cs374-lab-3-202201426

August 21, 2024

Question 1

```
[ ]: def f(x):
       return x**6 - x - 1

     def deriv(x):
       return 6*x**5 - 1


     def newton(a,max_error=1e-7,total=20):
       iteration=0
       for i in range(total):
         function_val=f(a)
         derivate_x=deriv(a)
         x_next=a-(function_val/derivate_x)
         err=abs(x_next-a)
         if(err<=max_error):
           print("iteration: ",iteration)
           return x_next
         a=x_next
         iteration=iteration+1

     root=newton(1.5)
     print("Root is: ",root)
```

```
iteration:  5
Root is:  1.1347241384015196
```

Question 2/3

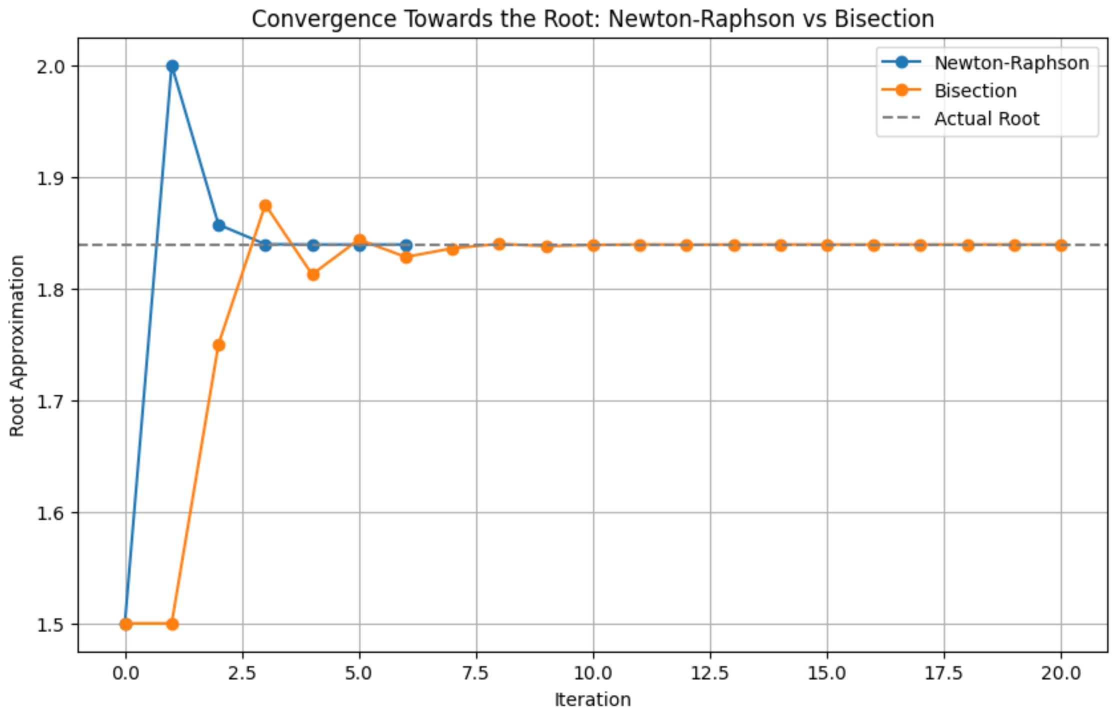
The Newton-Raphson method uses the derivative to understand how steep the function is, and adjusts its steps to reach the root faster.

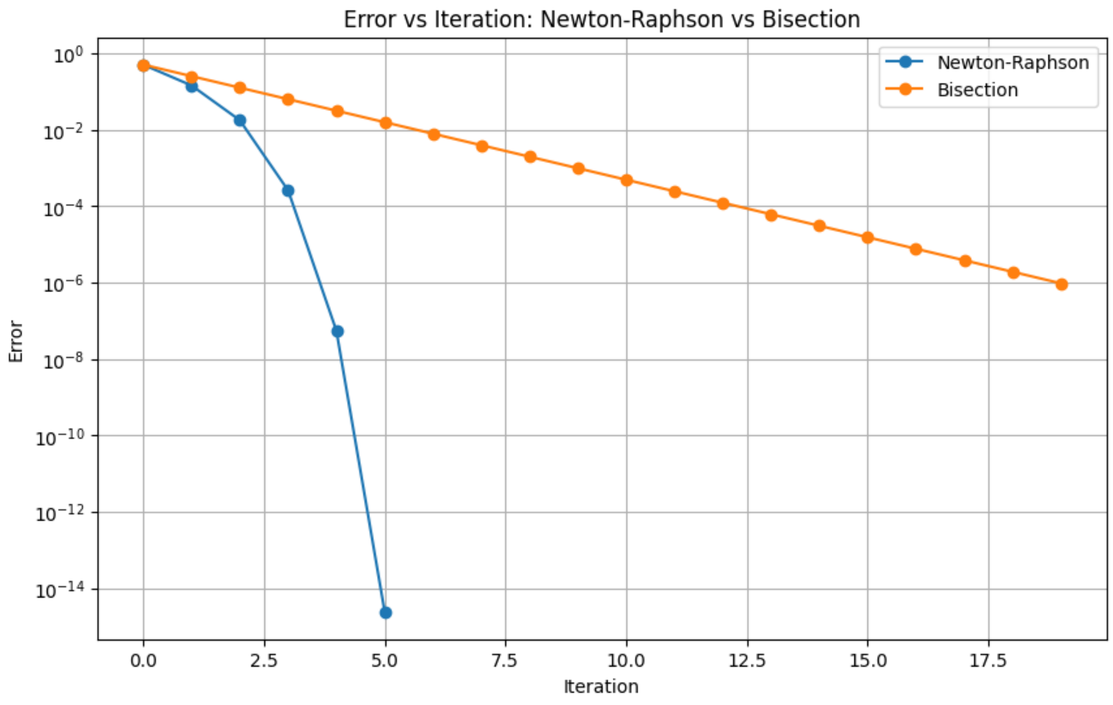
The Newton-Raphson method is a local method, meaning it converges rapidly when the initial guess is close to the root. This is because the method uses the local behavior of the function, as described by the derivative, to refine the estimate of the root.

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
```

```python
def f(x):
    return x**3 - x**2 - x - 1

def deriv(x):
    return 3*x**2 - 2*x - 1

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
    return x_next, approximations, errors

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors


a_newton = 1.5
a_bisect, b_bisect = 1.0, 2.0


root_newton, approx_newton, errors_newton = newton(a_newton)
```

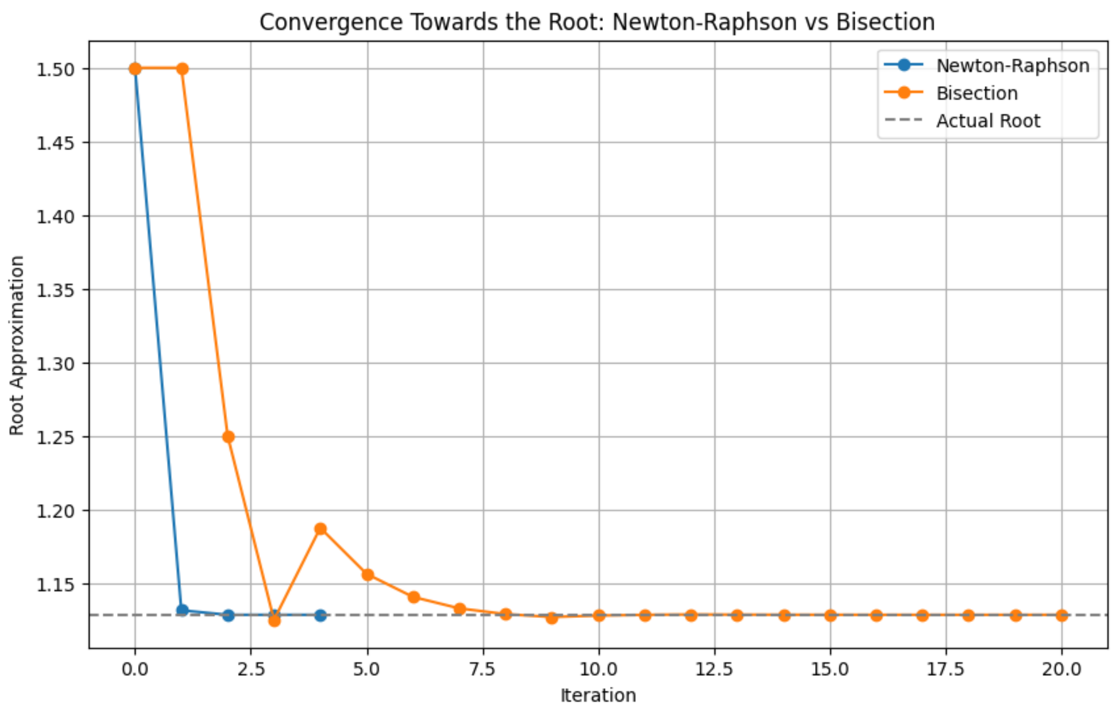
```
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
  ↪b_bisect)


plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()


plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

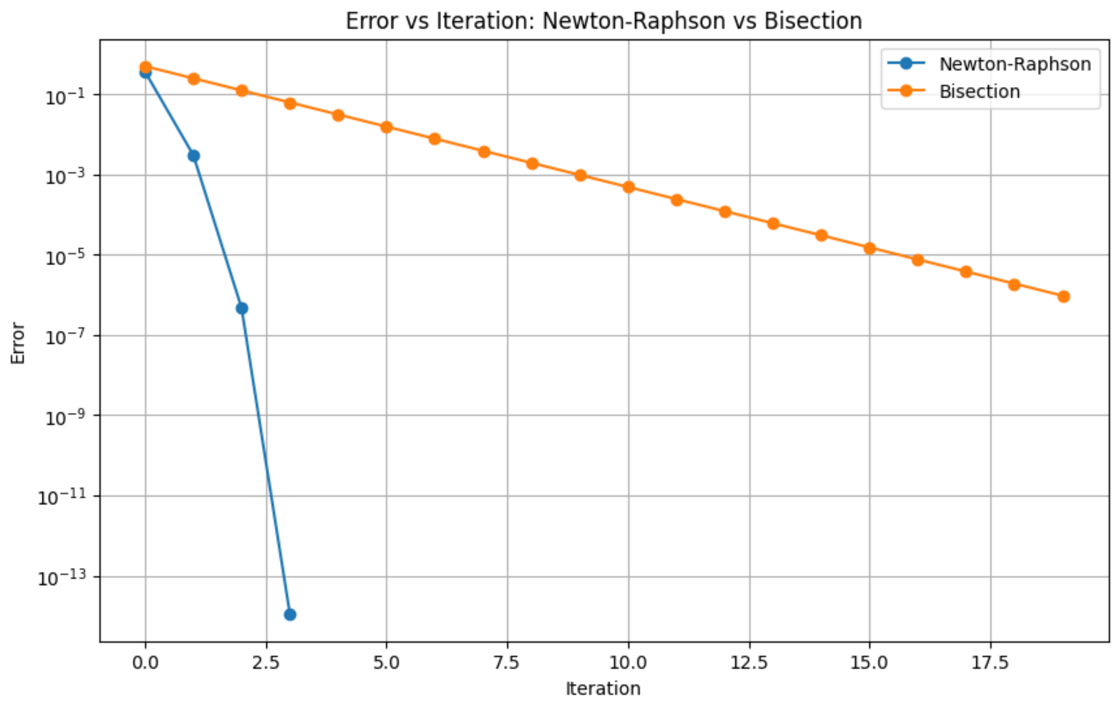




[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x - 1 - 0.3*np.cos(x)

def deriv(x):
    return 1+0.3*np.sin(x)

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton = 1.5
a_bisect, b_bisect = 1.0, 2.0

root_newton, approx_newton, errors_newton, iteration_newton = newton(a_newton)
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
 ↪b_bisect)

print("Root by Newton-Raphson method: ", root_newton)
print("Root by Bisection method: ", root_bisection)
print("Iterations by Newton-Raphson method: ", iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

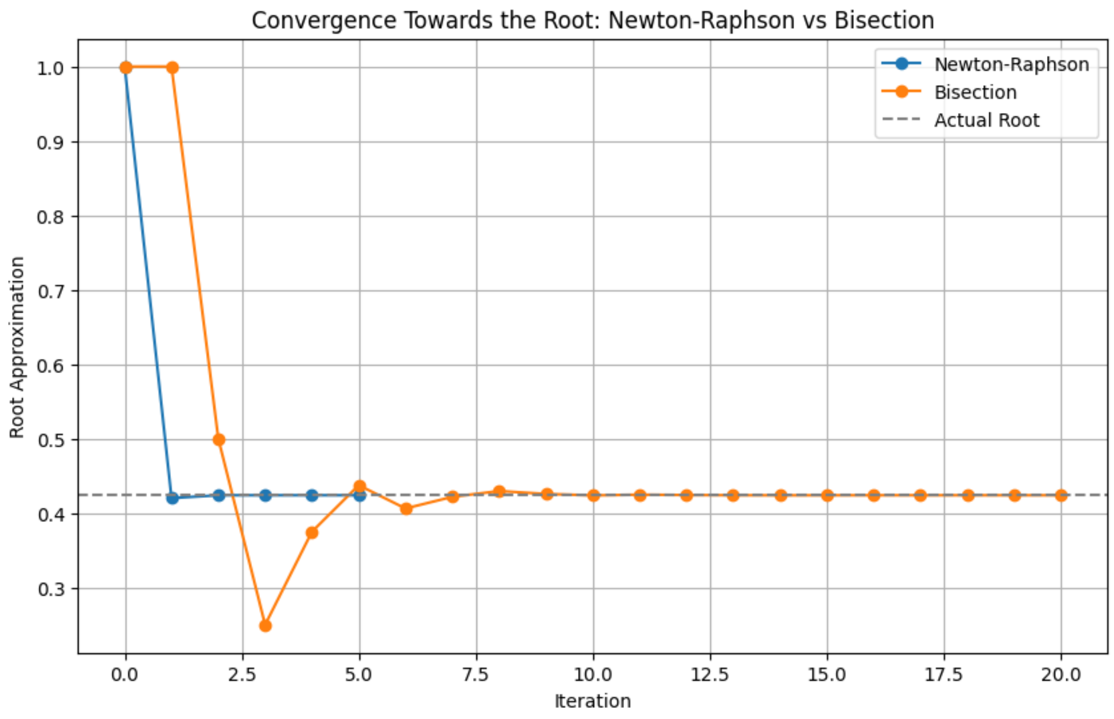
```
Root by Newton-Raphson method:  1.1284250929922246
Root by Bisection method:  1.1284246444702148
Iterations by Newton-Raphson method:  3
```

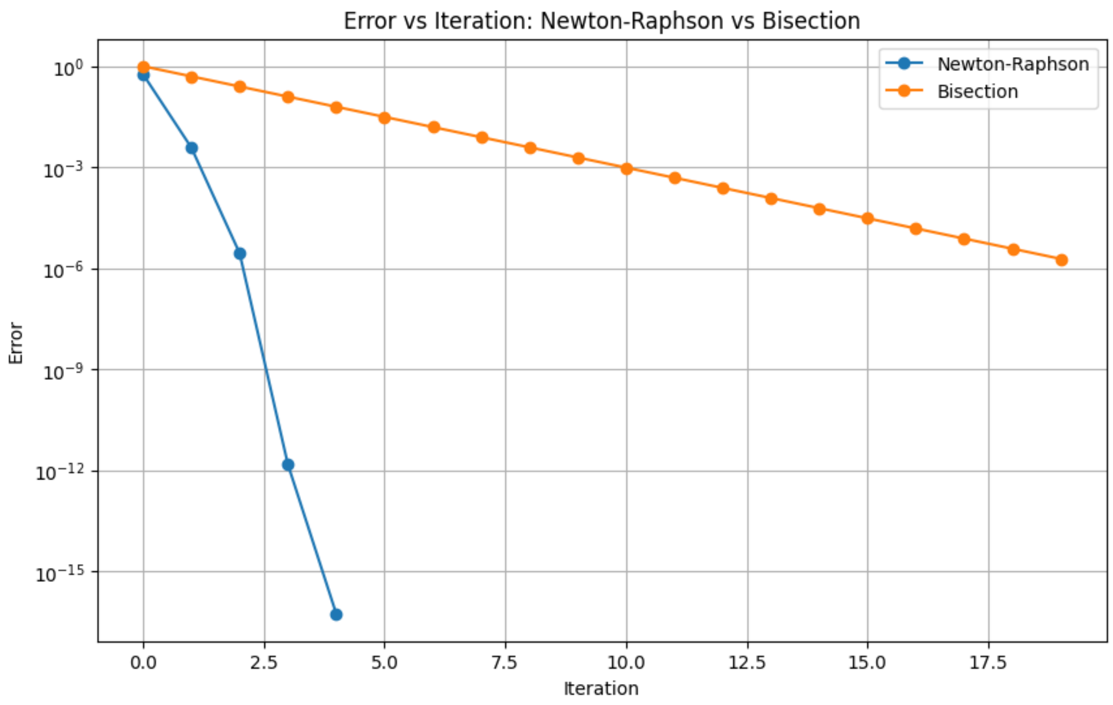




[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.cos(x) - 0.5 - np.sin(x)

