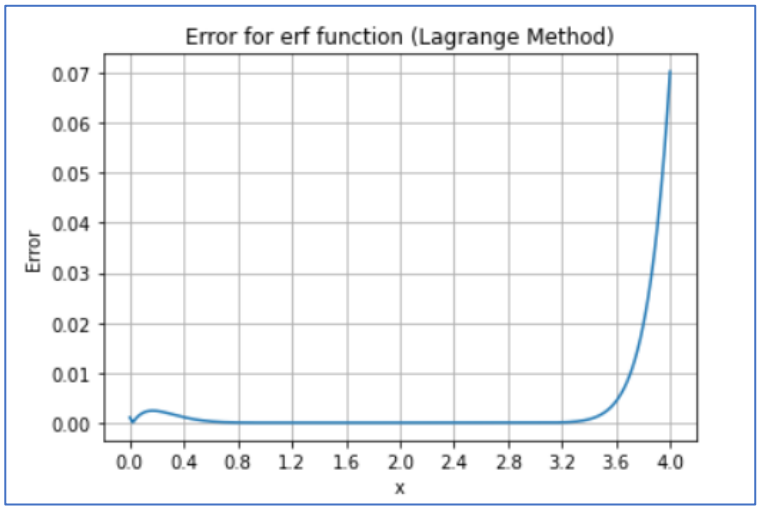
Monomial Interpolation Polynomial is:
```
L(x) = -0.000241x^10 +0.004953x^9 -0.044174x^8 +0.221559x^7 -0.671992x^6
+1.215517x^5 -1.160967x^4 +0.397747x^3 -0.300941x^2 +1.182277x^1 -0.001038
```

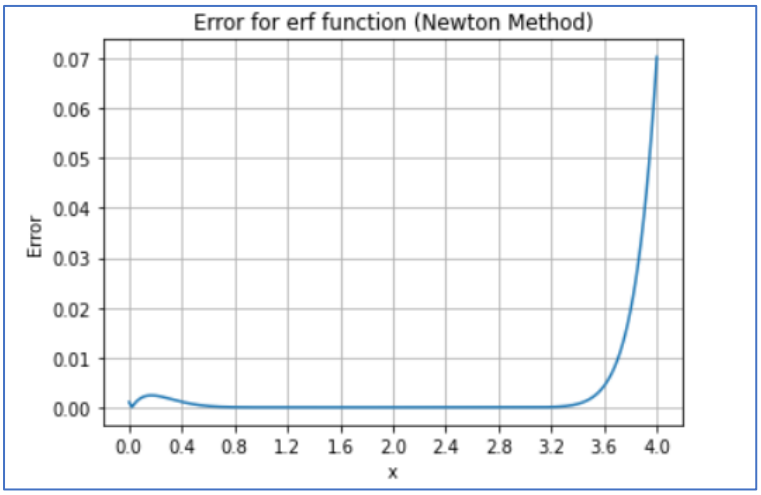
Newton Interpolation Polynomial is:
```
L(x) = -0.000241x^10 +0.004953x^9 -0.044174x^8 +0.221559x^7 -0.671992x^6
+1.215517x^5 -1.160967x^4 +0.397747x^3 -0.300941x^2 +1.182277x^1 -0.001038
```

Note:
The above coefficients were rounded up to 6 decimal values. However, while examining the values after 7 decimal points, slight differences (between various bases) were observed.

The error between the interpolating polynomial and the erf was calculated for **z = (0: 0.01: 4)** at each stage. The plotted graphs are as follows:

