Academic Year	Module	Assessment Number	Assessment Type
2023/2024	Computational Mathematics	1	Individual

Portfolio Assessment Briefing

Student ID: 2407046

Student Name: Kamal Dhital

Section: L4CG5

Module Leader: Uttam Acharya

Tutor: Basanta Singh

Submitted on: 20-05-2024

Table of Contents

1. Introduction

This portfolio assignment is for the course "4MM013: Computational Mathematics". It focuses on implementing various types of numerical methods using Python. Bisection and Newton-Raphson methods are two rood-finding methods that are implemented using Python. Furthermore, Midpoint Approximation and Trapezoidal Rule represent two numerical integrations which are also discussed in the report. This assignment needed to use NumPy and Matplotlib which demonstrates my skills and ability to use NumPy and Matplotlib effectively as it contains numerical methods such as root-finding and numerical integration. This report contains numerous tasks which are supposed to be done by us.

2. Root-Finding Method

Task - 1 is about implementing root-finding methods, particularly Bisection and Newton-Rapson. This method helps to discover all the possible roots of a given function and we will verify it with built-in functions called scipy.optimize.root() in the SciPy library. Bisection and Newton-Rapson remain two methods for recognizing roots that are implemented in this report and Python code too.

To find the roots, we have the given function:

a) $y = f(x) = x^2-x-1$
b) $y = f(x) = x^3-x^2-2x+1$

2.1 Bisection Method

A numerical approach for finding the roots of a function within the given interval if the function is continuous. Repeatedly halved the interval and then selected the subinterval where the function changes sign, indicating the presence of roots. Although it is slower than other methods, it remains a fundamental and reliable tool for root-finding tasks. With each division, the error gets smaller by the term ½^I, where I mean the number of iterations, indicating quick convergence to the true root.

Error Formula:

$|X_i – C| \leq (b-a) * 1/(2_i)$

Thus, the rate of convergence for the bisection method is 1.

4MM013: Computational Mathematics
Name: Kamal Dhital
University Student ID: 2407046

