cs374-lab2

August 11, 2024

LAB 2

Bisection Method

Question 1>

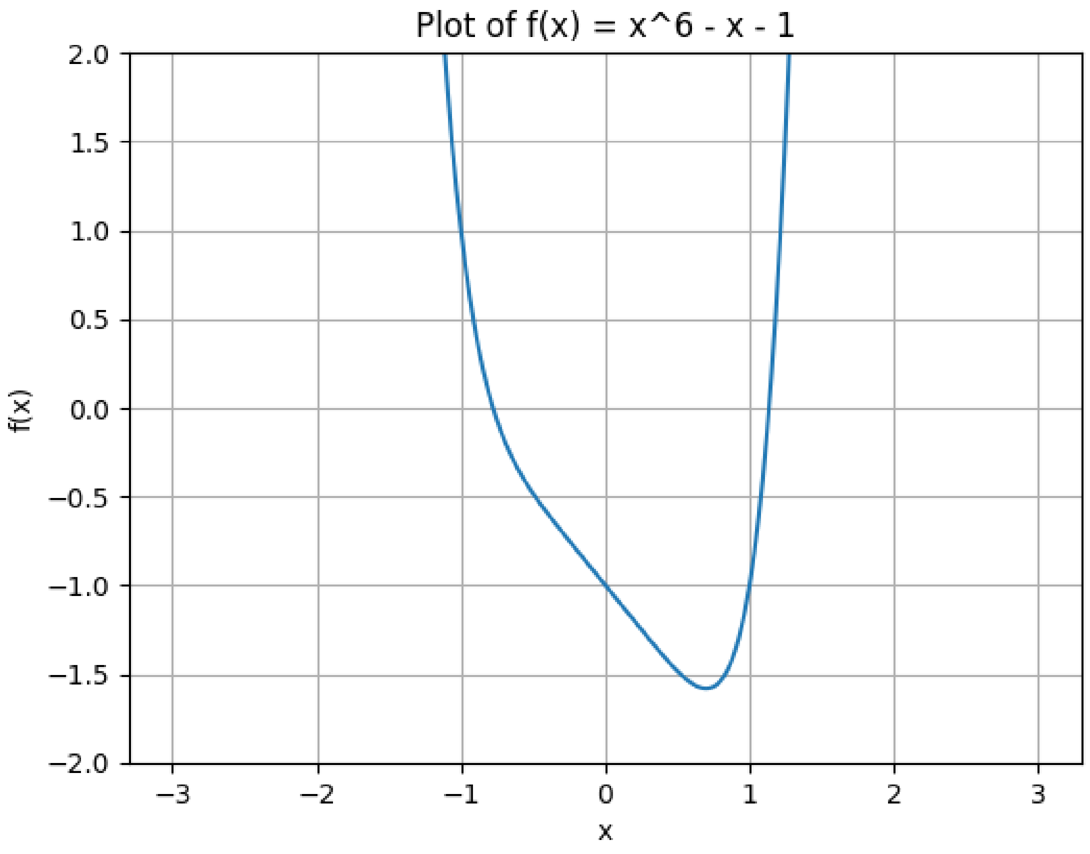
Write a code, applying the algorithm of the bisection method to determine both the real roots of f(x) = x^6 -x -1

Reason for taking two regions (-2,0) and (0,2) is because the given function has 2 roots as we found out that between (-2,0) and (0,2) function values change its sign and for the interval beyond it sign remains the same so it has 2 real roots and 4 imaginary roots.

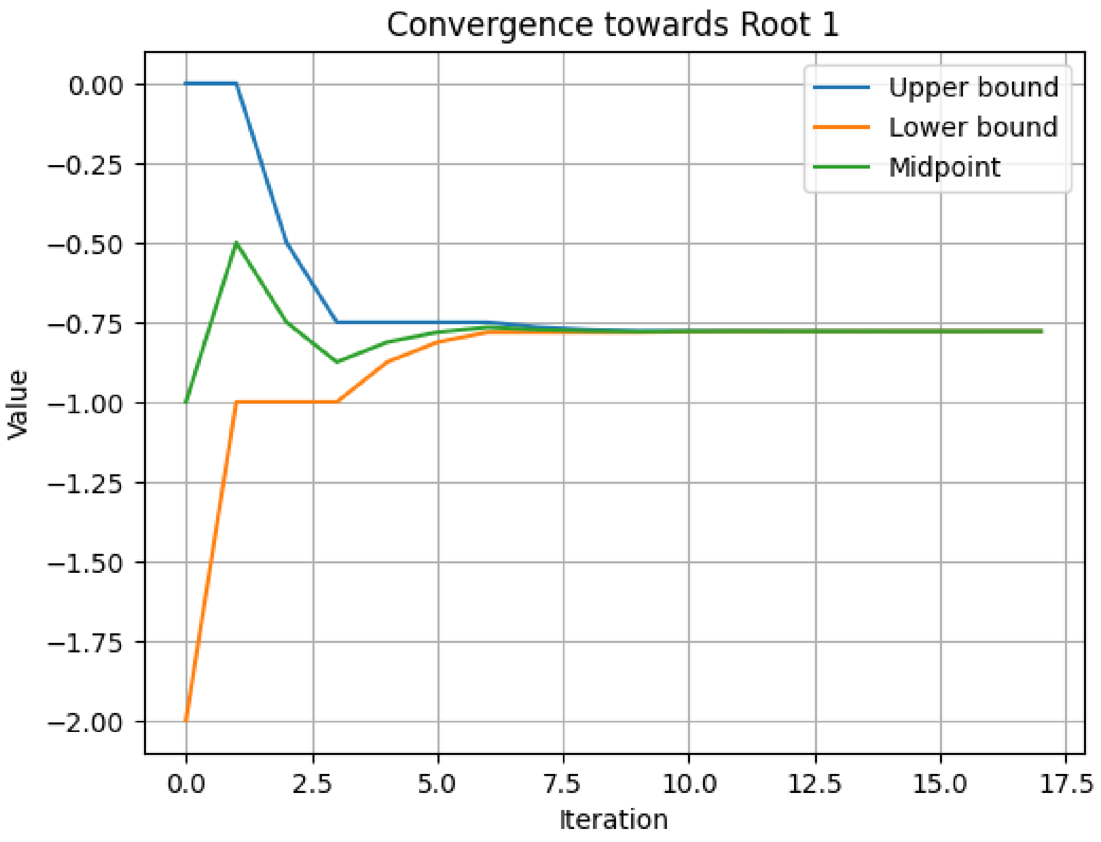
[2]:
```python
def f(x):
    return x**6 - x - 1

def bisection(a, b, max_error=0.01, total=100):
    for i in range(total):
        c = (a + b) / 2
        if f(c) == 0 or (b - a) / 2 < max_error:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2


a, b = -2, 0
root1 = bisection(a, b)
print("Root 1:", root1)

a, b = 0, 2
root2 = bisection(a, b)
print("Root 2:", root2)
```

```
Root 1: -0.7734375
Root 2: 1.1328125
```

