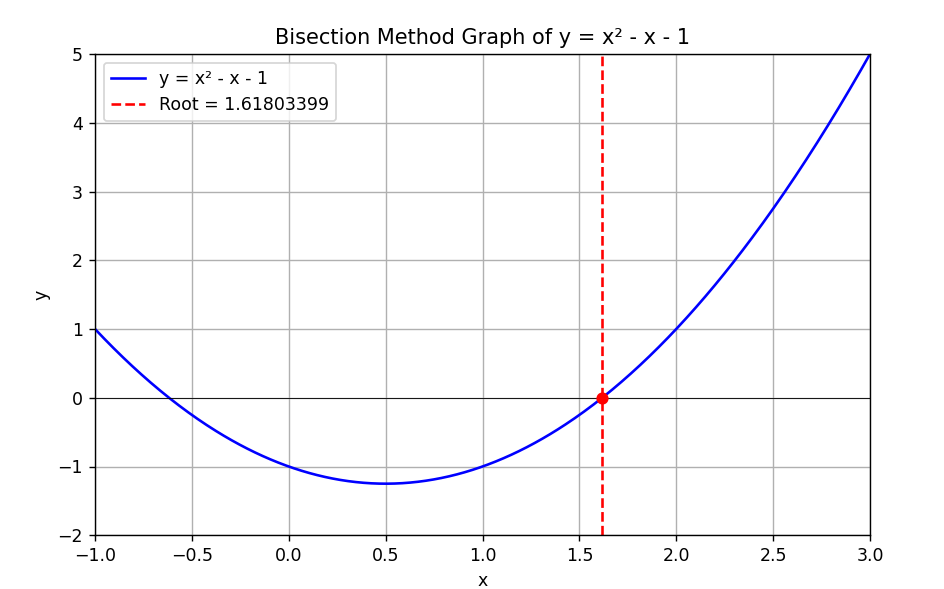
Task 1

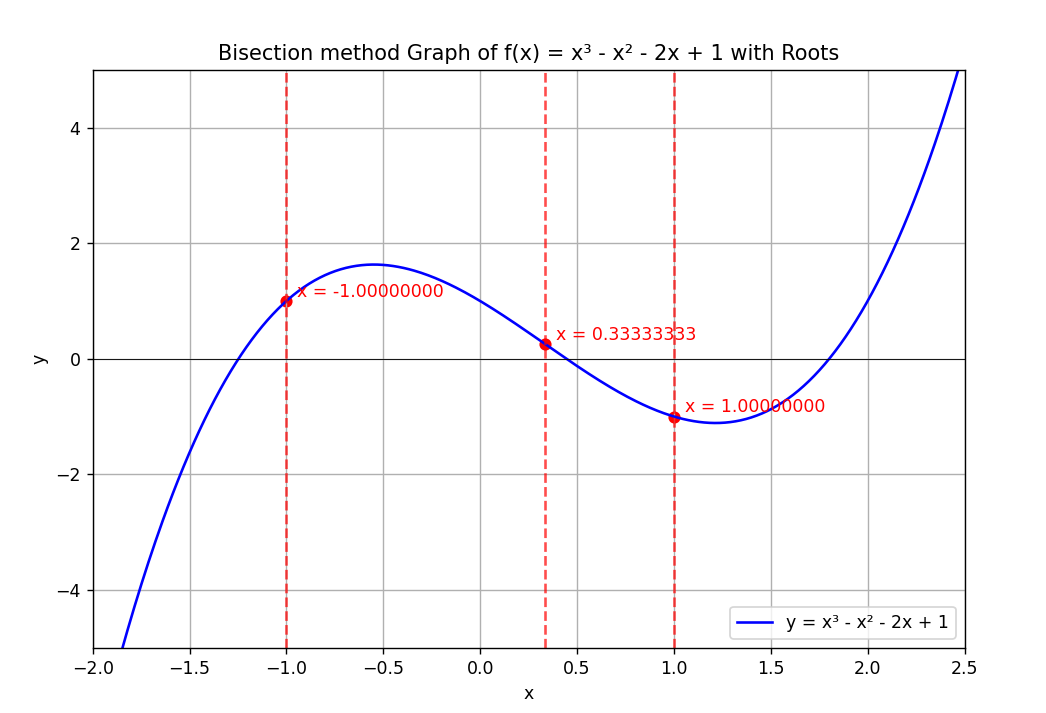
Function 1 𝑦 = 𝑓(𝑥) = 𝑥2 − 𝑥 − 1



Bisection method

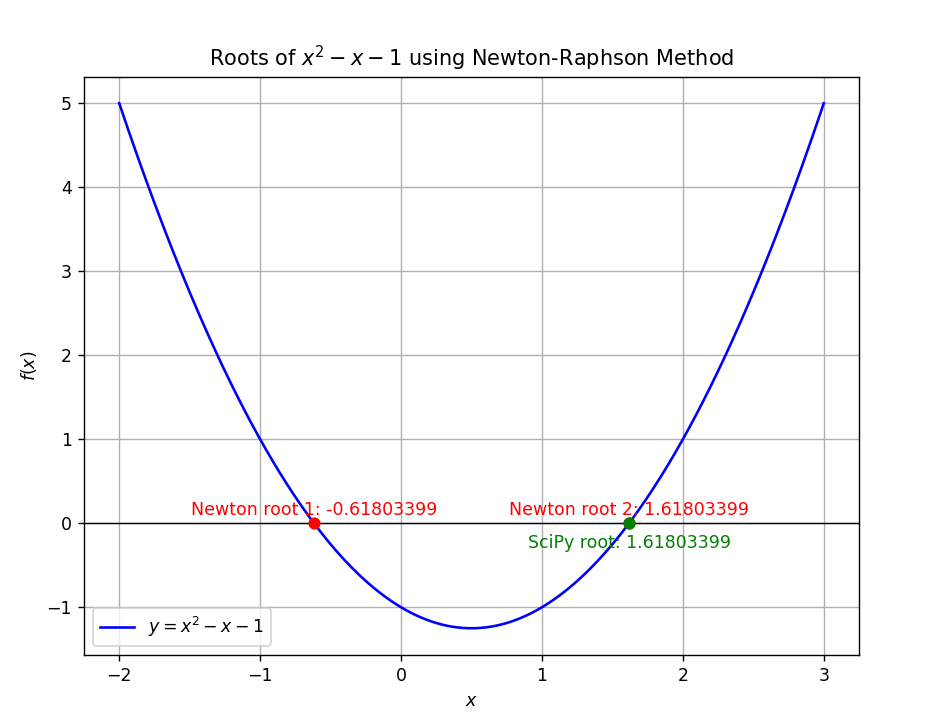
|  |  |  |  |
| --- | --- | --- | --- |
| Bisection method | interval | iterations | Root |
|  | [-2,0 | |  | | --- | |  |   21 | -o.161803436 |
|  | [1,2] | 20 | 1.61803389 |
| Scipy method |  |  | 1.61803399 |
|  |  |  |  |

Function 2 𝑦 = 𝑓(𝑥) = 𝑥3 − 𝑥2 − 2𝑥 + 1



|  |  |  |  |
| --- | --- | --- | --- |