2.1.1 Plot for the Bisection Method of the Given Function:

First Function: $y = f(x) = x^2-x-1$

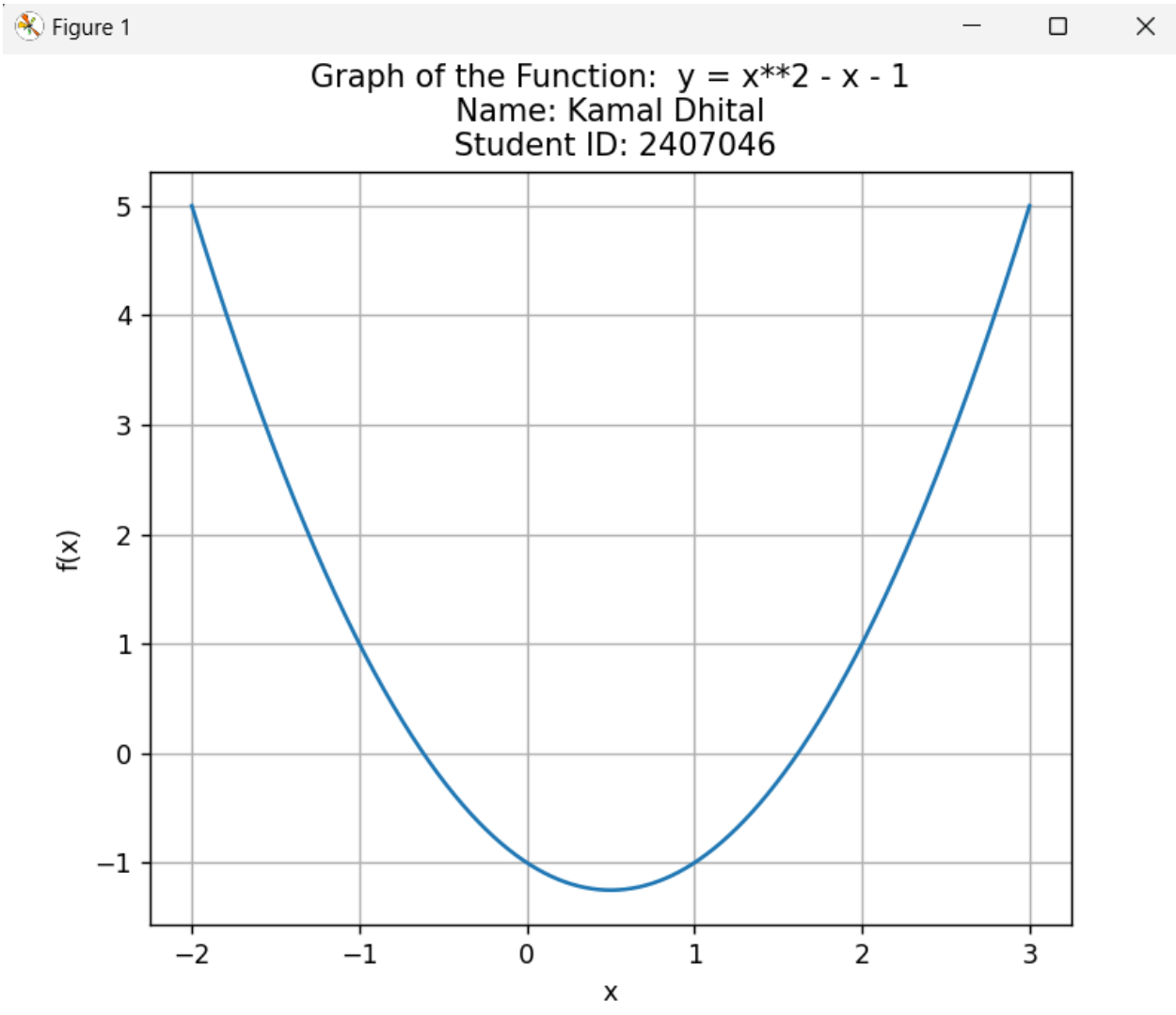
a_1 = 1

b_1 = 2

a_2 = -2

b_2 = 0

plot_function(func, -2, 3)



Second Function: y = x**3 – x**2 – 2*x + 1

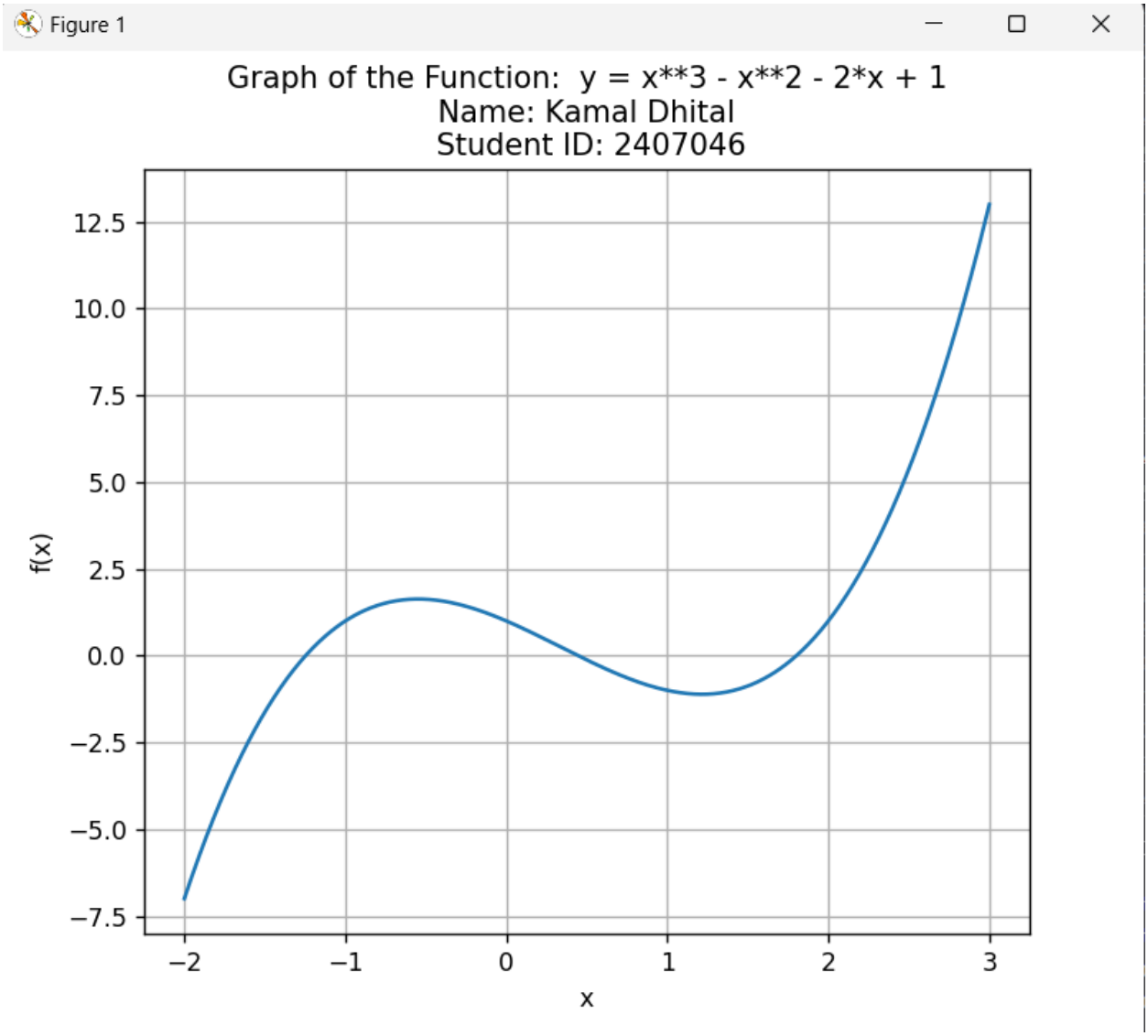
a_1 = 1

b_1 = 2

a_2 = -2

b_2 = 0

plot_function(func, -2, 3)



2.2 Newton-Rapson Method

The effective algorithm is based on linear approximation for finding successively better approximation to the roots of a real-valued function. It starts with an initial guess and improves the guess iteratively using the formula:

$$x_{n+1} = x_n - (f(x_n)/f^{-1}(x_n))$$

This method requires the function to be differentiable and may fail if the derivative is zero. Also, the initial guess should be too far from the root.