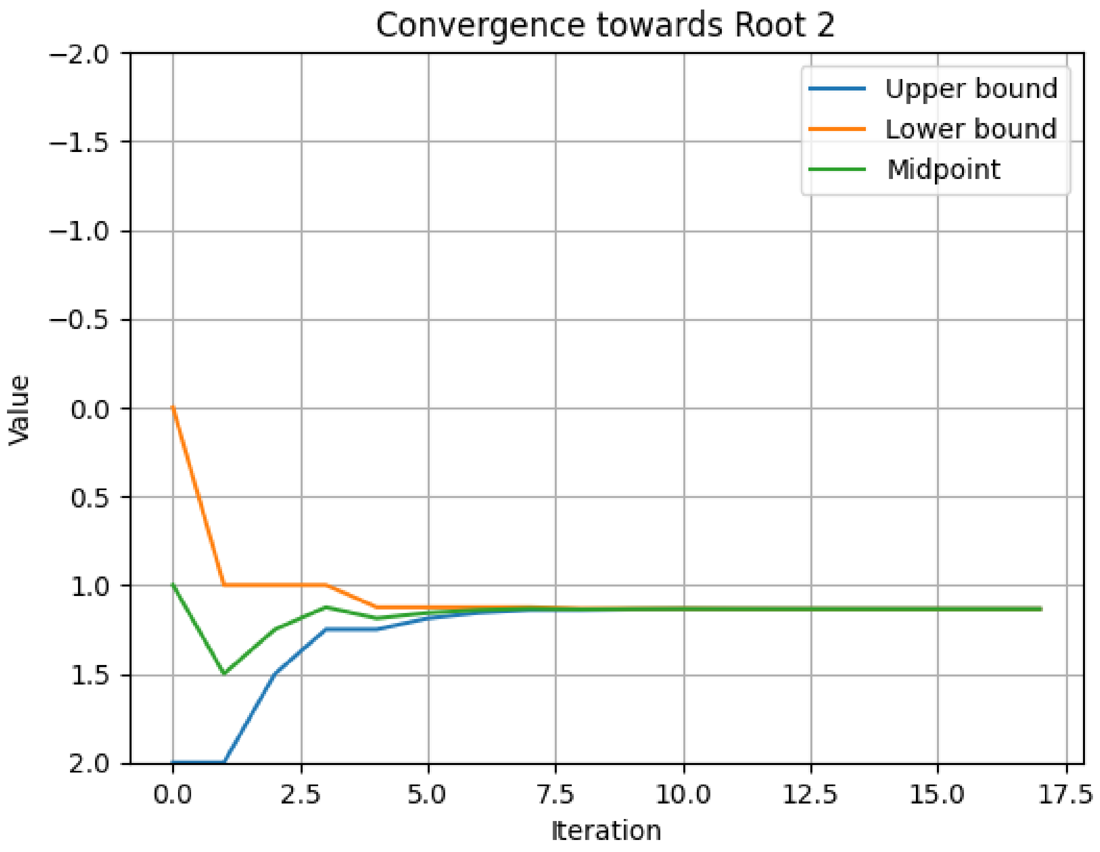
[26]:
```
import matplotlib.pyplot as plt
import numpy as np

def f(x):
    return x**6 - x - 1

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(-3, 3, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x^6 - x - 1')
plt.ylim(-2,2)
plt.grid(True)
plt.show()


a, b = -2, 0
r1, conv_table1 = bisect(a, b)
print("Root 1:", r1)


plt.plot([row[0] for row in conv_table1], [row[2] for row in conv_table1],␣
 ↪label='Upper bound')
plt.plot([row[0] for row in conv_table1], [row[1] for row in conv_table1],␣
 ↪label='Lower bound')
plt.plot([row[0] for row in conv_table1], [row[3] for row in conv_table1],␣
 ↪label='Midpoint')
plt.xlabel('Iteration')
plt.ylabel('Value')
plt.title('Convergence towards Root 1')
plt.legend()
plt.grid(True)
plt.show()

a, b = 0, 2
r2, conv_table2 = bisect(a, b)
print("Root 2:", r2)

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 2')
plt.legend()
plt.ylim(2,-2)
plt.grid(True)
plt.show()
```



```
Root 1: -0.7780838012695312
```



```
Root 2: 1.1347274780273438
```



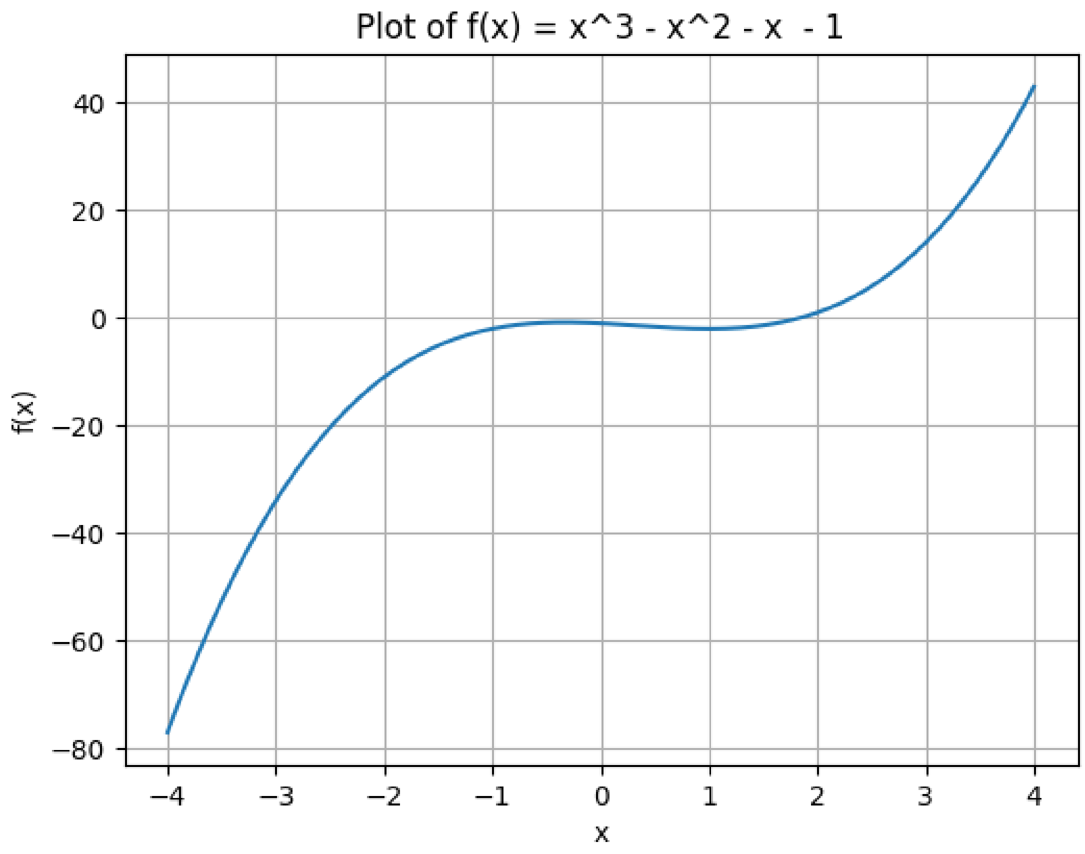
Question 2>

Numerically implement the bisection algorithm to solve all the problems given in the theory exercises (book) on bisection.

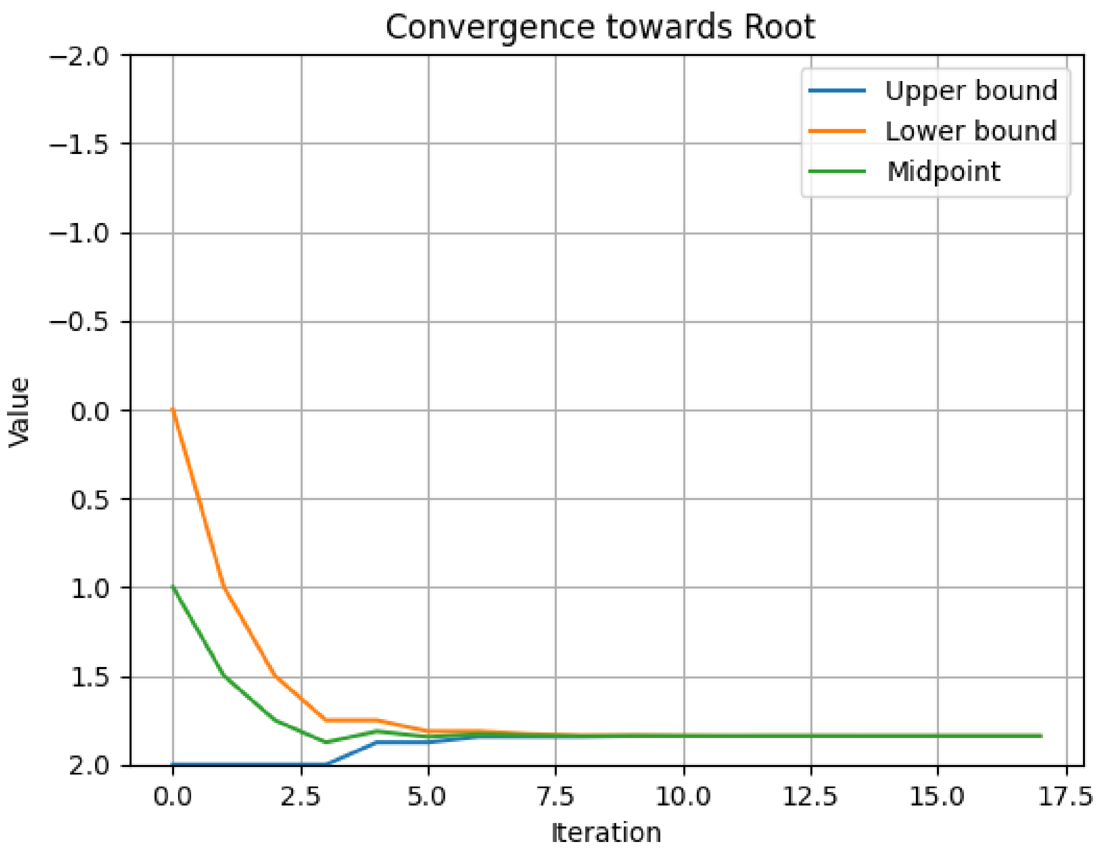
Part 1

Reason behind taking the interval (0,2) is because the sign of the function value changes in this interval only for the rest of interval it remains the same. Hence it has only 1 root.

