Ex

Exercise 3.1

- 14, 15, 16, 17, 21

Exercise 3.2

- 1, 7, 25, 35, 39

Exercise 3.3

- 10, 12

Com Ex

Computer Exercise 3.1

- 5, 18

Computer Exercise 3.2

- 4, 16, 41

Computer Exercise 3.3

- 7, 9, 15

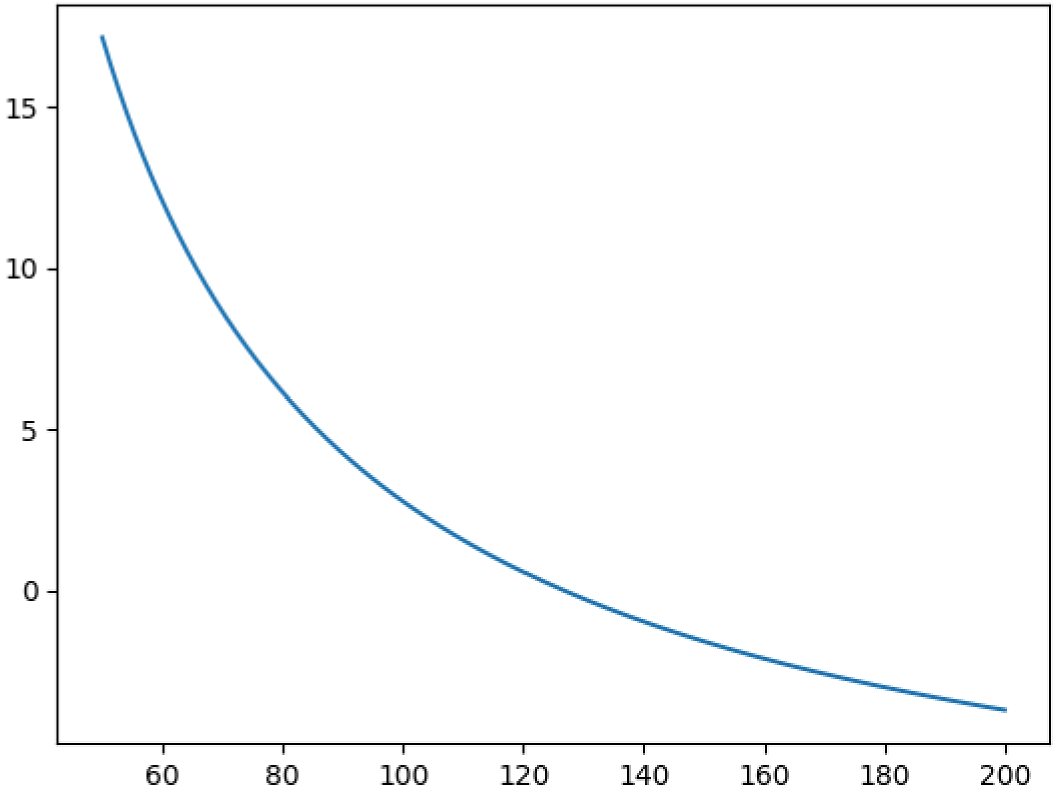
In [25]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

In [26]:
```
# 3.1 - 5

def bisection_method(f, a, b, n, tol = 1e-10):
    for i in range(n):
        c = (a + b) / 2
        if f(c) == 0 or (b - a) / 2 < tol:
            print(f"Function converged after {i} iterations.")
            print(f"Root: {c}")
            print(f"Error: {f(c)}")
            return c
        if np.sign(f(c)) == np.sign(f(a)):
            a = c
        else:
            b = c
    return c

f = lambda x: x * np.cosh(50 / x) - x - 10
```

```
plt.plot(np.linspace(50, 200, 1000), f(np.linspace(50, 200, 1000)))

bisection_method(f, 50, 200, 10000, 1e-20)
```

```
Function converged after 47 iterations.
Root: 126.63243603998868
Error: 0.0
```