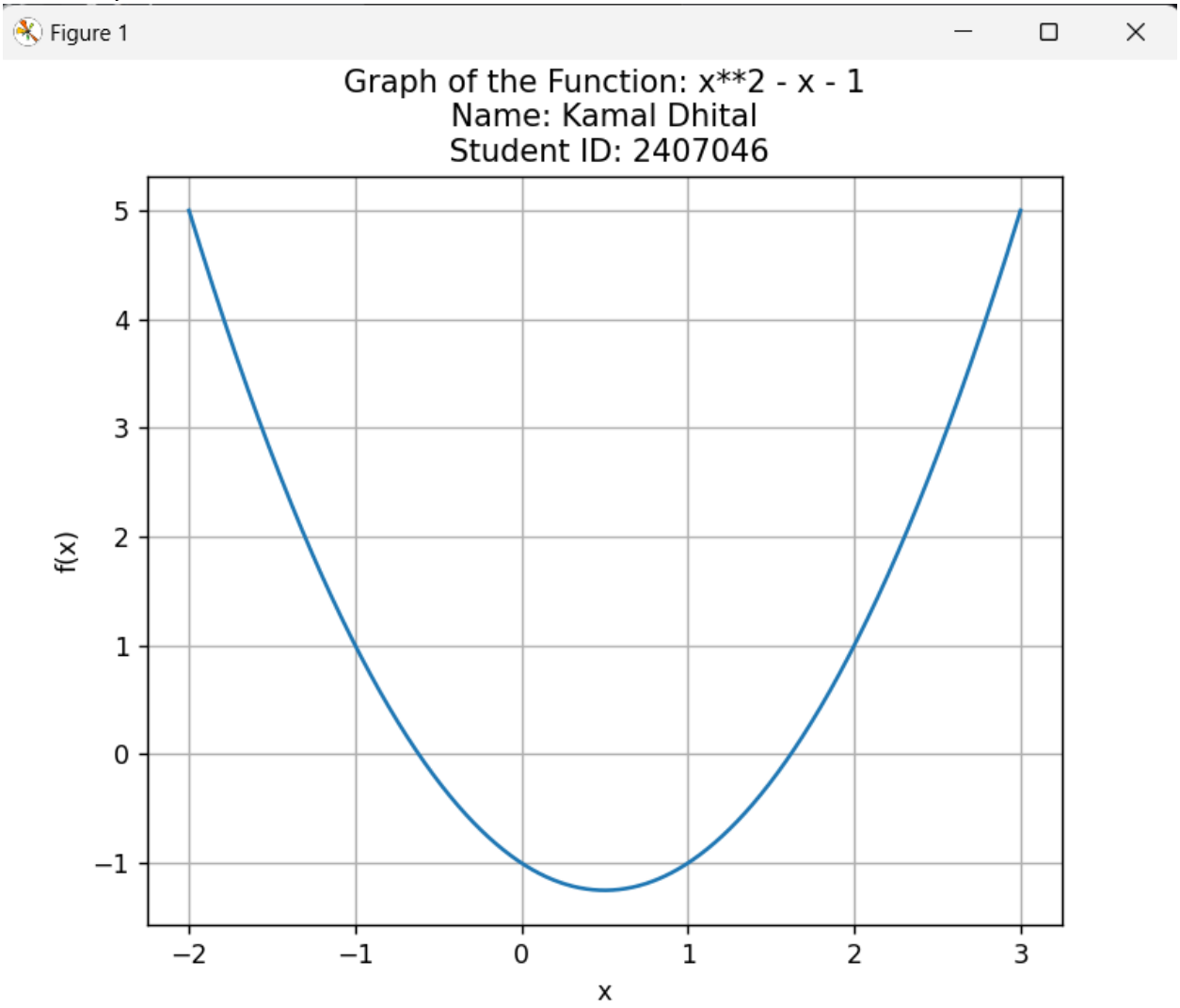
2.2.1 Plot for the Newton-Rapson Method of the Given Function:

func = lambda x: x**2 – x – 1

grad = lambda x: 2*x – 1

Root1: x0 = 0

Root2: x0 = 2

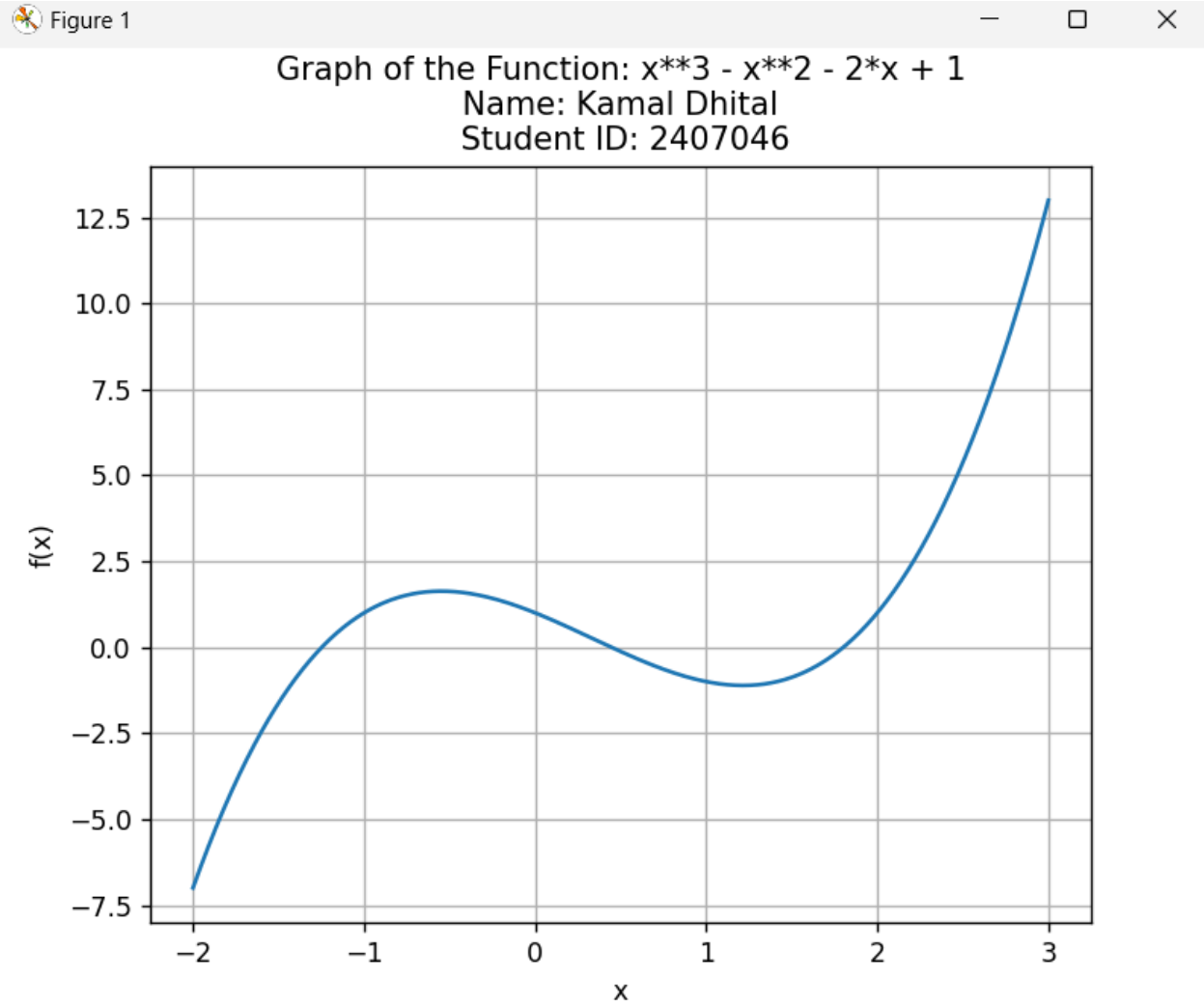


func = lambda x: x**3 – x**2 – 2*x + 1

grad = lambda x: 3*x**2 – 2*x - 2

Root1: x0 = 0

Root2: x0 = 0



2.3 Result Comparison of the Function:

3. For Function 1			
Bisection Method		Newton's Method	
[*aa*, *bb*]; #iteraons; Root1	[*aa*, *bb*]; #iteraons; Root2	*xx*0; #iteraons; Root1	*xx*0; #iteraons; Root2
[1,2]; #92; 1.61803436	[-2,0]; #57; -0.61803436	1.5; #47; 1.61803399	-1; #54; -0.61803399

SciPy Method	
$xx0$; Root1	$xx0$; Root2
1.5; 1.61803399	-1; -0.61803399

For Function 2					
Bisection Method			Newton's Method		
$[aa, bb]$; #iteraons; Root1	$[aa, bb]$; #iteraons; Root2	$[aa, bb]$; #iteraons; Root3	$xx0$; #iteraons; Root1	$xx0$; #iteraons; Root2	$xx0$; #iteraons; Root3
[1,2]; #75; 1.80193758	[-2,0]; #62; -1.24697971	[-3,-1]; #49; -1.24697971	1.5; #43; 1.80193774	-1; #88; -1.24697960	-2; #27; -1.24697960
SciPy Method					
$xx0$; Root1		$xx0$; Root2		$xx0$; Root2	
1.5; 1.80193774		-1; -1.24697960		-2; -1.24697960	