| Bisection method | interval | iterations | Root |
|  | [-2, 1] | 20 | -1.00000024 |
|  | [0.2, 0.9] | 21 | 0.33333325 |
|  | [1.6, 2.2] | 20 | 1.00000000 |
| Scipy method |  |  | -1.00000000 |
|  |  |  | 0.33333333 |
|  |  |  | 1.00000000 |

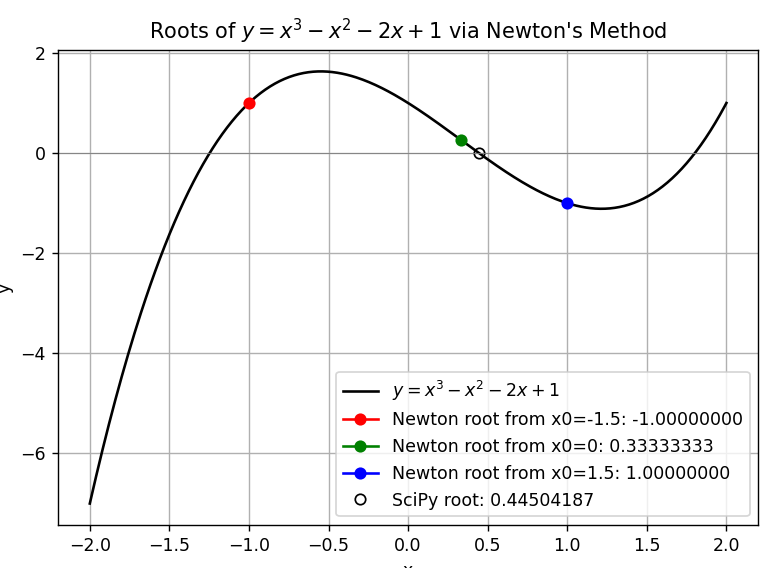
Function 1 . 𝑦 = 𝑓(𝑥) = 𝑥2 − 𝑥 – 1



Newton method

|  |  |  |  |
| --- | --- | --- | --- |
| Newton method | interval | iterations | Root |
|  | -1 | 4 | -0.61803399 |
|  | 2 | 5 | 1.61803399 |
| Scipy method |  |  |  |
|  |  |  | 1.61803399 |