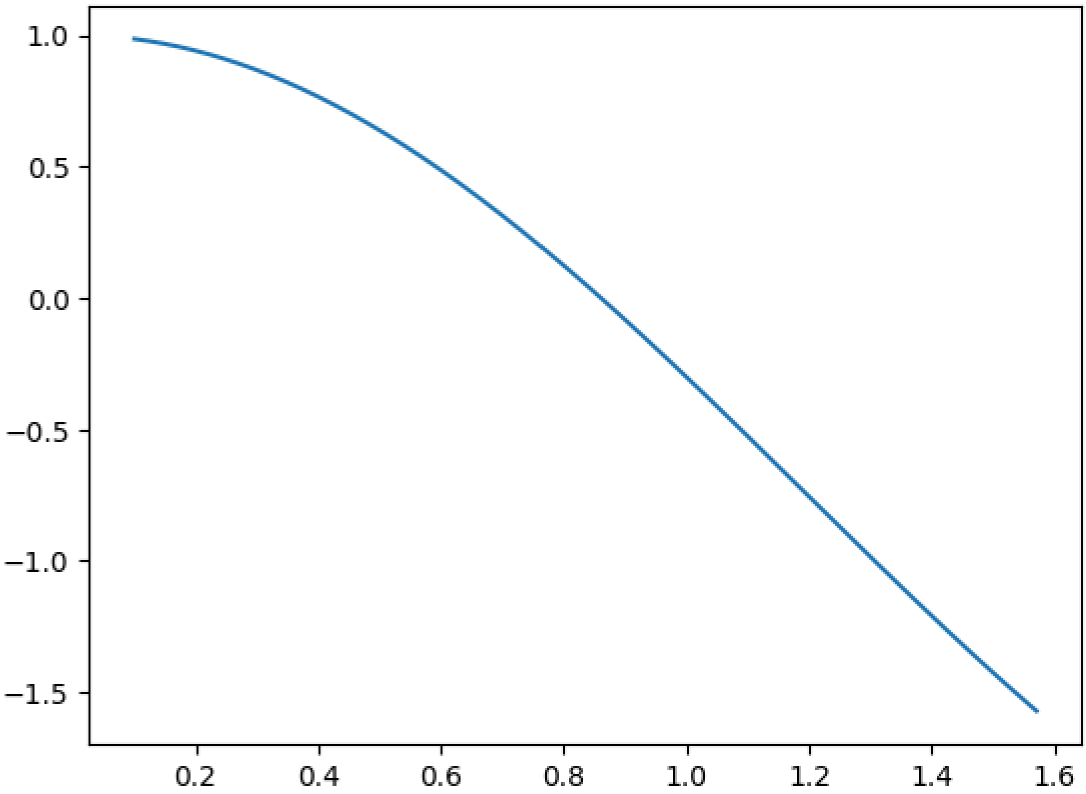
Out[26]: 126.63243603998868



In [27]:
```
# 3.1 - 18
l = np.pi / 2

def bisection_method(f, a, b, n, tol = 1e-10):
    for i in range(n):
        c = (a + b) / 2
        if f(c) == 0 or (b - a) / 2 < tol:
            print(f"Function converged after {i} iterations.")
            print(f"Root: {c}")
            print(f"Error: {f(c)}")
            return c
        if np.sign(f(c)) == np.sign(f(a)):
            a = c
        else:
            b = c


    return c

area = lambda x: x * np.cos(x)
area_prime = lambda x: np.cos(x) - x * np.sin(x)

plt.plot(np.linspace(0.1, l, 1000), area_prime(np.linspace(0.1, l, 1000)))
```

```python
x = bisection_method(area_prime, 0.1, l, 10000, 1e-20)
print("Maximum area: ", area(x))
```