def deriv(x):
    return -np.sin(x) - np.cos(x)

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton = 1
a_bisect, b_bisect = 0, 2

root_newton, approx_newton, errors_newton, iteration_newton = newton(a_newton)
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
 ↪b_bisect)

print("Root by Newton-Raphson method: ", root_newton)
print("Root by Bisection method: ", root_bisection)
print("Iterations by Newton-Raphson method: ", iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

```
Root by Newton-Raphson method:  0.42403103949074056
Root by Bisection method:  0.42403221130371094
Iterations by Newton-Raphson method:  4
```

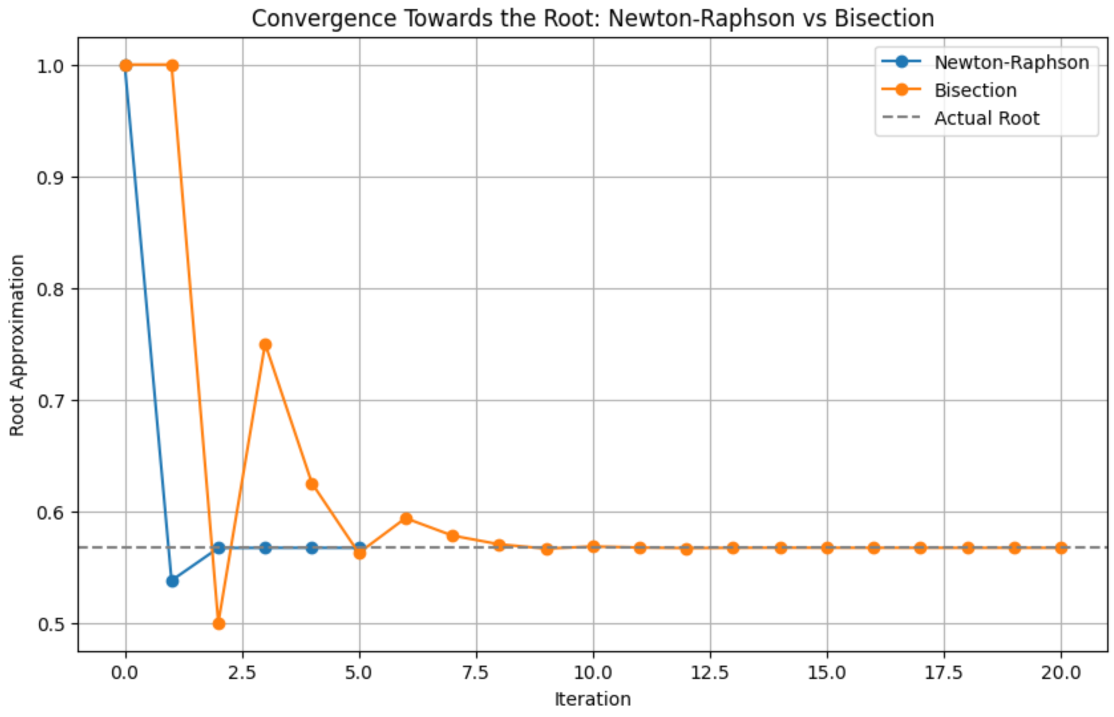


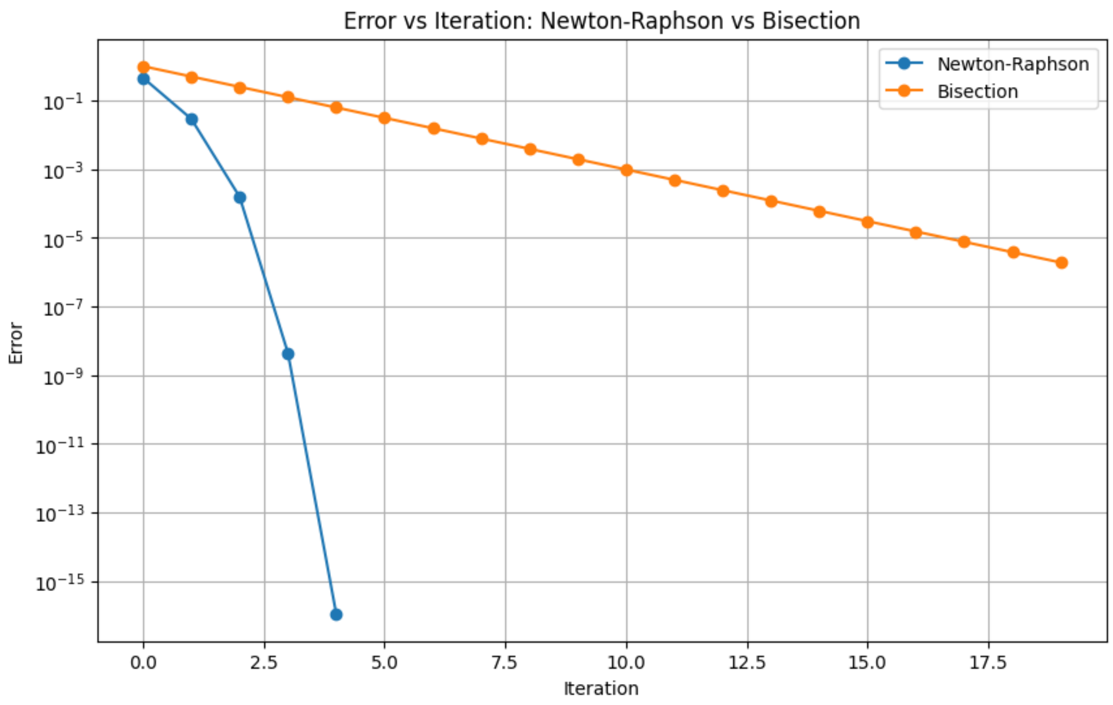


[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x - np.e**(-x)

def deriv(x):
    return 1 + np.e**(-x)

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton = 1
a_bisect, b_bisect = 0, 2

root_newton, approx_newton, errors_newton, iteration_newton = newton(a_newton)
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
  ↪b_bisect)

print("Root by Newton-Raphson method: ", root_newton)
print("Root by Bisection method: ", root_bisection)
print("Iterations by Newton-Raphson method: ", iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
```