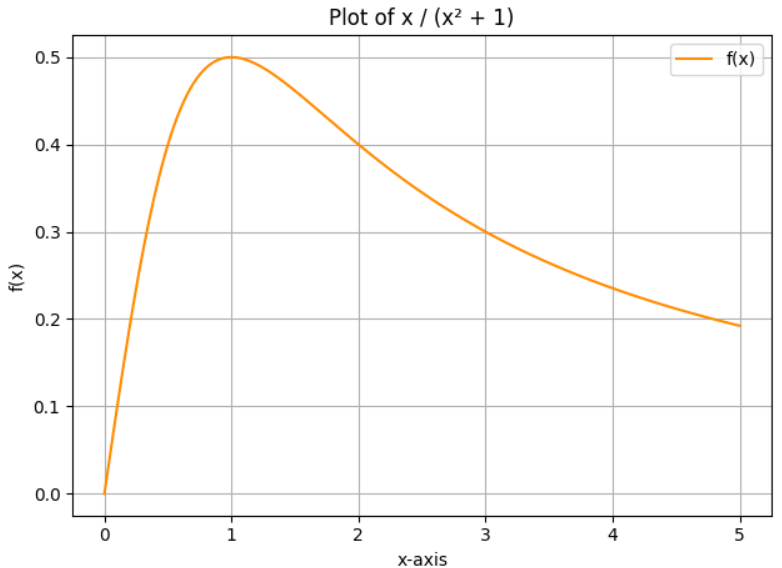
Function 2 𝑦 = 𝑓(𝑥) = 𝑥3 − 𝑥2 − 2𝑥 + 1



|  |  |  |  |
| --- | --- | --- | --- |
| newton method | interval | iterations | Root |
|  | X0=-1.5 | 5 | -1.00000000 |
|  | X0=0 | 7 | 0.33333333 |
|  | X0=1.5 | 4 | 1.00000000 |
| Scipy method |  |  | 0.33333333 |
|  |  |  |  |
|  |  |  |  |

Task 2

Function 1



A graph with a line

AI-generated content may be incorrect. Fig of Midpoint

Fig of trapezoidal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Midpoint Approximation (𝑀𝑀𝑁𝑁) | Trapezoidal Rule (𝑇𝑇𝑁𝑁) | Analytical (𝐼) | Abs Error midpoint | Abs Error TN-I |
| 10 | 0.804645 | 0.811335 | 0.804719 | 0.000074 | 0.006616 |
| 30 | 0.804708 | 0.806321 | 0.804719 | 0.000011 | 0.001602 |
| 50 | 0.804717 | 0.805350 | 0.804719 | 0.000002 | 0.000631 |
| 100 | 0.804719 | 0.805013 | 0.804719 | <0.000001 | 0.000294 |
| 500 | 0.804719 | 0.804829 | 0.804719 | <0.000001 | 0.000110 |