[10]:
```python
import matplotlib.pyplot as plt
import numpy as np

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

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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x^3 - x^2 - x  - 1')
plt.grid(True)
plt.show()


a, b = 0, 2
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



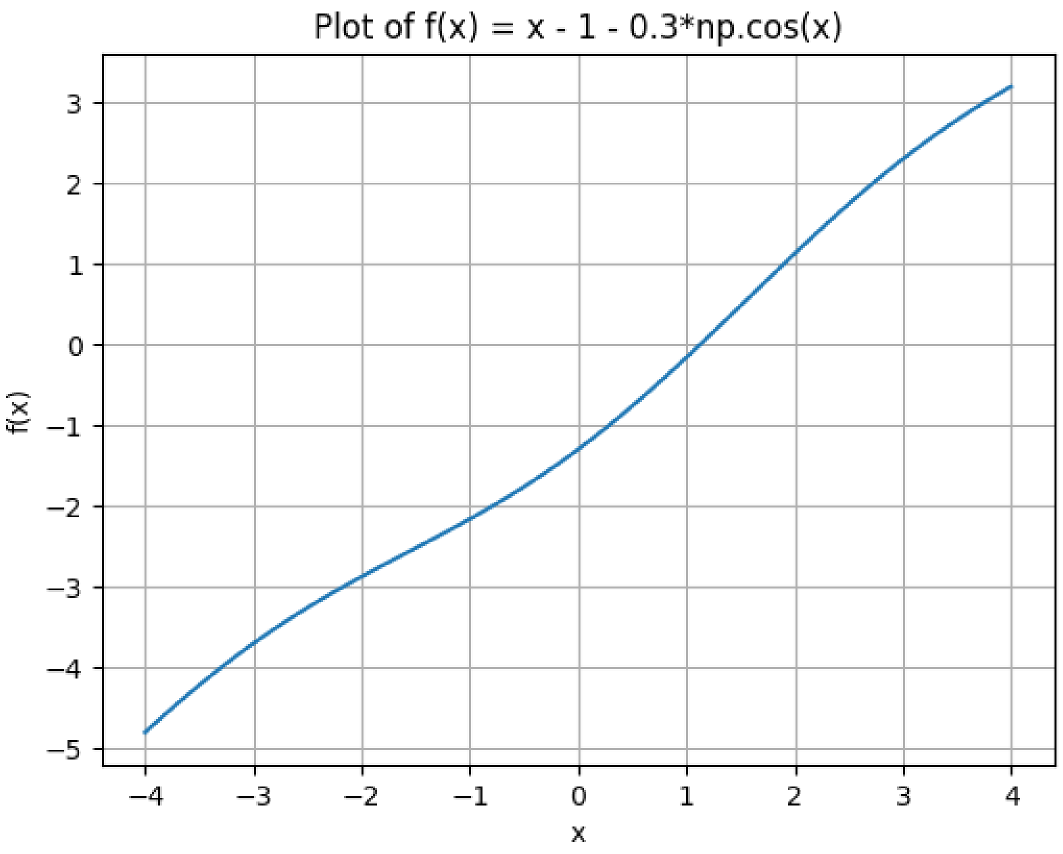
```
Root : 1.8392868041992188
```



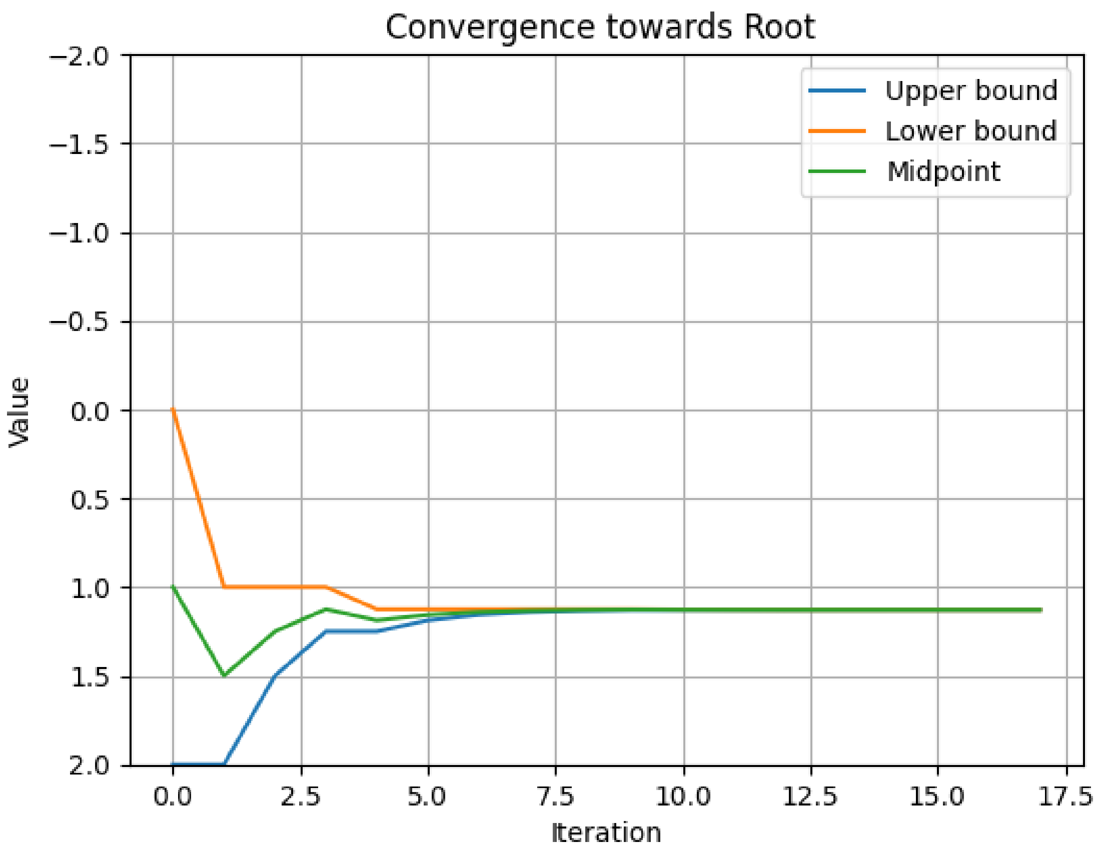
Part 2

Range of 1+0.3*cos(x) is [0.7,1.3]. Therefore for the solution x=1+0.3*cos(x) the root must lie between 0.7 and 1.3 hence the interval chosen.

[11]:
```python
import matplotlib.pyplot as plt
import numpy as np

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

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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x - 1 - 0.3*np.cos(x)')
plt.grid(True)
plt.show()