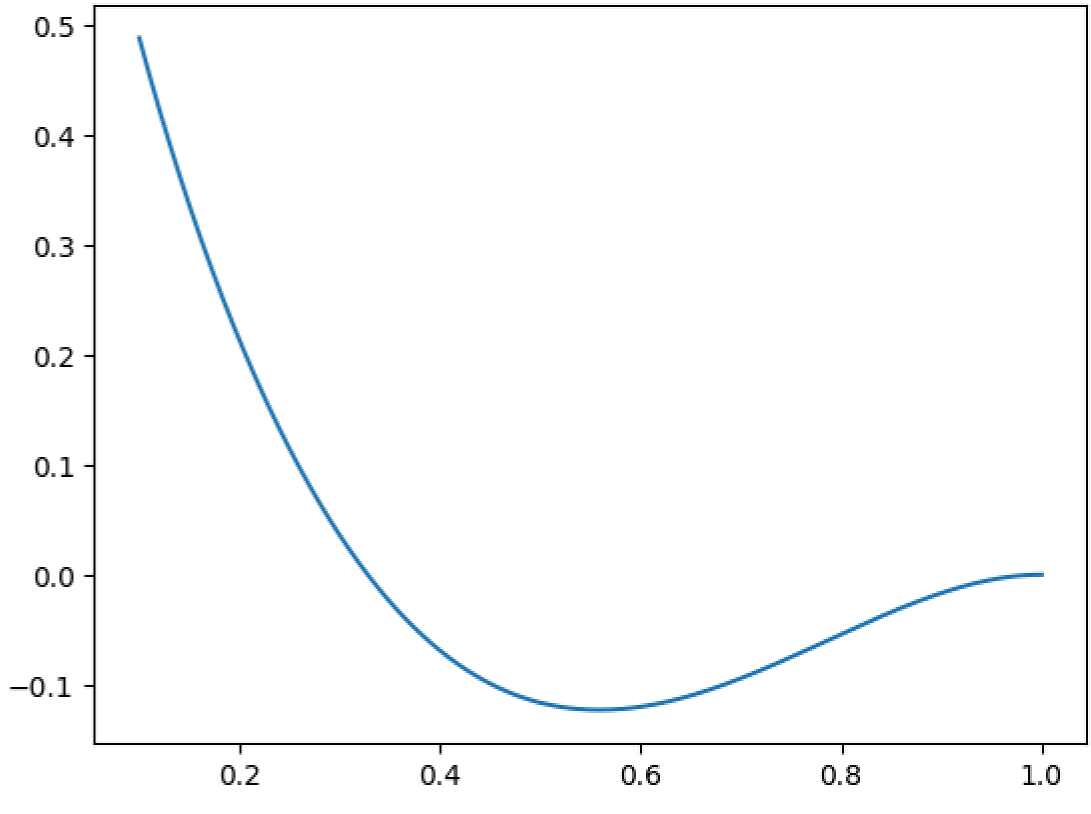
```
Maximum area:  0.5610963381910451
```



In [28]:
```python
# 3.2 - 4
def newton_method(f, f_prime, x0, n, tol = 1e-10, verbose = False):
    for i in range(n):
        x = x0 - f(x0) / f_prime(x0)
        if verbose:
            print(f"x_{i} = {x}")
        if abs(x - x0) < tol:
            print(f"Function converged after {i} iterations.")
            print(f"Root: {x}")
            print(f"Error: {f(x)}")
            return x
        x0 = x

    print("Function did not converge.")
    print(f"Root: {x}")
    print(f"Error: {f(x)}")
    return x

f = lambda x: 2*x*(1 - x**2 + x)*np.log(x) - x**2 + 1
f_prime = lambda x: 2*(1 - x**2 + x)*np.log(x) + 2*x*(1 - x**2 + x) / x - 2*x

plt.plot(np.linspace(0.1, 1, 1000), f(np.linspace(0.1, 1, 1000)))
newton_method(f, f_prime, 0.1, 10000, 1e-10, True)
newton_method(f, f_prime, 0.8, 10000, 1e-10, True)
```

```
x_0 = 0.26055755674963477
x_1 = 0.3321496259819407
x_2 = 0.3267790293447707
x_3 = 0.3282820881189536
x_4 = 0.32788295819858165
x_5 = 0.3279905551455337
x_6 = 0.3279616643433674
x_7 = 0.32796943012972996
x_8 = 0.32796734330407806
x_9 = 0.32796790412024973
x_10 = 0.327967753408937
x_11 = 0.32796779391067016
x_12 = 0.327967783026365
x_13 = 0.32796778595137904
x_14 = 0.3279677851653203
x_15 = 0.32796778537656307
x_16 = 0.3279677853197943
Function converged after 16 iterations.
Root: 0.3279677853197943
Error: 1.4283019211802639e-11
x_0 = 1.0676844681201105
x_1 = 1.0032428331611858
x_2 = 1.000008690010944
x_3 = 1.0000000000645168
x_4 = 1.0000000000645168
Function converged after 4 iterations.
Root: 1.0000000000645168
Error: 0.0
```