3. Numerical Integration

Task – 2 is all about working on the numerical integration of the given function and limit, comparing numerical integration methods to approximate the definite integrals of given functions. For this, we will use Midpoint Approximation and Trapezium Rule, both of which are fundamental techniques in numerical analysis for estimating the area under the curve. For this, we have the given functions and limits:

1. $y = f(x) = x/(x^2 + 1)$ and Limits [0.5]
2. $y = f(x) = e^x$ and Limits [0,5]

We are going to implement these functions to evaluate the definite integrals of the above functions over the corresponding limits and verify our answer with the Analytical results. The indefinite integrals of the above functions are:

1. $\int x / (x^2 + 1)\, dx = ½ (\log |1 + x^2| + C$

2. $\int e^x\, dx = e^x + C$

which are used to calculate analytical values.

3.1 Midpoint Approximation

A numerical method is used to estimate the value of a definite integral. It involves dividing the interval of integration into equal subintervals, calculating the functions at the midpoint of each subinterval, and then summing those values. This method provides a good balance between accuracy for many functions.

dx = (b-a) / N dx = distance between two intervals

Area= $\sum_{i=1}^{n} f(c_i) * dx$

where, c_i = functional value of midpoint

When the function is relatively smooth over the interval, this method can provide a more accurate estimate than another method.

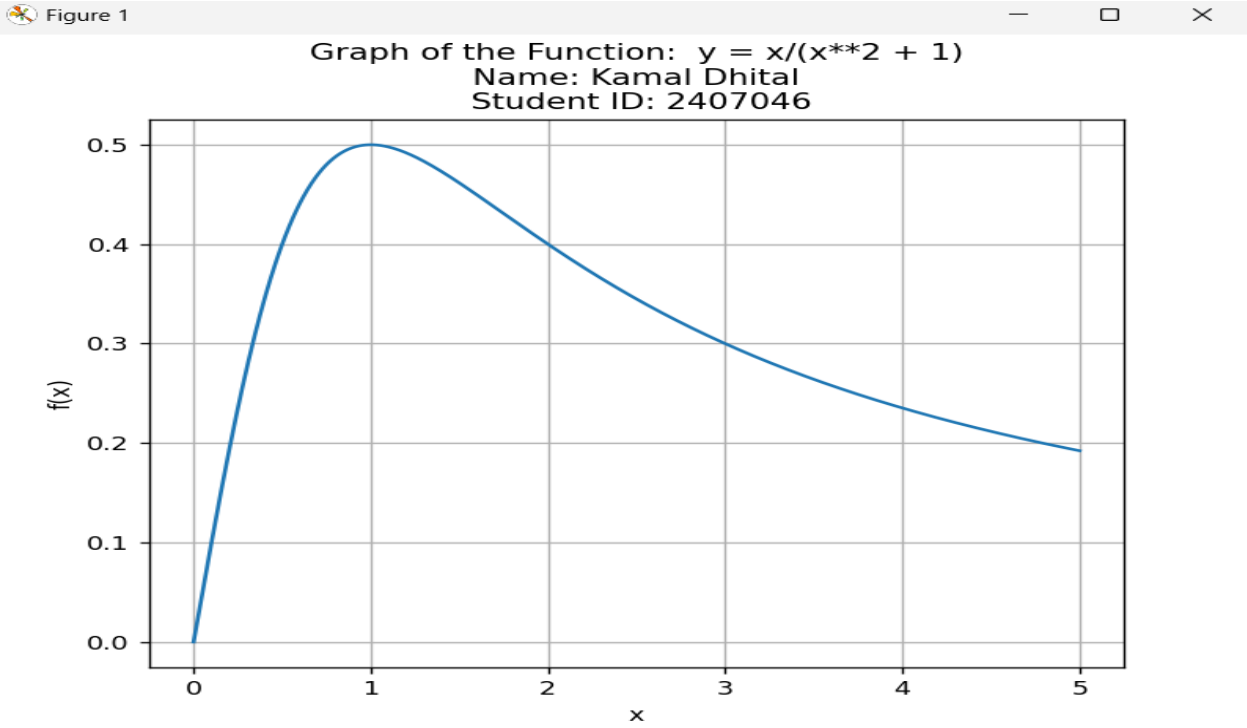
3.1.1 Plot for the Midpoint Approximation of the Given Function:

```
func_1 = lambda x: x/ (x**2 + 1)

antiderivative_1 = lambda x: 0.5 * np.log (x**2 + 1)

a1 = 0; b1 = 5;

N1 = 100
```