a, b = 0, 2
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



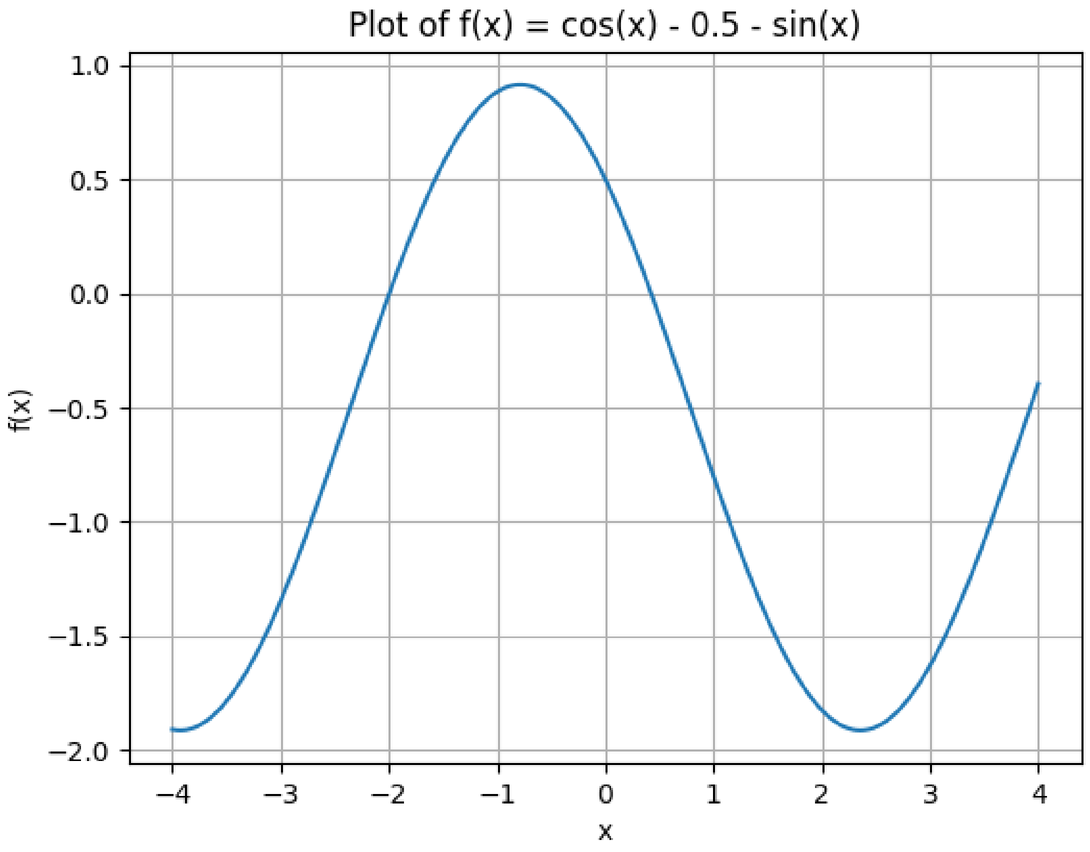
```
Root : 1.1284255981445312
```



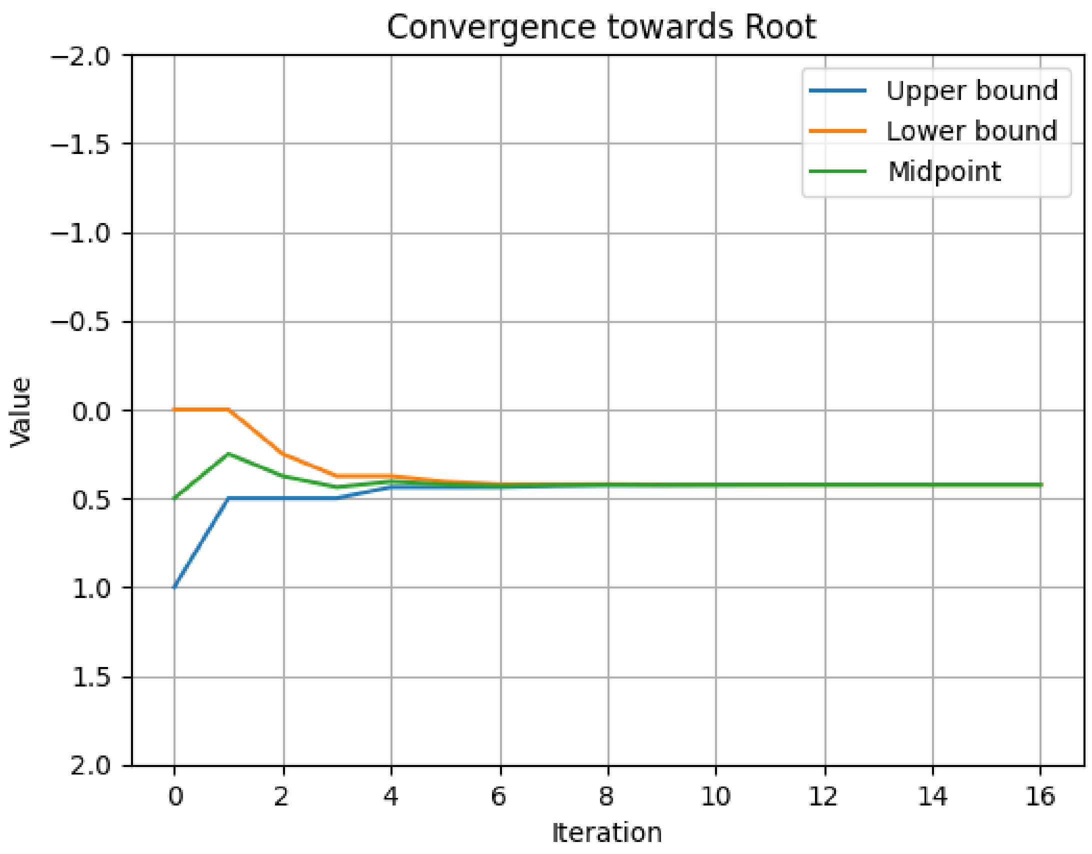
Part 3

There are infinite positive roots for this equation but the smallest positive root lies between (0,pi/2) as it is the first region where sign do change.

[13]:
```
import matplotlib.pyplot as plt
import numpy as np

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


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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = cos(x) - 0.5 - sin(x)')
plt.grid(True)
plt.show()


a, b = 0, 1
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



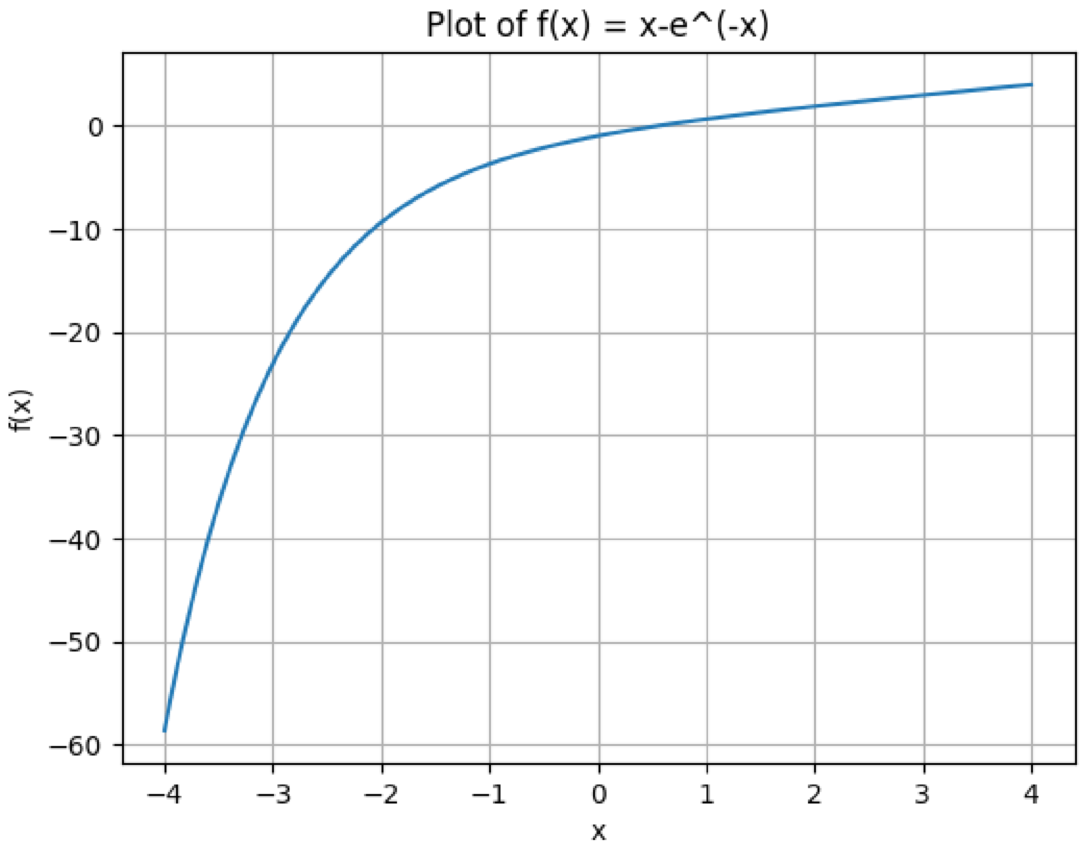
```
Root : 0.42403411865234375
```



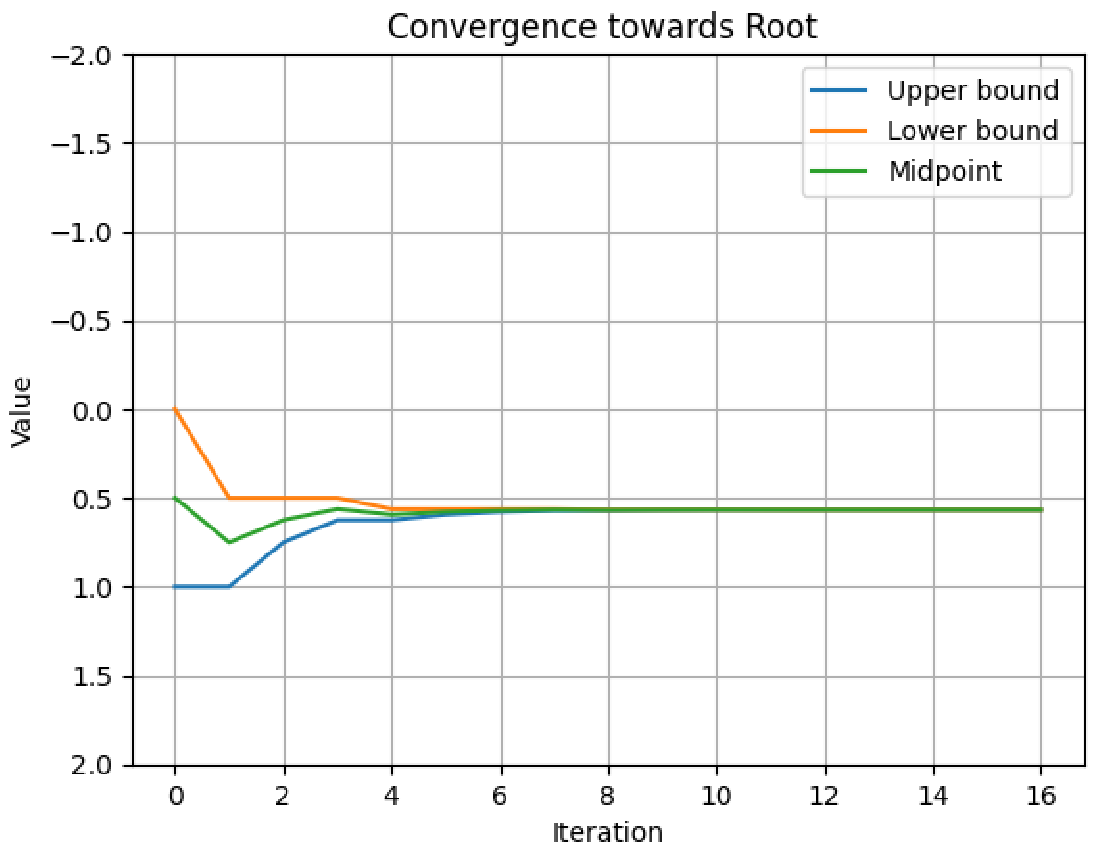
Part 4

As we know that e^(-x) is strictly decreasing function and x is strictly increasing, they would cut on only 1 point so they have only one root and between 0 and 1 it changes its sign.

[14]:
```
import matplotlib.pyplot as plt
import numpy as np