Out[28]: np.float64(1.0000000000645168)



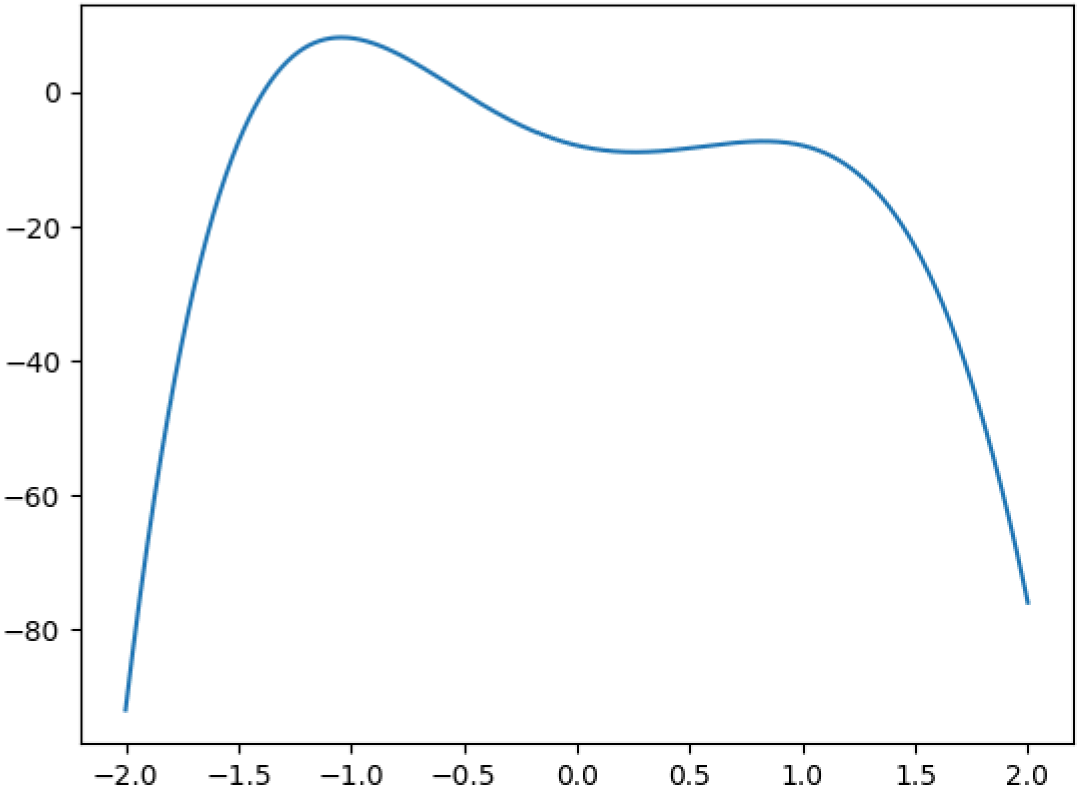
In [29]:

```
# 3.2 - 16
f = lambda x: x**5 - 9*x**4 - x**3 + 17*x**2 - 8*x - 8
f_prime = lambda x: 5*x**4 - 36*x**3 - 3*x**2 + 34*x - 8

plt.plot(np.linspace(-2, 2, 1000), f(np.linspace(-2, 2, 1000)))
```

```
newton_method(f, f_prime, 0, 10, 1e-20)
```

```
Function did not converge.
Root: -1.0
Error: 8.0
```

Out[29]: -1.0



This equation is speical case that newton's raw doesn't work.