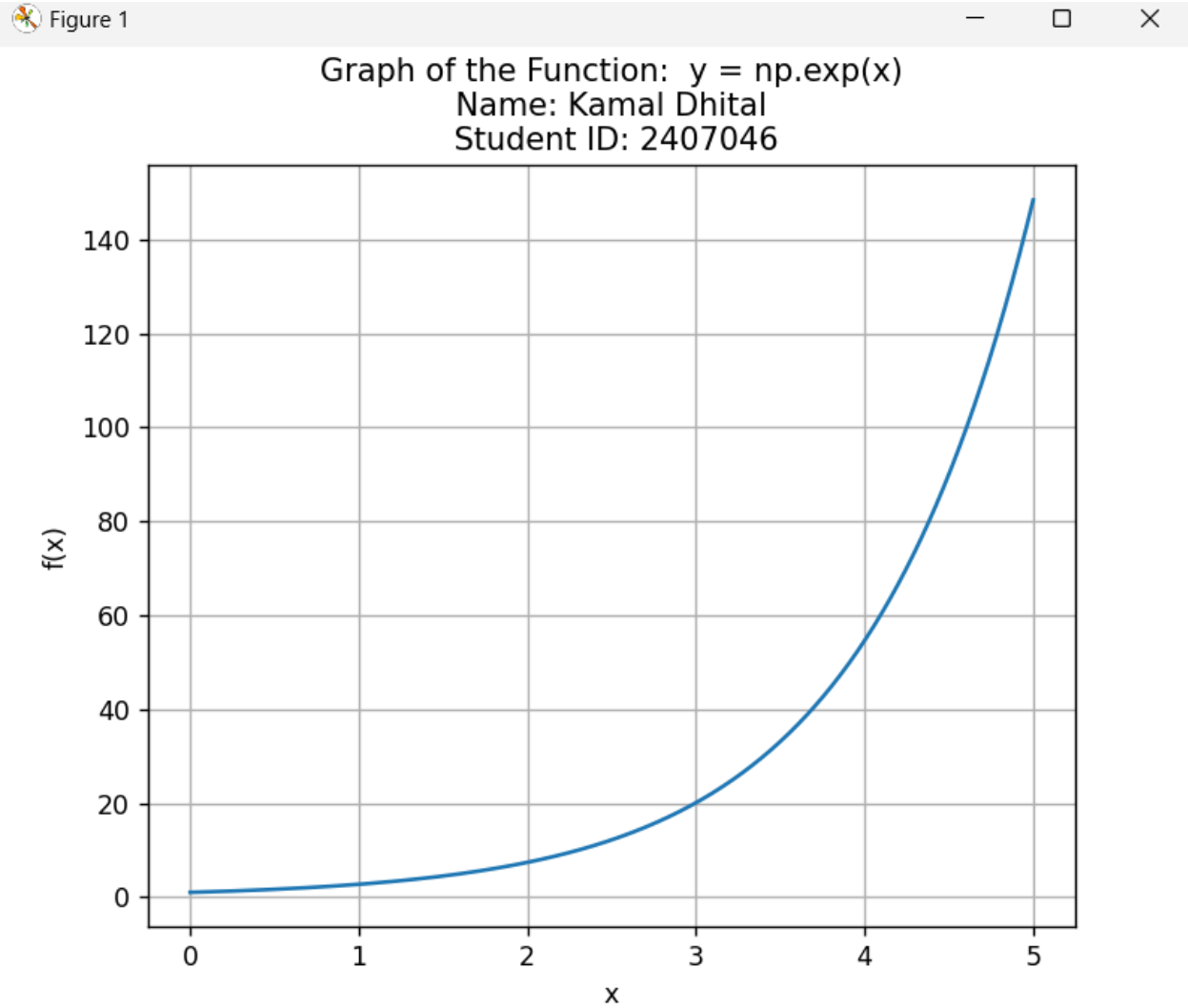


func_2 = lambda x: np.exp(x)

antiderivative_2 = lambda x: np.exp(x)

a2 = 0; b2 = 5;

N2 = 100



3.2 Trapezoid Rule

A numerical method is used to approximate the definite integral of a function. As its name, it works by dividing the area under the curve into a series of trapezoids which can provide a more accurate estimate for the integral. Functions that are linear over small intervals, provide a more accurate estimate.