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


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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x-e^(-x)')
plt.grid(True)
plt.show()


a, b = 0, 1
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



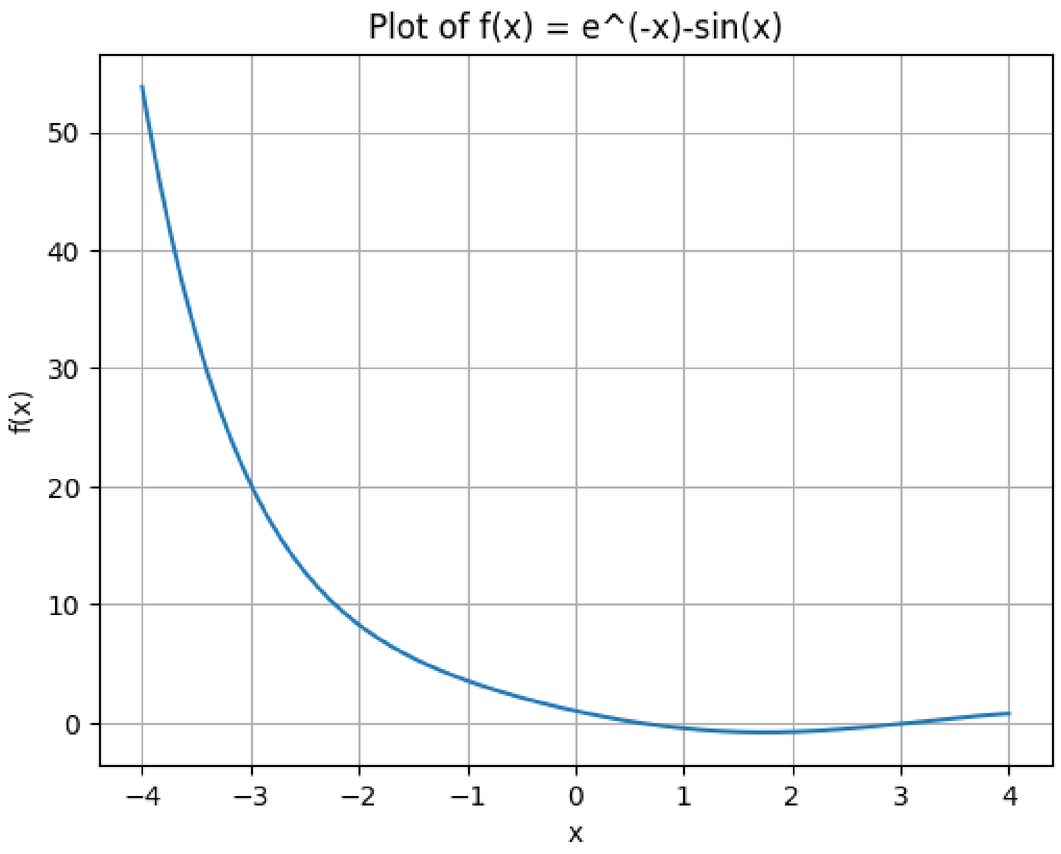
```
Root : 0.5671463012695312
```



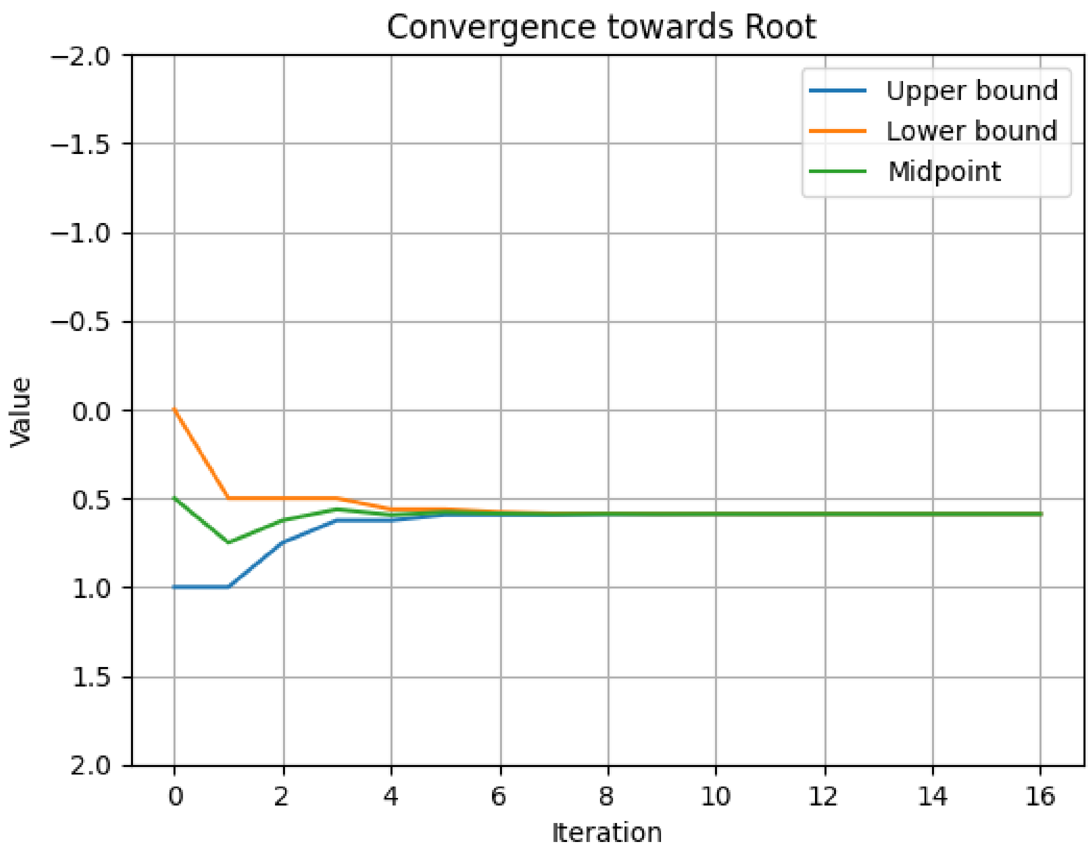
Part 5

This function also has infinite roots but we need to find out the smallest positive therefore the first positive interval where the function value changes it sign is between (0,1).