```python
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

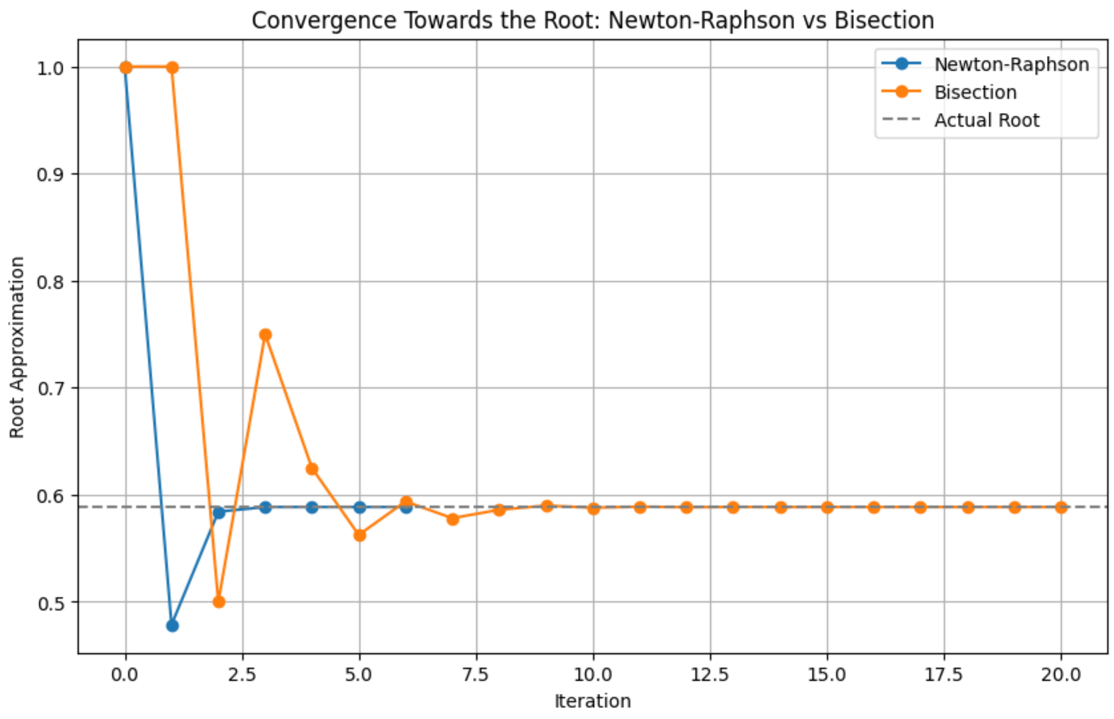
```
Root by Newton-Raphson method:  0.567143290409784
Root by Bisection method:  0.5671443939208984
Iterations by Newton-Raphson method:  4
```

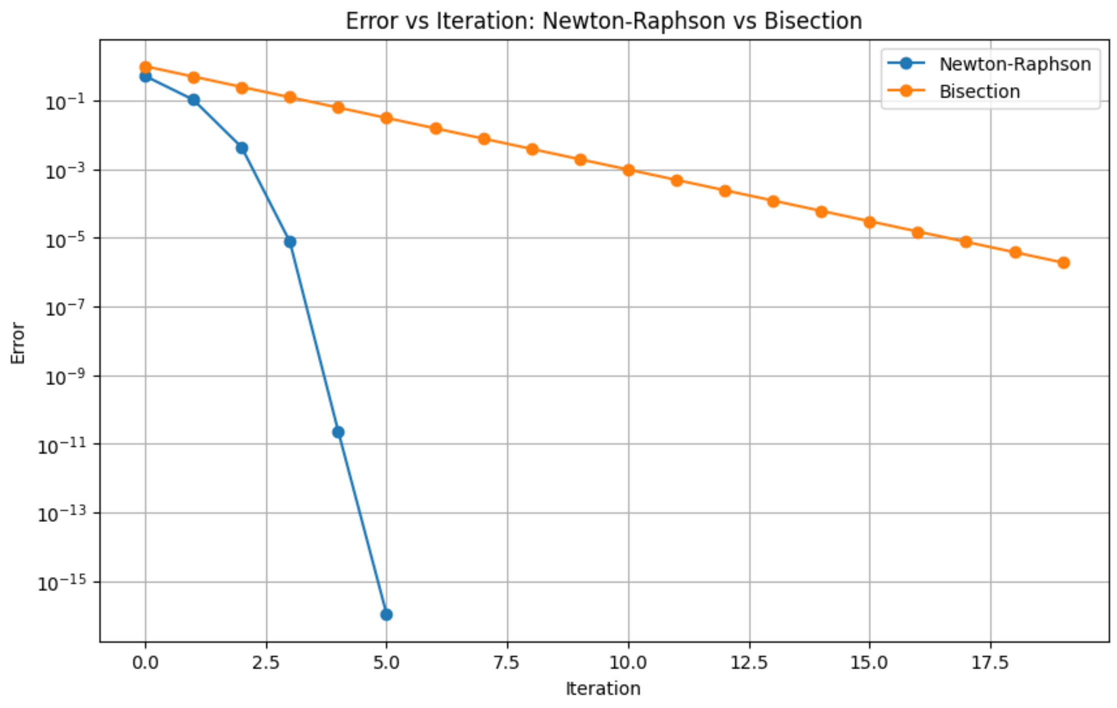




[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.e**(-x) - np.sin(x)

def deriv(x):
    return -np.e**(-x) - np.cos(x)

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton = 1
a_bisect, b_bisect = 0, 2

root_newton, approx_newton, errors_newton, iteration_newton = newton(a_newton)
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
↪b_bisect)

print("Root by Newton-Raphson method: ", root_newton)
print("Root by Bisection method: ", root_bisection)
print("Iterations by Newton-Raphson method: ", iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

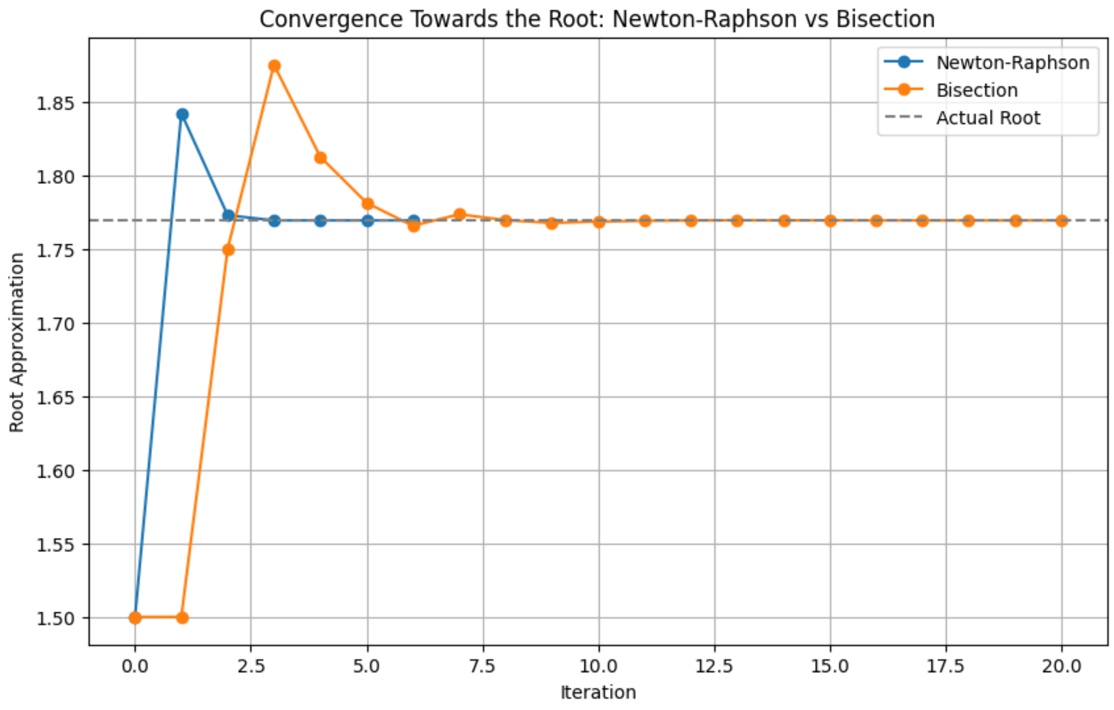
```
Root by Newton-Raphson method:  0.5885327439818611
Root by Bisection method:  0.5885334014892578
Iterations by Newton-Raphson method:  5
```