In [30]:
```
a1, b1, c1 = 1.0, 1.0, 1.0
a2, b2, c2 = 3.0, 0.0, 0.0
a3, b3, c3 = 0.0, 4.0, 0.0
a4, b4, c4 = 0.0, 0.0, 5.0
C = 3
D = 0.7

def F(X):
    x, y, z, t = X
    f1 = (x - a1)**2 + (y - b1)**2 + (z - c1)**2 - C**2 * (t - D)**2
    f2 = (x - a2)**2 + (y - b2)**2 + (z - c2)**2 - C**2 * (t - D)**2
    f3 = (x - a3)**2 + (y - b3)**2 + (z - c3)**2 - C**2 * (t - D)**2
    f4 = (x - a4)**2 + (y - b4)**2 + (z - c4)**2 - C**2 * (t - D)**2
    return np.array([f1, f2, f3, f4])

def J(X):
    x, y, z, t = X
    J_matrix = np.array([
        [2*(x - a1), 2*(y - b1), 2*(z - c1), -2*C**2*(t - D)],
        [2*(x - a2), 2*(y - b2), 2*(z - c2), -2*C**2*(t - D)],
        [2*(x - a3), 2*(y - b3), 2*(z - c3), -2*C**2*(t - D)],
        [2*(x - a4), 2*(y - b4), 2*(z - c4), -2*C**2*(t - D)]
    ])
    return J_matrix
```

```python
X = np.array([0.5, 0.5, 0.5, 1.0])

def nonlinear_newton_method(F, J, X, max_iter=100, tol=1e-10):
    for i in range(max_iter):
        F_val = F(X)
        normF = np.linalg.norm(F_val)
        if normF < tol:
            print(f"Converged after {i} iterations.")
            print(f"Solution: x={X[0]}, y={X[1]}, z={X[2]}, t={X[3]}")
            print(f"Residual: {F_val}")
            return X

        J_val = J(X)
        delta = np.linalg.solve(J_val, F_val)
        X = X - delta

    print(f"Did not converge after {max_iter} iterations.")
    print(f"Solution: x={X[0]}, y={X[1]}, z={X[2]}, t={X[3]}")
    print(f"Residual: {F_val}")
    return X

nonlinear_newton_method(F, J, X, 100000)
```

```
Converged after 5 iterations.
Solution: x=8.423076923076923, y=7.1923076923076925, z=6.653846153846153, t=-3.03
29273870589244
Residual: [0. 0. 0. 0.]
```

Out[30]:

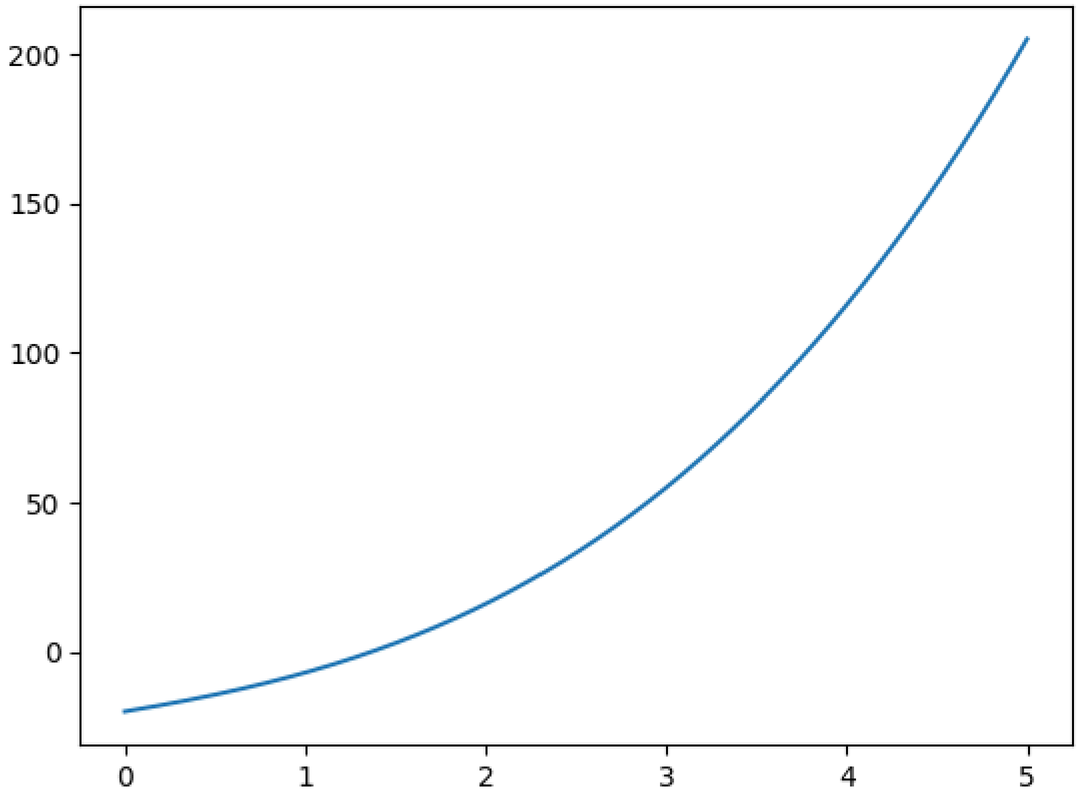
```
array([ 8.42307692,  7.19230769,  6.65384615, -3.03292739])
```