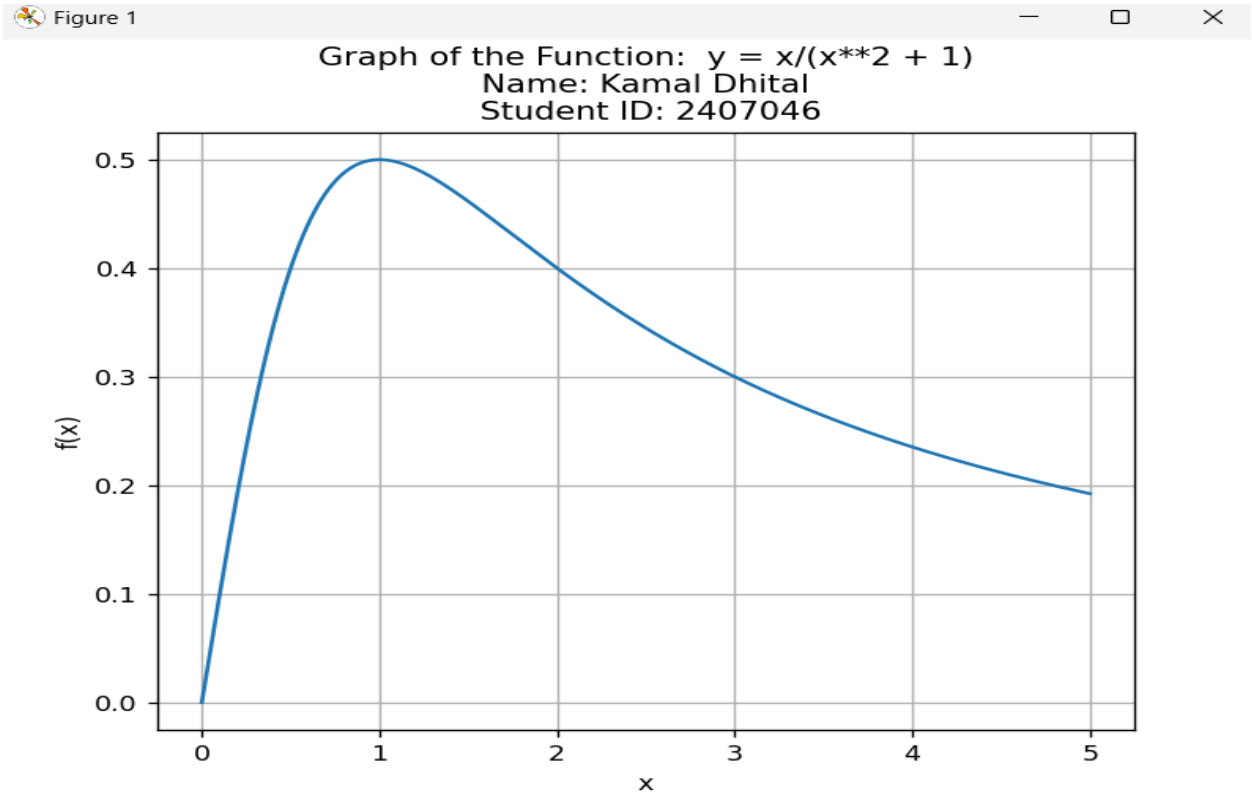
$dx = (b-a) / N$ where, N = Number of Sub-intervals

$x_i = a + i.dx$ for I = 0,1,…,N

$$\int_a^b f(x)dx \approx \frac{2}{2}\Delta x\left[f(x_0)+2\sum_{i=1}^{N-1} f(x_i)+f(x_N)\right]$$

3.2.1 Plot for the Trapezoid Rule of the Given Function:

```
func_1 = lambda x: x/ (x**2 + 1)
antiderivative_1 = lambda x: 0.5 * np.log (x**2 + 1)
a1 = 0; b1 = 5;
N1 = 100
```