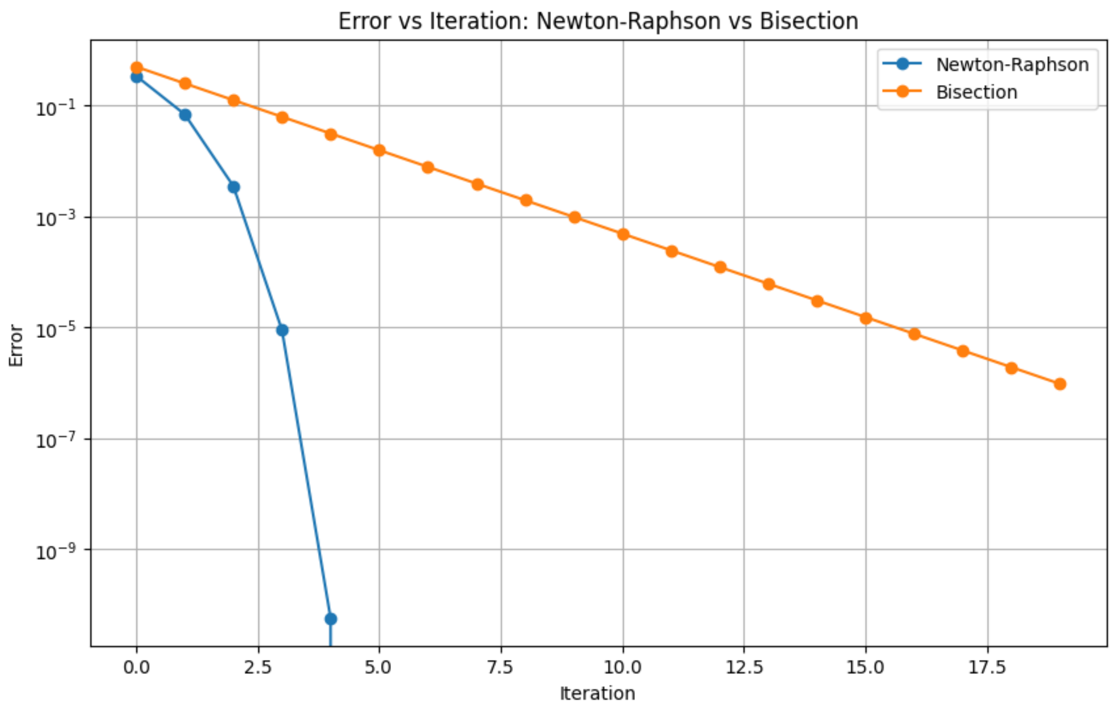




[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**3 - 2*x - 2

def deriv(x):
    return 3*(x**2) - 2

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton = 1.5
a_bisect, b_bisect = 1, 2

root_newton, approx_newton, errors_newton, iteration_newton = newton(a_newton)
root_bisection, approx_bisection, errors_bisection = bisection(a_bisect,␣
 ↪b_bisect)

print("Root by Newton-Raphson method: ", root_newton)
print("Root by Bisection method: ", root_bisection)
print("Iterations by Newton-Raphson method: ", iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton, label='Newton-Raphson', marker='o')
plt.plot(approx_bisection, label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton, label='Newton-Raphson', marker='o')
plt.plot(errors_bisection, label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

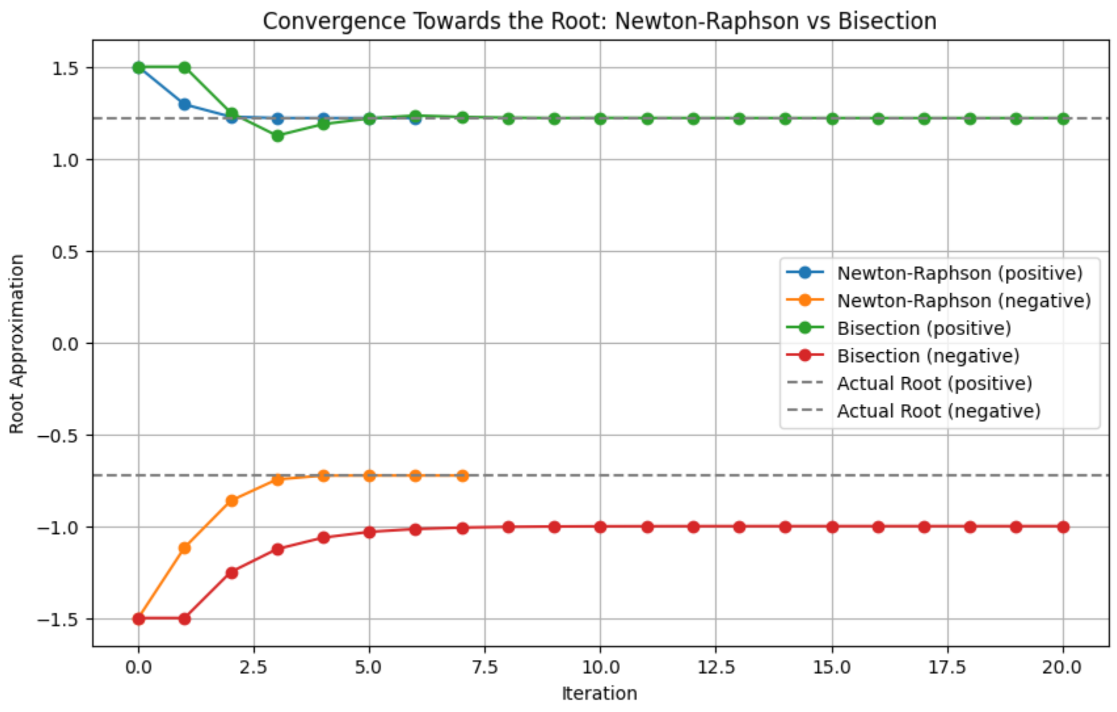
```
Root by Newton-Raphson method:  1.7692923542386314
Root by Bisection method:  1.769291877746582
Iterations by Newton-Raphson method:  5
```

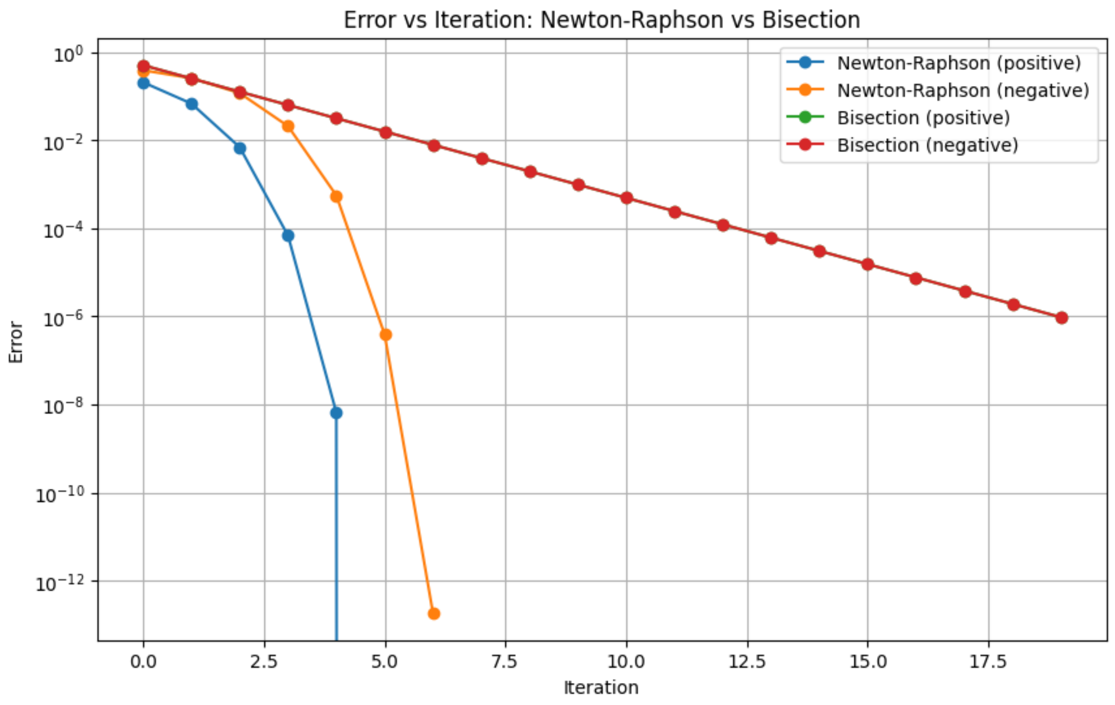




[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**4 - x - 1

def deriv(x):
    return 4*(x**3) - 1

# Newton-Raphson method
def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    iteration = 0
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