[15]:
```
import matplotlib.pyplot as plt
import numpy as np

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


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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = e^(-x)-sin(x)')
plt.grid(True)
plt.show()


a, b = 0, 1
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



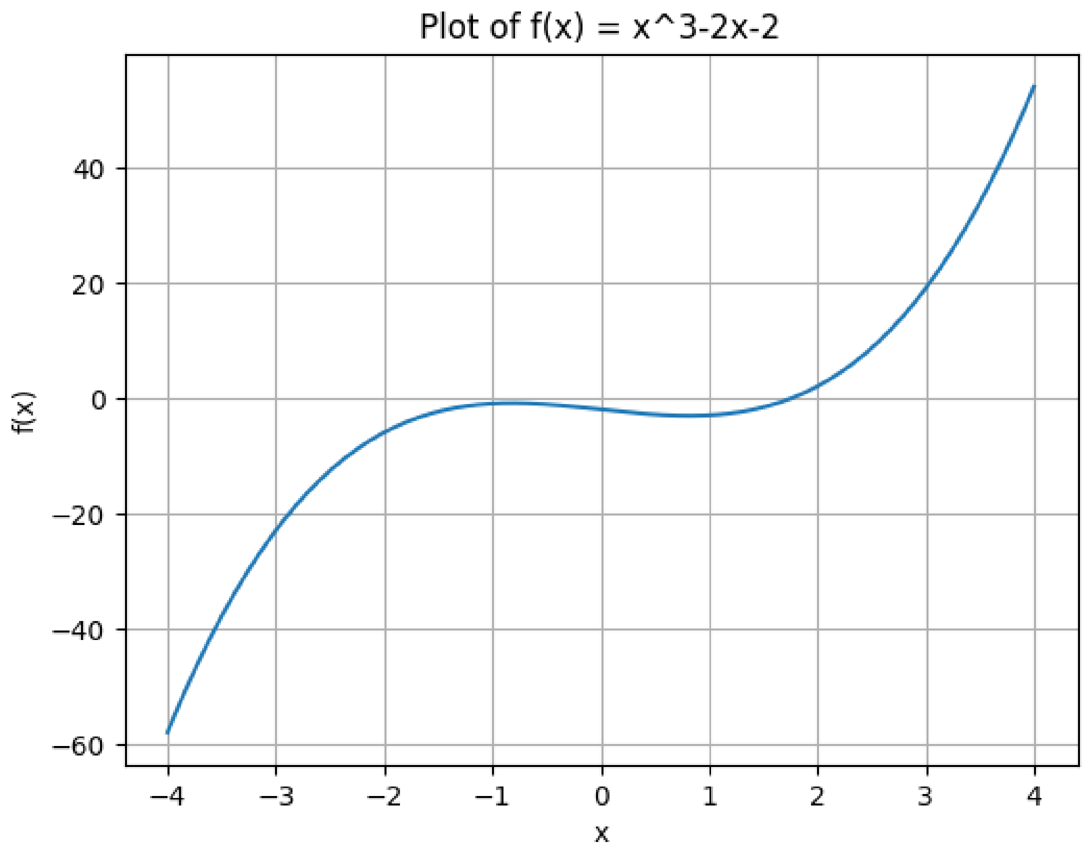
```
Root : 0.5885391235351562
```



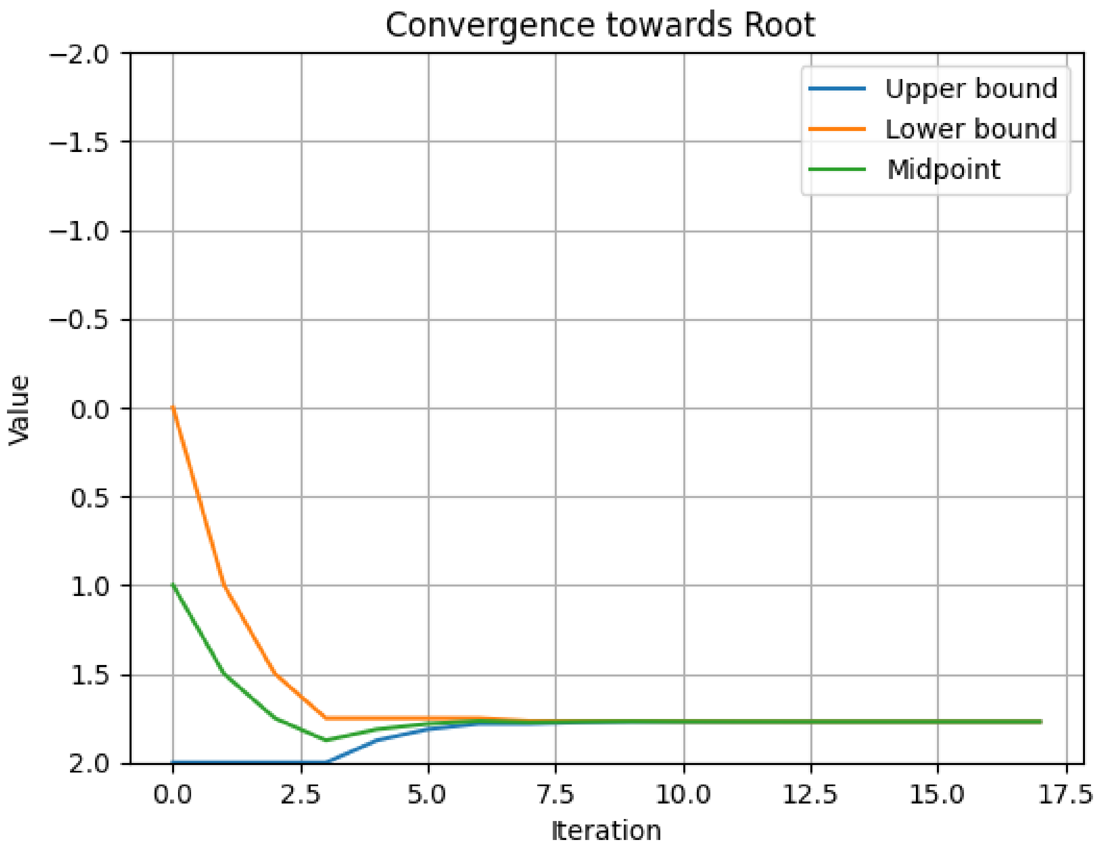
Part 6

This function too has the same reason that for the interval [0,2] the function changes it sign between it and for the rest of the intervals the sign remains as it is.

[16]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x^3-2x-2')
plt.grid(True)
plt.show()


a, b = 0, 2
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()
```