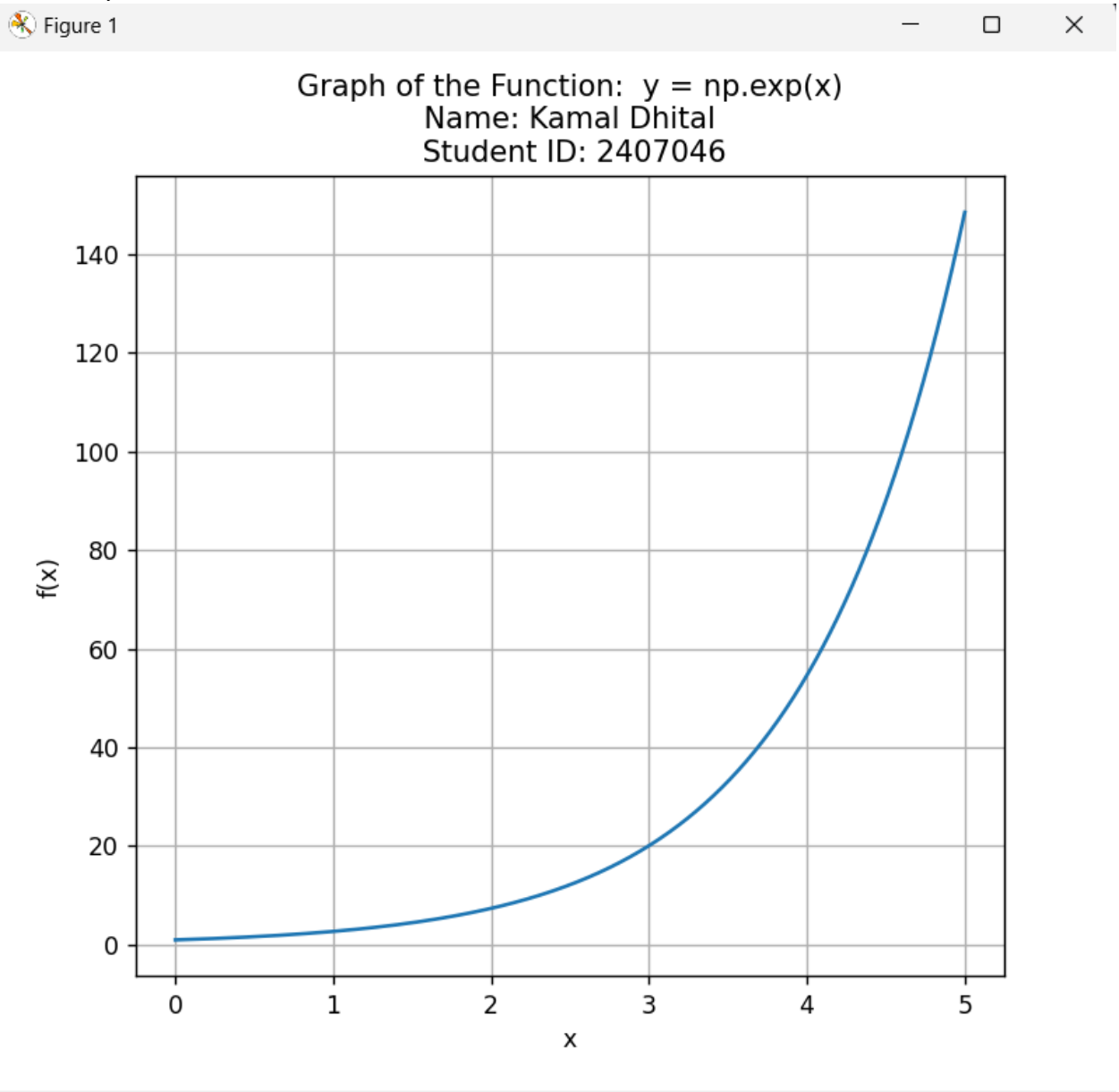



```
func_2 = lambda x: np.exp(x)

antiderivative_2 = lambda x: np.exp(x)

a2 = 0; b2 = 5;

N2 = 100
```



3.3 Result Comparison of the Function:

N	Midpoint Approximation (MM_{NN})	Trapezoidal Rule (TT_{NN})	Analytical (II)	Abs Error $\lvert MM_{NN} - II \rvert$	Abs Error $\lvert TT_{NN} - II \rvert$

For Function 1					
10	1.640378	1.606865	1.629048	0.01132983	0.02218344
30	1.630252	1.626645		0.00120420	0.00240351
50	1.629480	1.628185		0.00043219	0.00086376
100	1.629156	1.628832		0.00010791	0.00021578
500	1.629053	1.629040		0.00000431	0.00000863
For Function 2					
10	145.888729	150.471546	147.413159	1.52443031	3.05838690
30	147.242680	147.754235		0.17047895	0.34107629
50	147.351755	147.535983		0.06140424	0.12282383
100	147.397805	147.443869		0.01535442	0.03070980
500	147.412545	147.414388		0.00061422	0.00122844

3.4 Error Percentage Calculation

a. Midpoint Rule:

Formula = (Actual-Approximated)/Actual = 100%

For Function 1: (error when N=500)

Error = ((1.629048-1.629053)/1.629048) * 100%

= Approximately 0%

For Function 2: (Error when N = 500)

Error = ((147.413159 - 147.412545) / 147.413159) * 100%

= Approximately 0%

b. Trapezoidal Rule:

Formula = ((Actual-Approximated)/Actual) * 100%

For Function: (Error when N=500)

Error = ((1.629048 - 1.629040) / 1.629048) * 100%

= 0.0008%

For Function 2: (error when N=500)

Error = ((147.413159 - 147.414388) / 147.413159) * 100%

= Approximately 0%

4. Alternative Methods

4.1 Root-Finding Method:

Fixed Point Method:

A numerical technique to find the root of the function. It doesn't seek the root of the function f(x)=0, it changes the problem into finding a fixed point of a related function.

Secant Method:

An alternative technique for finding the root of the function. It finds the roots of the form f(x) = 0. It utilizes an approximation of derivatives to refine the estimated root in a loop. It is simple and efficient to use and generally faster than the bisection method.

4.2 Numerical Integration Methods:

Simpson's Rule:

A numerical technique for calculating the area under the curve which uses parabolic arcs instead of straight lines. It is more accurate than the Trapezoidal rule for the same number of intervals and it can handle non-linear functions better.

5. Conclusion

In Conclusion, this report successfully explored and implemented the key of various numerical methods for root finding and numerical integration using Python, specifically the Bisection method and Newton-Rapson method for root-finding and Midpoint Approximation and Trapezoidal Rule for numerical integration. Applying these methods to a given function and comparing them with analytical values confirms the precision of these methods. The use of Python, coupled with NumPy and Matplotlib libraries, provides a clear graphical representation of the results. The exercise has enhanced my computational skills and reinforced the practical utility of these numerical techniques.