# Bisection method
def bisection(a, b, max_error=1e-12, total=20):
    approximations = [(a + b) / 2.0]
    errors = []
    for i in range(total):
        c = (a + b) / 2.0
        approximations.append(c)
        err = abs(b - a) / 2.0
        errors.append(err)
        if err <= max_error or f(c) == 0:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, approximations, errors

a_newton1 = 1.5
a_newton2 = -1.5
a_bisect1, b_bisect1 = 1, 2
a_bisect2, b_bisect2 = -2, -1

root_newton1, approx_newton1, errors_newton1, iteration_newton1 =␣
 ↪newton(a_newton1)
root_newton2, approx_newton2, errors_newton2, iteration_newton2 =␣
 ↪newton(a_newton2)
root_bisection1, approx_bisection1, errors_bisection1 = bisection(a_bisect1,␣
 ↪b_bisect1)
root_bisection2, approx_bisection2, errors_bisection2 = bisection(a_bisect2,␣
 ↪b_bisect2)

print("Root by Newton-Raphson method (positive): ", root_newton1)
print("Root by Newton-Raphson method (negative): ", root_newton2)
print("Root by Bisection method (positive): ", root_bisection1)
print("Root by Bisection method (negative): ", root_bisection2)
print("Iterations by Newton-Raphson method (positive): ", iteration_newton1)
print("Iterations by Newton-Raphson method (negative): ", iteration_newton2)

plt.figure(figsize=(10, 6))
plt.plot(approx_newton1, label='Newton-Raphson (positive)', marker='o')
plt.plot(approx_newton2, label='Newton-Raphson (negative)', marker='o')
plt.plot(approx_bisection1, label='Bisection (positive)', marker='o')
plt.plot(approx_bisection2, label='Bisection (negative)', marker='o')