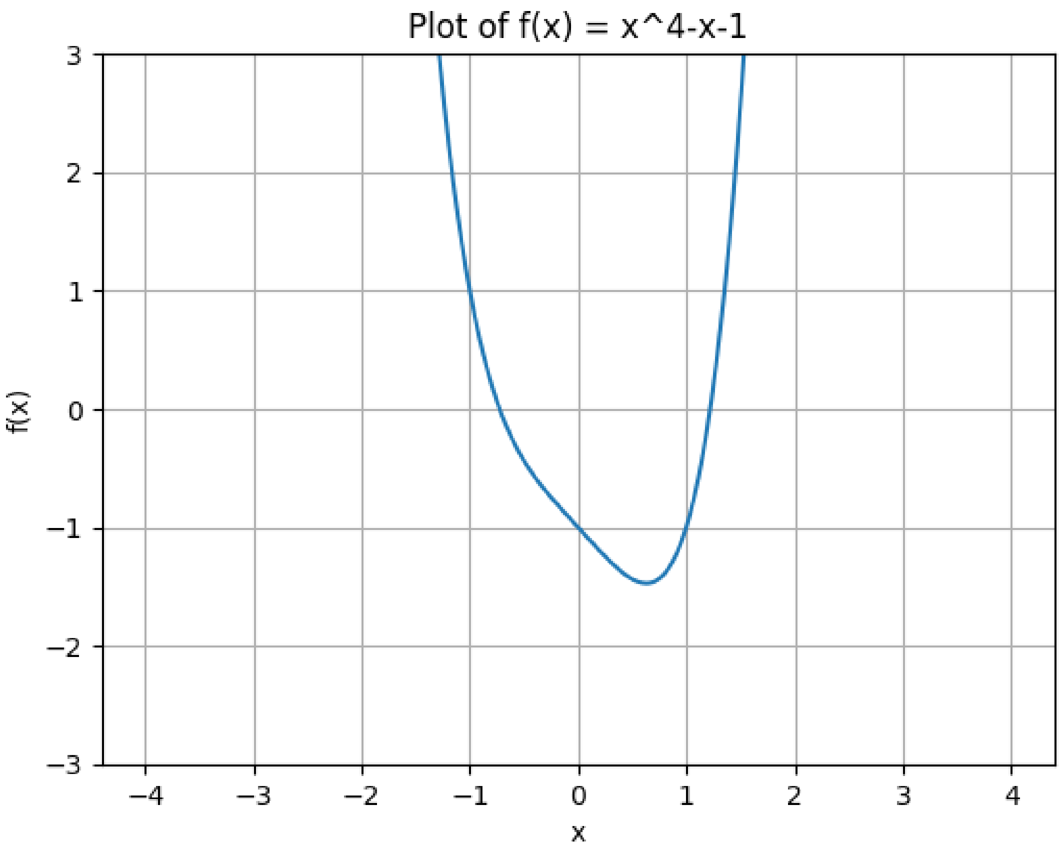
```
Root : 1.7692947387695312
```



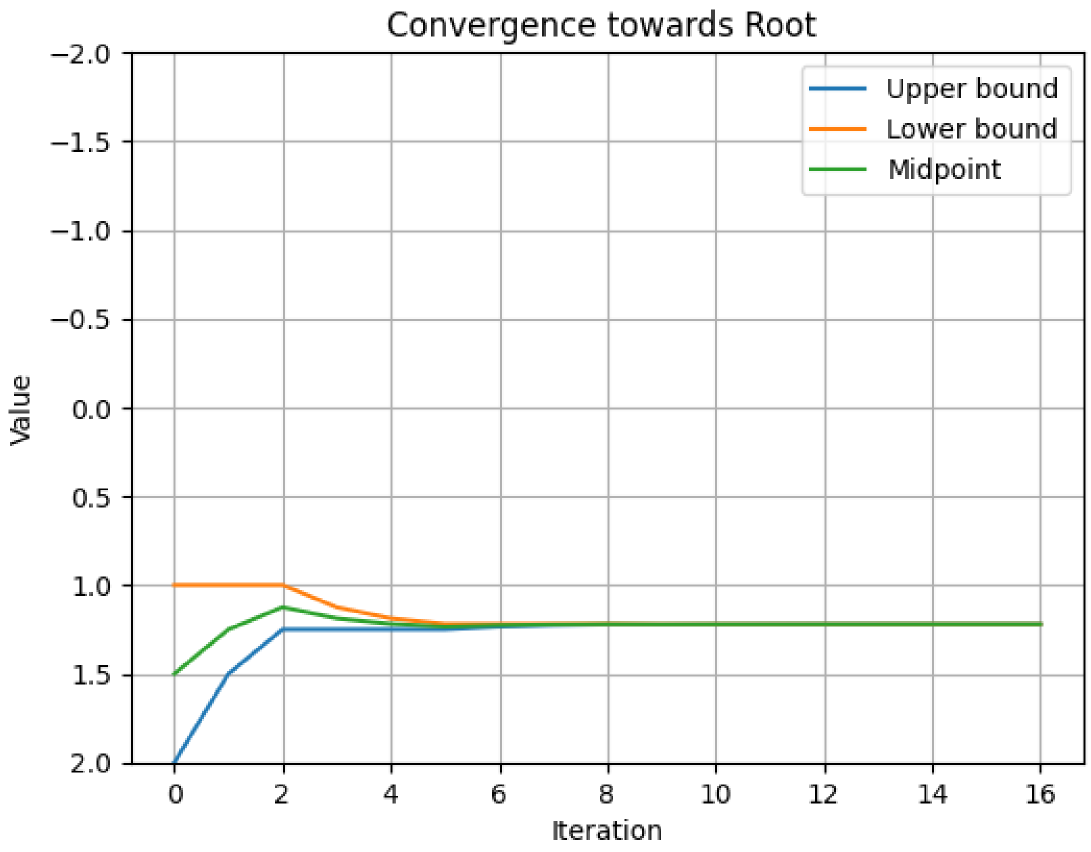
Part 7

From the plot we can see that it has 2 roots and we can also verify that by checking the sign of the function values which change between (1,2) and (-2,0)

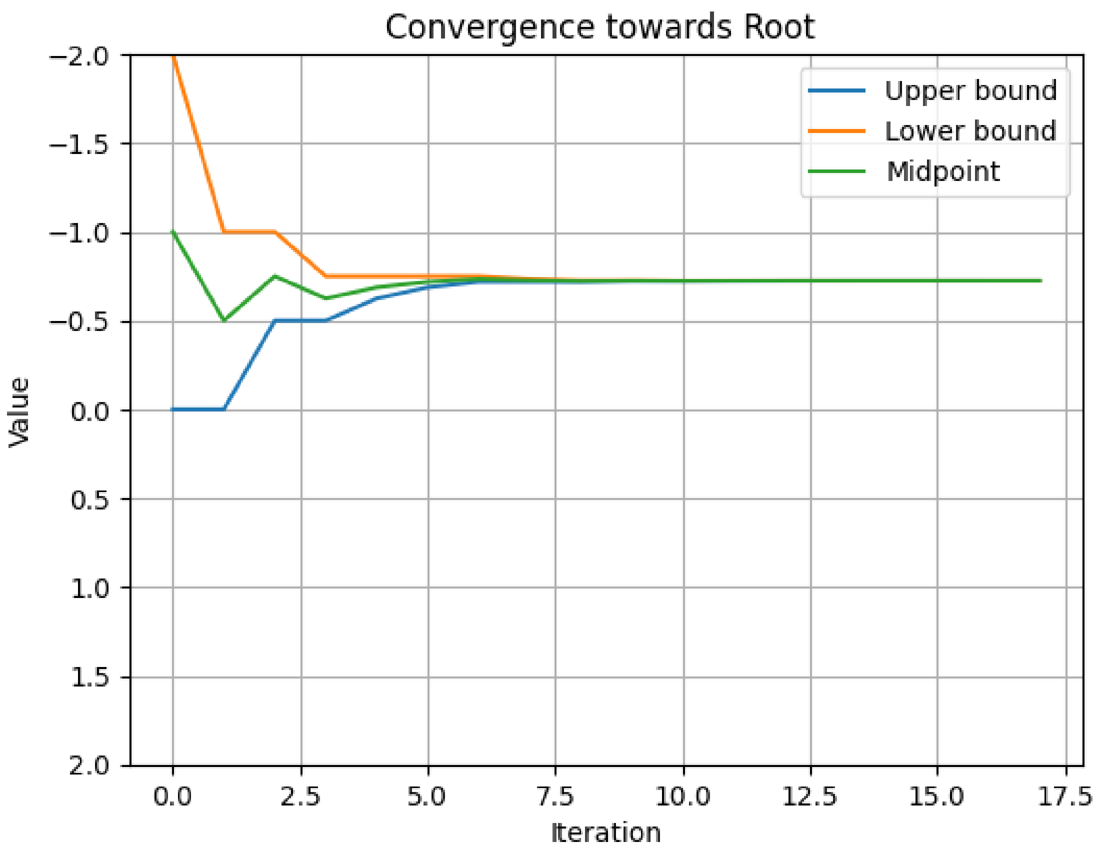
[21]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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(-4, 4, 400)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = x^4-x-1')
plt.ylim(-3,3)
plt.grid(True)
plt.show()


a, b = 1, 2
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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(2,-2)
plt.grid(True)
plt.show()

a, b = -2, 0
r1, conv_table2 = bisect(a, b)
print("Root :", r1)

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(2,-2)
plt.grid(True)
plt.show()
```