Function 2

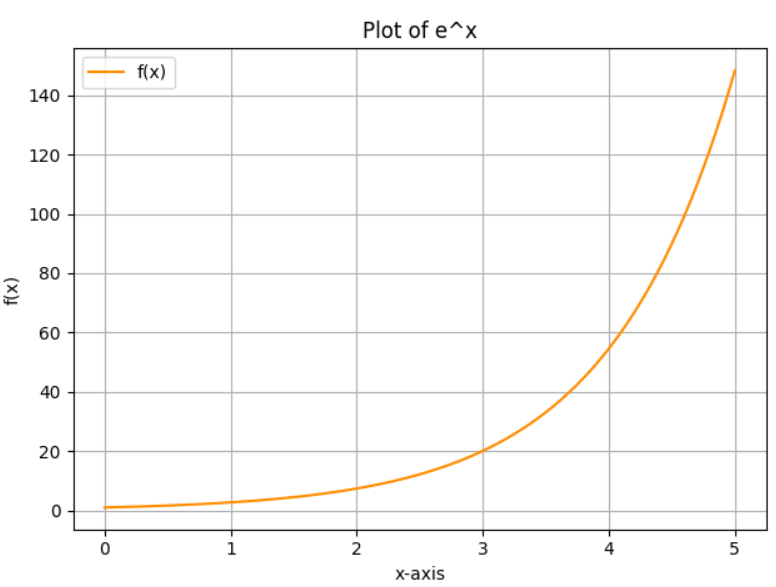


Fig of midpoint

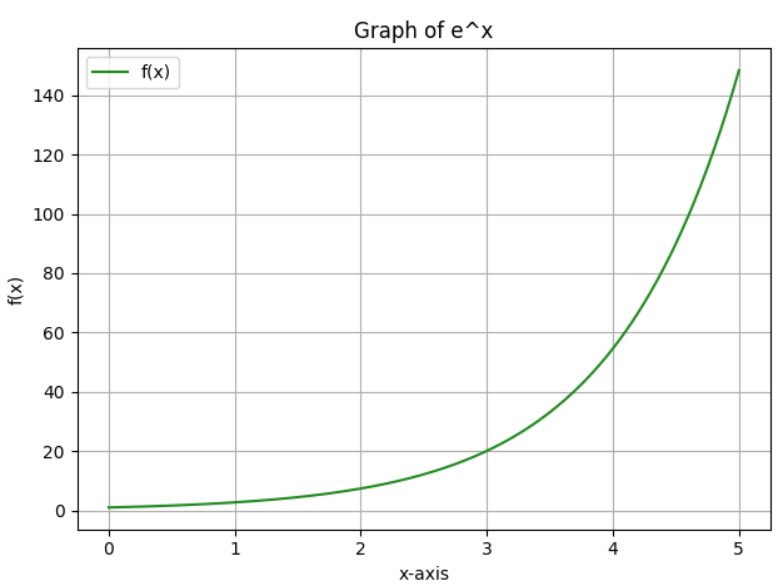


Fig of trapezoidal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Midpoint Approximation (𝑀𝑀𝑁𝑁) | Trapezoidal Rule (𝑇𝑇𝑁𝑁) | Analytical (𝐼) | Abs Error midpoint | Abs Error TN-I |
| 10 | 147.040640 | 150.401091 | 146.413159 | 0.627481 | 3.987932 |
| 30 | 146.534081 | 147.886196 | 146.413159 | 0.120922 | 1.473037 |
| 50 | 146.451656 | 147.280917 | 146.413159 | 0.038497 | 0.865778 |
| 100 | 146.426113 | 146.841239 | 146.413159 | 0.012954 | 0.428080 |
| 500 | 146.415096 | 146.513661 | 146.413159 | 0.001937 | 0.100502 |

#Root finding methods

**1. Bracketing Methods**

What these methods do is to take a pair of values, say aaa and bbb, such that the function changes its value in between (f(a)<0f(a) < 0f(a)<0 and f(b)>0f(b) > 0f(b)>0). This is a clear indication that a root is situated between the two.

**1.1. Bisection Method**

* The interval is repeatedly halved.
* We determine which part of the interval is a root and we discard all others at each step of the process.
* It is a secure process, and it leads us to the answer where a sign change is present
* Disadvantage: It is true that this method is quite a timewaster in comparison to other methods.

**1.2. False Position Method**

* Similar to bisection, but instead of the midpoint, we use a straight line between the endpoints to guess the root.
* This line often gives a better estimate than just cutting the interval in half.
* **Good**: Often faster than bisection.
* **Bad**: Sometimes it gets stuck near one endpoint and becomes slow.

**2. Open Methods**

These methods don’t need two points like bracketing ones. You usually just start with one guess (or two) and hope it gets better. They are fast but might fail if the guess is bad.

**2.1. Newton-Raphson Method**

* We use the formula:

xn+1=xn−f(xn)f′(xn)x\_{n+1} = x\_n - \frac{f(x\_n)}{f'(x\_n)}xn+1​=xn​−f′(xn​)f(xn​)​

* Basically, you draw a tangent at the current point and see where it crosses the x-axis — that’s your next guess.
* **Pros**: Very fast when close to the root.
* **Cons**: Needs the derivative of the function. Might fail if your guess is far off.

**2.2. Secant Method**

* This is like Newton-Raphson but without needing the derivative.
* It uses two previous points to draw a line and estimate the root.
* **Good**: No derivative needed and faster than bisection.
* **Not-so-good**: Not as reliable as bracketing methods.

**2.3. Fixed-Point Iteration**

* We rewrite the equation f(x)=0f(x) = 0f(x)=0 as x=g(x)x = g(x)x=g(x).
* Then we keep plugging the value back in:

xn+1=g(xn)x\_{n+1} = g(x\_n)xn+1​=g(xn​)

* **Simple** and easy to code.
* But it doesn’t always work unless the function behaves nicely.

**3. Hybrid Method**

**3.1. Brent’s Method**

* This is like the best of both worlds.
* It mixes bisection, secant, and another method called inverse quadratic interpolation.
* **Fast and reliable**, and used in real-life libraries like Python's scipy.
* The only catch is it’s a bit complex to write yourself.