plt.axhline(y=root_newton1, color='gray', linestyle='--', label='Actual Root␣
 ↪(positive)')
plt.axhline(y=root_newton2, color='gray', linestyle='--', label='Actual Root␣
 ↪(negative)')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(errors_newton1, label='Newton-Raphson (positive)', marker='o')
plt.plot(errors_newton2, label='Newton-Raphson (negative)', marker='o')
plt.plot(errors_bisection1, label='Bisection (positive)', marker='o')
plt.plot(errors_bisection2, label='Bisection (negative)', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()
```

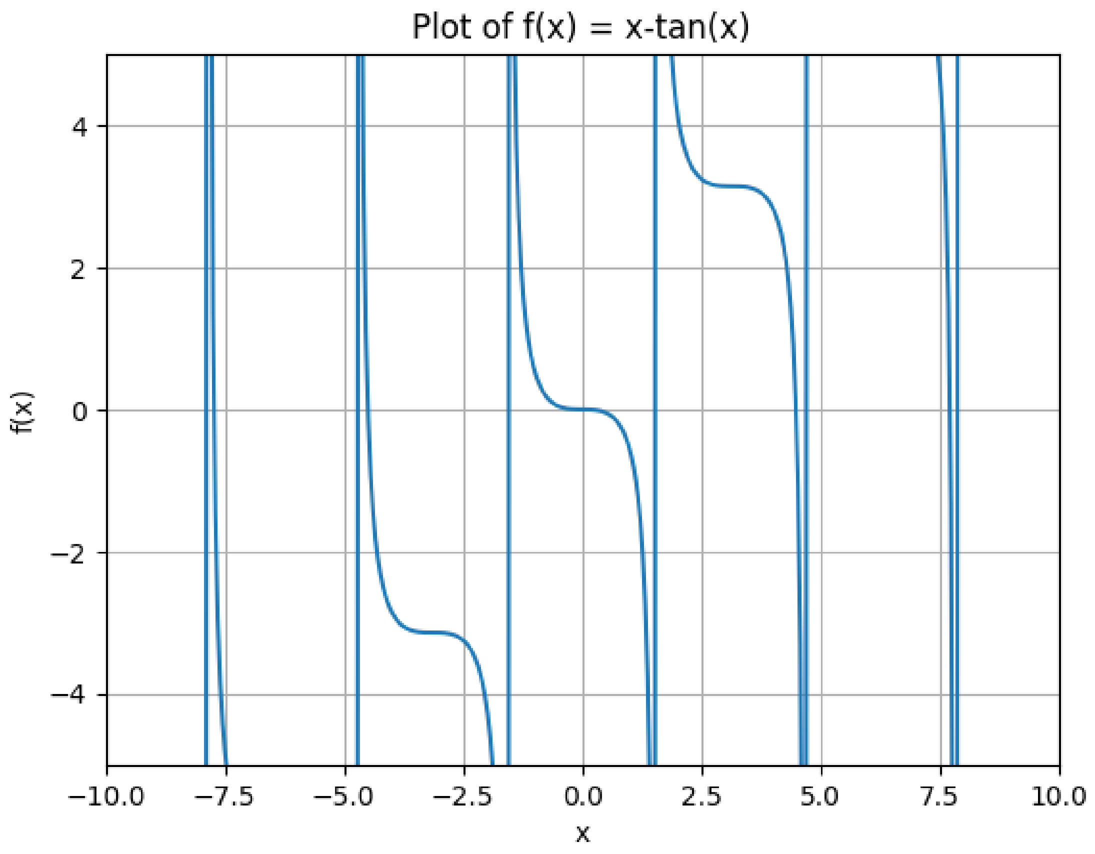
```
Root by Newton-Raphson method (positive):  1.2207440846057596
Root by Newton-Raphson method (negative):  -0.7244919590005157
Root by Bisection method (positive):  1.2207441329956055
Root by Bisection method (negative):  -1.0000009536743164
Iterations by Newton-Raphson method (positive):  5
Iterations by Newton-Raphson method (negative):  6
```

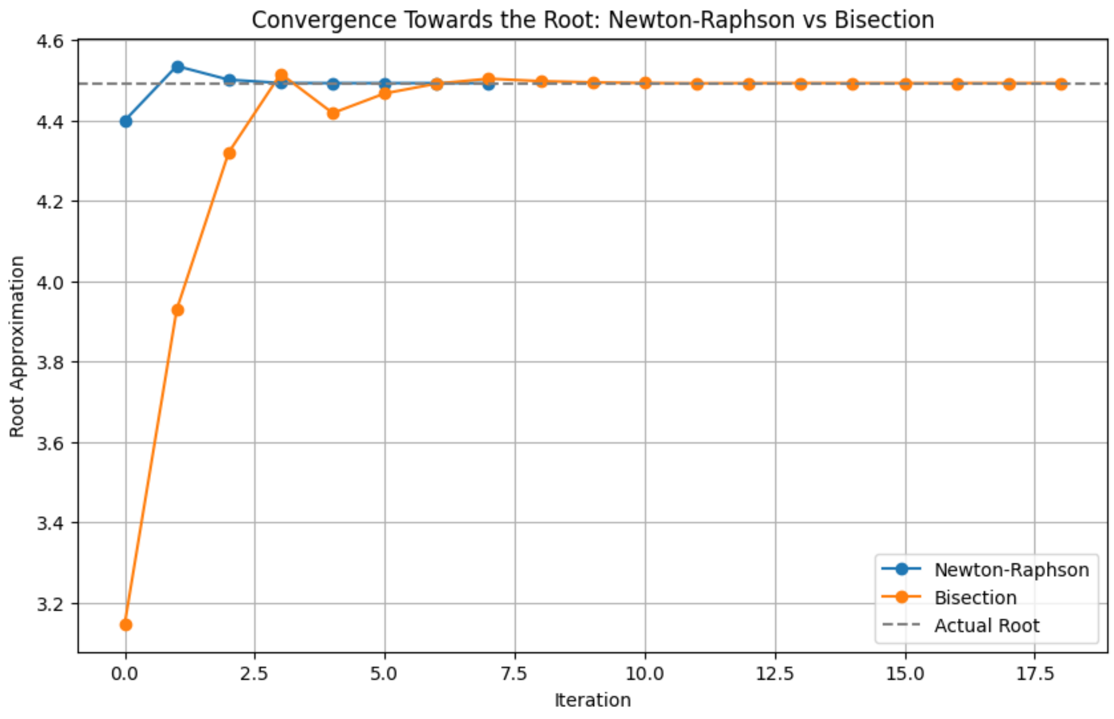

Convergence Towards the Root: Newton-Raphson vs Bisection
Root Approximation
Iteration
Newton-Raphson (positive)
Newton-Raphson (negative)
Bisection (positive)
Bisection (negative)
Actual Root (positive)
Actual Root (negative)

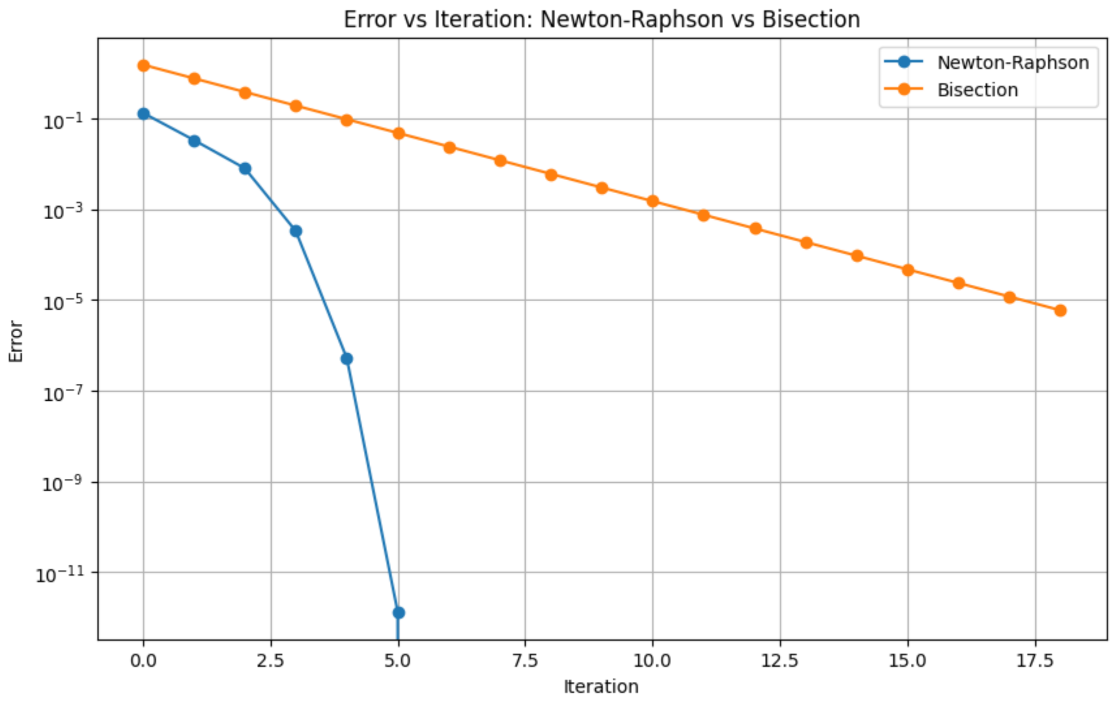

Error vs Iteration: Newton-Raphson vs Bisection
Error
Iteration
Newton-Raphson (positive)
Newton-Raphson (negative)
Bisection (positive)
Bisection (negative)

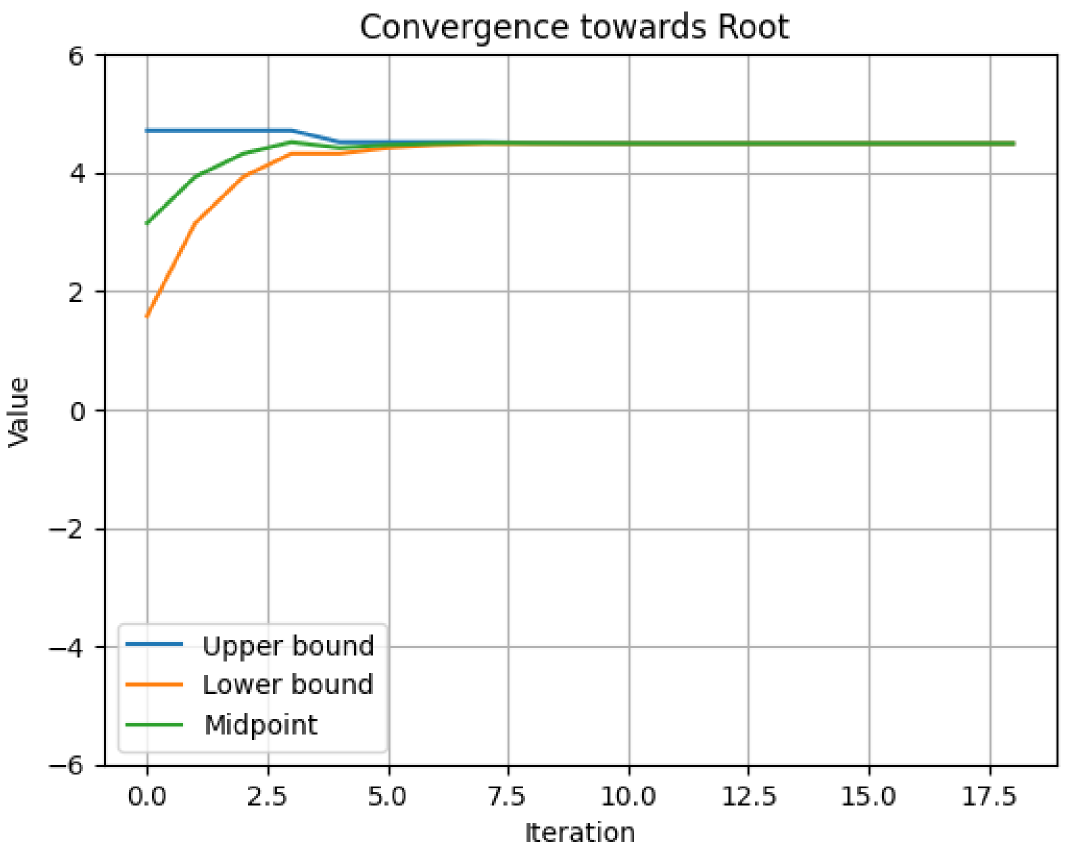
[3]:
```python
import matplotlib.pyplot as plt
import numpy as np