In [31]:

```python
# 3.3 - 7

def secant_method(f, x0, x1, n, tol = 1e-10, print_progress = False):
    for i in range(n):
        x = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        if print_progress:
            print(f"x_{i} = {x}")
        if abs(x - x1) < tol:
            print(f"Function converged after {i} iterations.")
            print(f"Root: {x}")
            print(f"Error: {f(x)}")
            return x
        x0 = x1
        x1 = x

    print("Function did not converge.")
    print(f"Root: {x}")
    print(f"Error: {f(x)}")
    return x

f = lambda x: x**3 + 2*x**2 + 10*x - 20

plt.plot(np.linspace(0, 5, 1000), f(np.linspace(0, 5, 1000)))
secant_method(f, 2, 1, 20, 1e-20, True)
newton_method(f, lambda x: 3*x**2 + 4*x + 10, 2, 20, 1e-10, True)
```

```
x_0 = 1.3043478260869565
x_1 = 1.3760536203919975
x_2 = 1.3686719535313416
x_3 = 1.368807822525221
x_4 = 1.3688081078326166
x_5 = 1.3688081078213725
x_6 = 1.3688081078213725
Function converged after 6 iterations.
Root: 1.3688081078213725
Error: 0.0
x_0 = 1.4666666666666668
x_1 = 1.3715120138059207
x_2 = 1.3688102226338952
x_3 = 1.3688081078226673
x_4 = 1.3688081078213725
Function converged after 4 iterations.
Root: 1.3688081078213725
Error: 0.0
```

Out[31]: 1.3688081078213725



In [32]:
```
# 3.3 - 9
# (x - 1) * (x - 2) * (x - 3) * (x - 4) * (x - 5) ... (x - n)
def build_f(n):
    f = lambda x: np.prod([x - i for i in range(1, n + 1)])
    return lambda x: f(x) - 1e-8 * x**19

f = build_f(20)

x = secant_method(f, 22, 21, 100, 1e-20, True)
```

```
x_0 = 21.33483224926118
x_1 = 20.84696215948804
x_2 = 20.74581199481376
x_3 = 20.56130245863504
x_4 = 20.44093642526385
x_5 = 20.340411150461463
x_6 = 20.27847784830728
x_7 = 20.248982845698496
x_8 = 20.241126286615774
x_9 = 20.24029509611107
x_10 = 20.240275172894005
x_11 = 20.240275126532065
x_12 = 20.24027512652954
x_13 = 20.24027512652954
Function converged after 13 iterations.
Root: 20.24027512652954
Error: 752.0
```