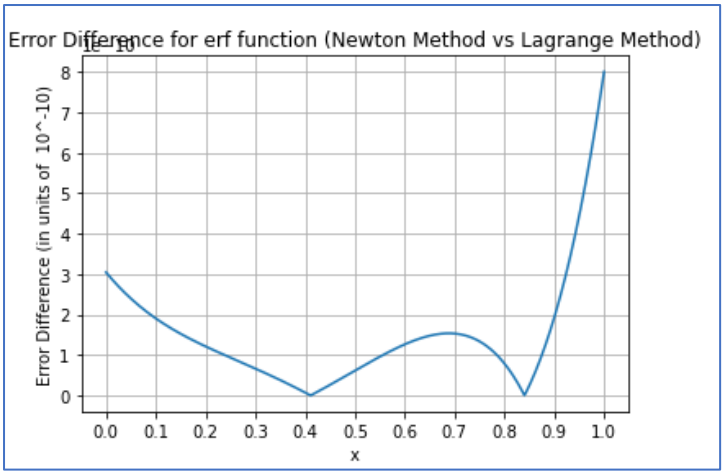


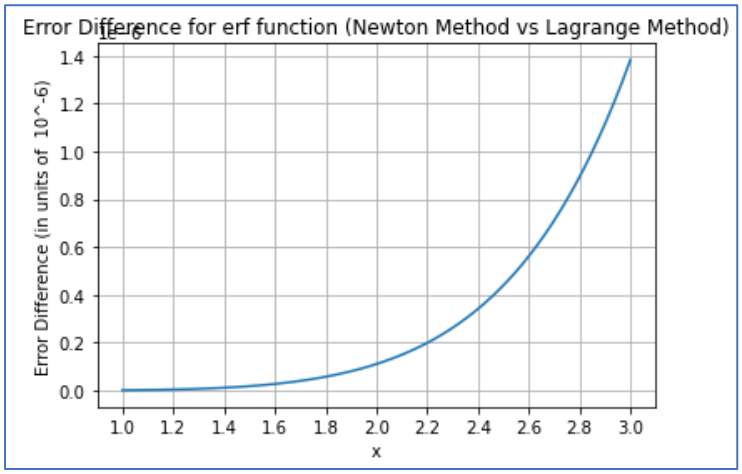




Based on the results, the error rises to almost **0.07** at **x = 4.0** for all cases. The error value is negligible for values between [1, 3]. However, for values between **[0, 0.6]** and **[3.2, 4]**, error attains comparably high values, and hence this approximate function is not recommended outside the range of [1, 3].

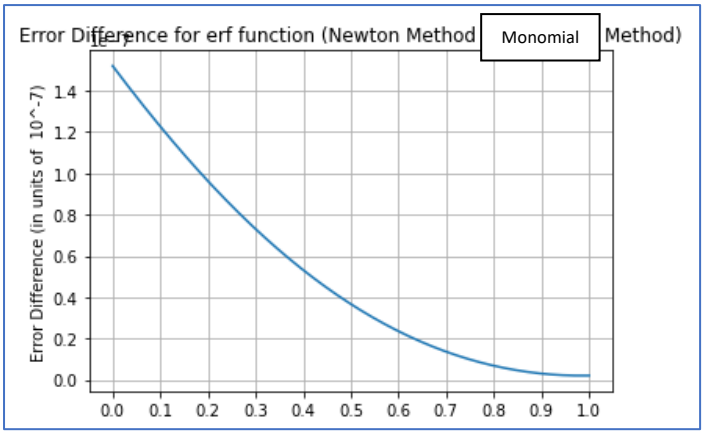
Then, the error between the interpolation polynomials calculated using the Lagrange and Newton Method between [0, 3] was calculated. The plot is as follows:



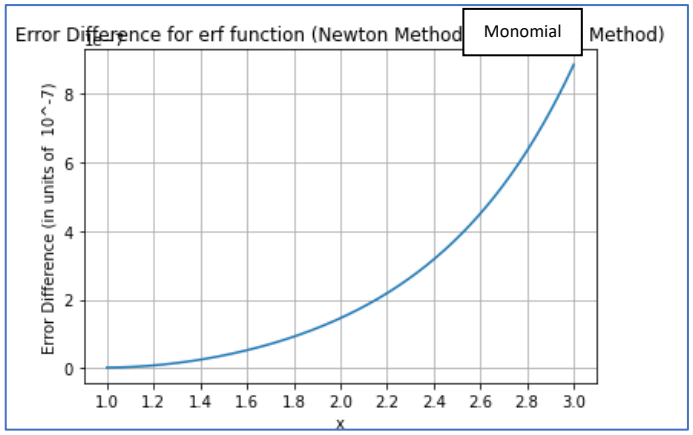


[0,1] [1,3]

Then, the error between the interpolation polynomials calculated using the Monomial and Newton Method between [0, 3] was calculated. The plot is as follows:



[0,1]



[1,3]

For the interval [0,1], it can be seen that using the Lagrange Method, the order of magnitude of error is **(-10)**, but for Lagrange method, it is **(-7)**. Hence, if we consider the range [0,1], Monomial method produces lesser errors. However, for the range [1,3], it produces Lagrange's method produces smaller errors.