def f(x):
    return x - np.tan(x)

def deriv(x):
    return 1 - (1 / np.cos(x))**2

def newton(a, max_error=1e-12, total=20):
    approximations = [a]
    errors = []
    for i in range(total):
        function_val = f(a)
        derivate_x = deriv(a)
        x_next = a - (function_val / derivate_x)
        approximations.append(x_next)
        err = abs(x_next - a)
        errors.append(err)
        if err <= max_error:
            break
        a = x_next
    return x_next, approximations, errors

def bisect(a, b, max_error=1e-5, total=100):
    conv_table = []
    for i in range(total):
        c = (a + b) / 2
        conv_table.append([i, a, b, c, f(c)])
        if f(c) == 0 or (b - a) / 2 < max_error:
            return c, conv_table
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2, conv_table

x = np.linspace(-10, 10, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x-tan(x)')
plt.ylim(-5,5)
plt.xlim(-10,10)
plt.grid(True)
plt.show()
```

```python
a_newton = 4.4
a_bisect, b_bisect = (np.pi)/2 + 0.01, 1.5*(np.pi)

root_newton, approx_newton, errors_newton = newton(a_newton)
root_bisection, conv_table2 = bisect(a_bisect, b_bisect)

plt.figure(figsize=(10, 6))
plt.plot([i for i in range(len(approx_newton))], approx_newton,␣
 ↪label='Newton-Raphson', marker='o')
plt.plot([row[0] for row in conv_table2], [row[3] for row in conv_table2],␣
 ↪label='Bisection', marker='o')
plt.axhline(y=root_newton, color='gray', linestyle='--', label='Actual Root')
plt.xlabel('Iteration')
plt.ylabel('Root Approximation')
plt.title('Convergence Towards the Root: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot([i for i in range(len(errors_newton))], errors_newton,␣
 ↪label='Newton-Raphson', marker='o')
plt.plot([row[0] for row in conv_table2], [(row[2] - row[1]) / 2 for row in␣
 ↪conv_table2], label='Bisection', marker='o')
plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error vs Iteration: Newton-Raphson vs Bisection')
plt.legend()
plt.grid(True)
plt.show()

plt.plot([row[0] for row in conv_table2], [row[2] for row in conv_table2],␣
 ↪label='Upper bound')
plt.plot([row[0] for row in conv_table2], [row[1] for row in conv_table2],␣
 ↪label='Lower bound')
plt.plot([row[0] for row in conv_table2], [row[3] for row in conv_table2],␣
 ↪label='Midpoint')
plt.xlabel('Iteration')
plt.ylabel('Value')
plt.title('Convergence towards Root ')
plt.legend()
plt.ylim(-6,6)
plt.grid(True)
plt.show()
```


Plot of f(x) = x-tan(x)
f(x)
x
4
2
0
−2
−4
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Convergence Towards the Root: Newton-Raphson vs Bisection
Root Approximation
Iteration
4.6
4.4
4.2
4.0
3.8
3.6
3.4
3.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Newton-Raphson
Bisection
Actual Root

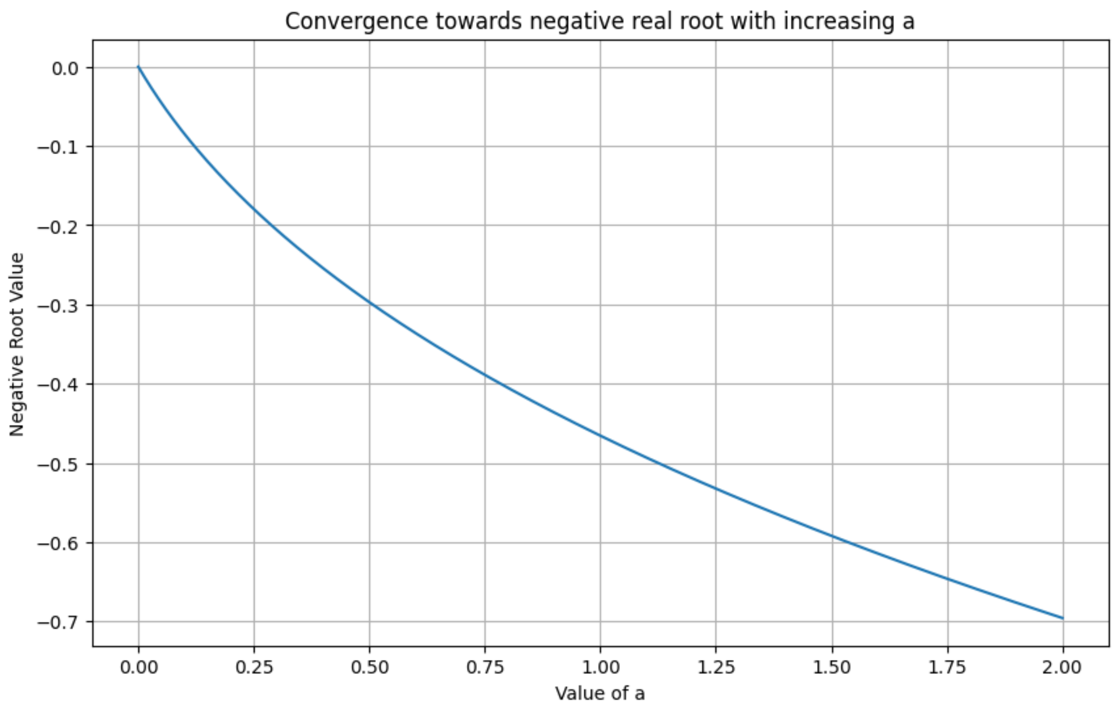

Error vs Iteration: Newton-Raphson vs Bisection
Newton-Raphson
Bisection
Error
Iteration
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

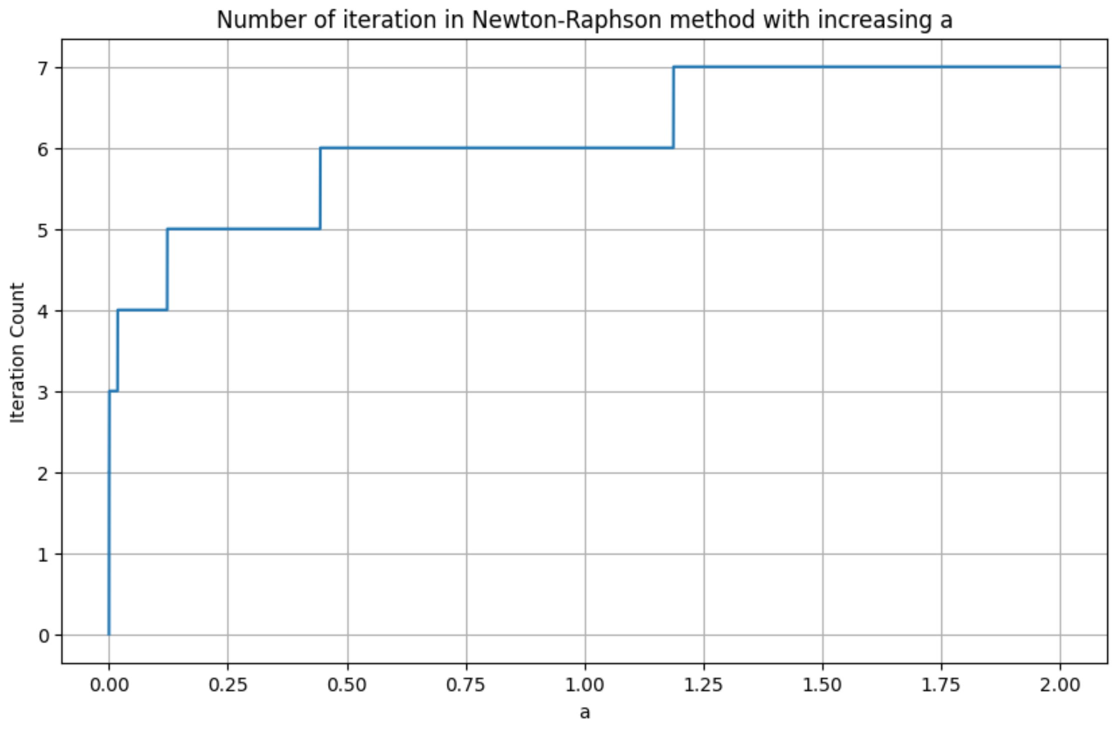


The reason for taking initial point of newton-raphson to be 4.4 is because it is very close to the actual root and other values are not converging towards the actual root and rather diverges because of the derivative we take in case of newton-raphson.

Question 4

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x, a):
    return a + x * ((x - 1) ** 2)

def deriv(x):
    return (x - 1) ** 2 + 2 * x * (x - 1)

def newton(a, x0, err):
    approximations = [x0]
    errors = []
    iteration = 0
    while True:
        function_val = f(x0, a)
        derivate_x = deriv(x0)
        x_next = x0 - (function_val / derivate_x)
        approximations.append(x_next)
        err_val = abs(x_next - x0)
        errors.append(err_val)
        if err_val <= err:
            break
        x0 = x_next
        iteration += 1
    return x_next, approximations, errors, iteration

a_newton = np.arange(0, 2, 0.0001)
roots_newton = []
iter_newton = []
for a in a_newton:
    root_newton, approx_newton, errors_newton, iteration_newton = newton(a, 0,␣
 ↪1e-12)
    roots_newton.append(root_newton)
    iter_newton.append(iteration_newton)

plt.figure(figsize=(10, 6))
plt.plot(a_newton, roots_newton)
plt.grid(True)
plt.xlabel('Value of a')
plt.ylabel('Negative Root Value')
plt.title('Convergence towards negative real root with increasing a')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(a_newton, iter_newton)
plt.grid(True)
plt.xlabel('a')
plt.ylabel('Iteration Count')
plt.title('Number of iteration in Newton-Raphson method with increasing a')
plt.show()
```


Convergence towards negative real root with increasing a
Negative Root Value
Value of a
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Number of iteration in Newton-Raphson method with increasing a
Iteration Count
a
0
1
2
3
4
5
6
7
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00