In [33]:

```
# 3.3 - 15

def fixed_point_iteration(f, x0, n, tol = 1e-10, print_progress = False):
    for i in range(n):
        x = f(x0)
        if print_progress:
            print(f"x_{i} = {x}")
        if abs(x - x0) < tol:
            print(f"Function converged after {i} iterations.")
            print(f"Root: {x}")
            print(f"Error: {f(x)}")
            return x
        x0 = x

    print("Function did not converge.")
    print(f"Root: {x}")
    print(f"Error: {f(x)}")
    return x

tent = lambda x: 2*x if x < 0.5 else 2 - 2*x

fixed_point_iteration(tent, 0.1, 100, 1e-20, True)

f = lambda x: 10 * x  - int(10 * x)

fixed_point_iteration(f, 0.347123, 100, 1e-20, True)
```

```
x_0 = 0.2
x_1 = 0.4
x_2 = 0.8
x_3 = 0.3999999999999999
x_4 = 0.7999999999999998
x_5 = 0.40000000000000036
x_6 = 0.8000000000000007
x_7 = 0.3999999999999986
x_8 = 0.7999999999999972
x_9 = 0.4000000000000057
x_10 = 0.8000000000000114
x_11 = 0.39999999999997726
x_12 = 0.7999999999999545
x_13 = 0.40000000000009095
x_14 = 0.8000000000001819
x_15 = 0.3999999999996362
x_16 = 0.7999999999992724
x_17 = 0.4000000000014552
x_18 = 0.8000000000029104
x_19 = 0.39999999999417923
x_20 = 0.7999999999883585
x_21 = 0.40000000002328306
x_22 = 0.8000000000465661
x_23 = 0.39999999990686774
x_24 = 0.7999999998137355
x_25 = 0.40000000037252903
x_26 = 0.8000000007450581
x_27 = 0.3999999985098839
x_28 = 0.7999999970197678
x_29 = 0.4000000059604645
x_30 = 0.800000011920929
x_31 = 0.3999999761581421
x_32 = 0.7999999523162842
x_33 = 0.40000009536743164
x_34 = 0.8000001907348633
x_35 = 0.39999961853027344
x_36 = 0.7999992370605469
x_37 = 0.40000152587890625
x_38 = 0.8000030517578125
x_39 = 0.399993896484375
x_40 = 0.79998779296875
x_41 = 0.4000244140625
x_42 = 0.800048828125
x_43 = 0.39990234375
x_44 = 0.7998046875
x_45 = 0.400390625
x_46 = 0.80078125
x_47 = 0.3984375
x_48 = 0.796875
x_49 = 0.40625
x_50 = 0.8125
x_51 = 0.375
x_52 = 0.75
x_53 = 0.5
x_54 = 1.0
x_55 = 0.0
x_56 = 0.0
Function converged after 56 iterations.
Root: 0.0
Error: 0.0
```

```
x_0 = 0.47123000000000026
x_1 = 0.7123000000000026
x_2 = 0.12300000000002598
x_3 = 0.23000000000025977
x_4 = 0.30000000000259774
x_5 = 2.5977442419389263e-11
x_6 = 2.5977442419389263e-10
x_7 = 2.5977442419389263e-09
x_8 = 2.5977442419389263e-08
x_9 = 2.5977442419389263e-07
x_10 = 2.5977442419389263e-06
x_11 = 2.5977442419389263e-05
x_12 = 0.00025977442419389263
x_13 = 0.0025977442419389263
x_14 = 0.025977442419389263
x_15 = 0.25977442419389263
x_16 = 0.5977442419389263
x_17 = 0.9774424193892628
x_18 = 0.774424193892628
x_19 = 0.7442419389262795
x_20 = 0.44241938926279545
x_21 = 0.4241938926279545
x_22 = 0.24193892627954483
x_23 = 0.4193892627954483
x_24 = 0.19389262795448303
x_25 = 0.9389262795448303
x_26 = 0.3892627954483032
x_27 = 0.8926279544830322
x_28 = 0.9262795448303223
x_29 = 0.26279544830322266
x_30 = 0.6279544830322266
x_31 = 0.2795448303222656
x_32 = 0.7954483032226562
x_33 = 0.9544830322265625
x_34 = 0.544830322265625
x_35 = 0.44830322265625
x_36 = 0.4830322265625
x_37 = 0.830322265625
x_38 = 0.30322265625
x_39 = 0.0322265625
x_40 = 0.322265625
x_41 = 0.22265625
x_42 = 0.2265625
x_43 = 0.265625
x_44 = 0.65625
x_45 = 0.5625
x_46 = 0.625
x_47 = 0.25
x_48 = 0.5
x_49 = 0.0
x_50 = 0.0
Function converged after 50 iterations.
Root: 0.0
Error: 0.0
```

```
Out[33]: 0.0
```