```
Root : 1.2207412719726562
```



```
Root : -0.7244949340820312
```

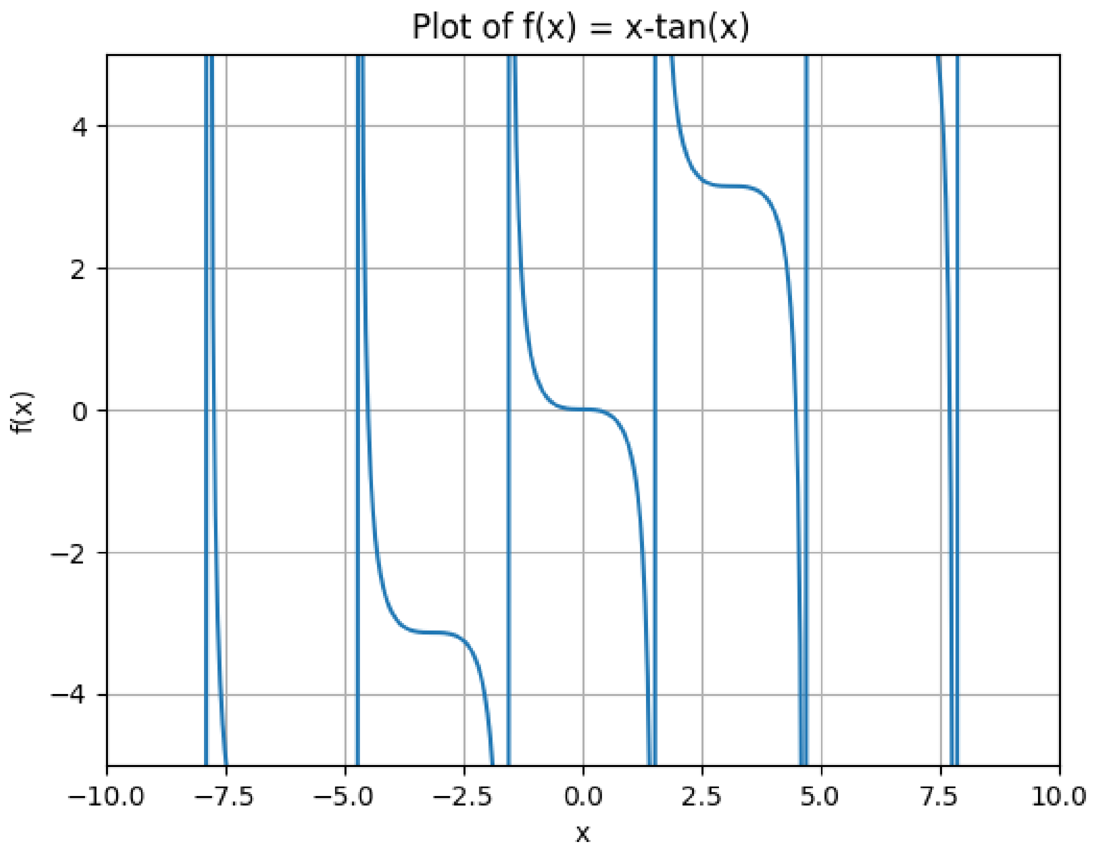


Part 8

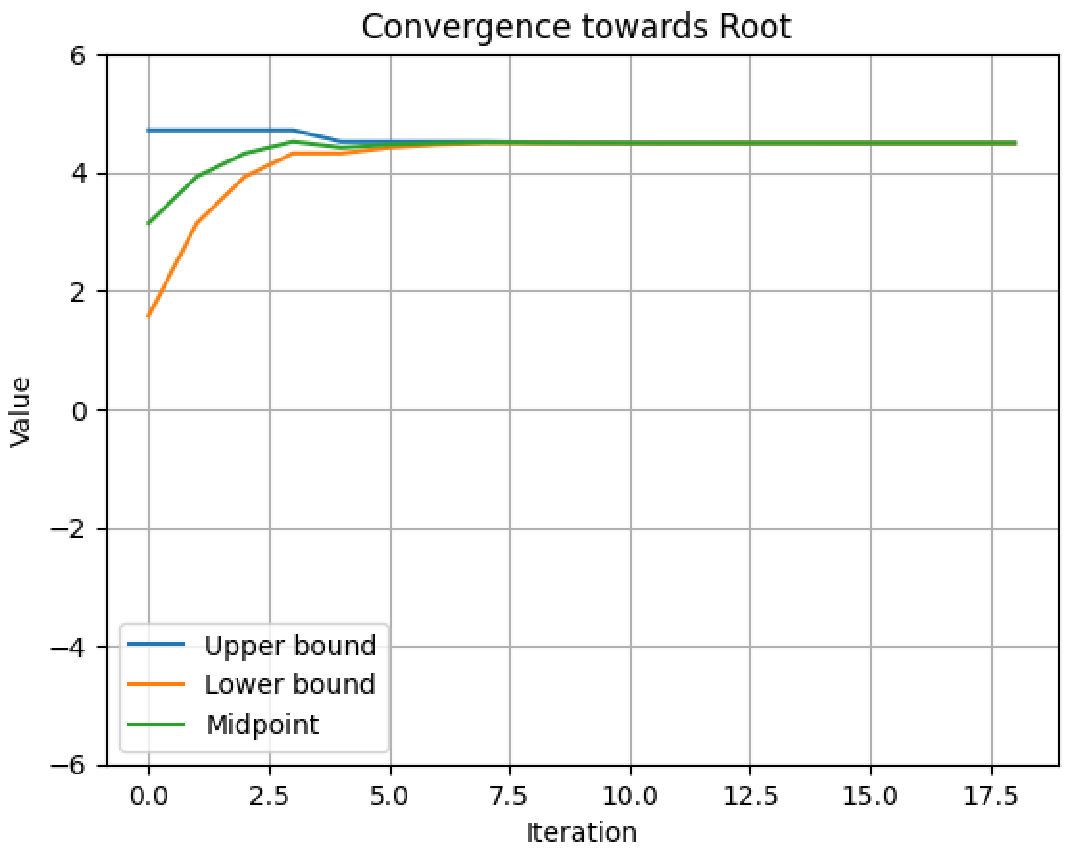
Reason for taking such an interval is because for this interval the function value changes its sign.

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

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


      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()


a, b = (np.pi)/2 +0.01, 1.5*(np.pi)
r2, conv_table2 = bisect(a, b)
print("Root :", r2)

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()
```



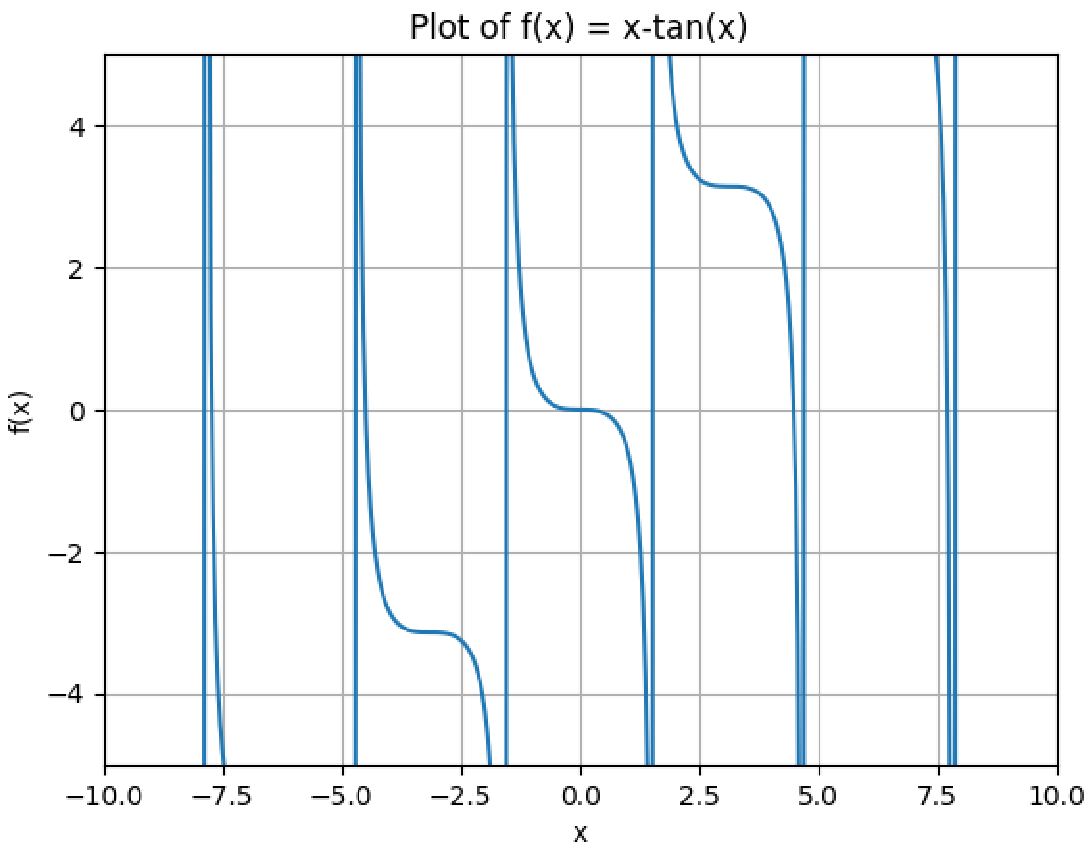
```
Root : 4.493411398839326
```



Part 9

For finding out the root near to 100 it is difficult to define integral interval as the function changes its sign very rapidly so the best and nearest interval to 100 is the one defines in the code

[34]:
```
import matplotlib.pyplot as plt
import numpy as np

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


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()


a, b =98.0,98.9520
r2, conv_table2 = bisect(a, b)
print("Root :", r2)
```



```
Root : 98.95006072998046
```

Machine Epsilon

Question 1>

Write a function that computes and prints the machine epsilon as defined in the lectures.

```
[1]: def compute_machine_epsilon():
         eps = 1.0
         while 1.0 + eps != 1.0:
             eps /= 2.0
         eps *= 2.0
         print("Machine epsilon:", eps)

     compute_machine_epsilon()
```

```
Machine epsilon: 2.220446049250313e-16
```

Question 2>

Generalize your function for any other integer say ‘n’. Will the value change? Justify with examples.

From the below examples we can see that as the n increases the epsilon value also increases. The floating point of any n is computed as n(1+delta) by the computer. For n=1 delta is the epsilon value but for n the equation simplifies as n+n*delta ; therefore n*delta is the resultant epsilon which is upscaled due to factor of n hence it increases as n increases.

```
[27]: def eps(n):
        first = 1.0
        while n + first != n:
          first = first / 2
        first *= 2
        return first

      ns = [1, 2, 4, 8]
      for n in ns:
        print(f"epsilon for {n}: {eps(n)}")
```

```
epsilon for 1: 2.220446049250313e-16
epsilon for 2: 4.440892098500626e-16
epsilon for 4: 8.881784197001252e-16
epsilon for 8: 1.7763568394002505e-15
```