HW#2

Input:

import math

import numpy as np

*# ====================================================================*

def g\_func(x):

    return math.log(x - 1) + math.cos(x - 1)

*# ====================================================================*

def g\_func\_prime(x):

    return 1/(x-1) - math.sin(x - 1)

*# ====================================================================*

def newton(p\_0, TOL, N\_0):

*# declare and define i*

    i = 1

*# Newtons method after calling upon function and function prime values*

    while i <= N\_0:

        p = p\_0 - g\_func(p\_0)/g\_func\_prime(p\_0)

        if abs(p - p\_0) < TOL:

            return p

        i += 1

        p\_0 = p

    return 'Not enough Iterations'

*# =====================================================================*

def main():

*# Define constants and criteria*

    p\_0 = 1.5

    TOL = 10e-8

    N\_0 = 10

    root\_true = newton(p\_0, TOL, N\_0)

*# Find interval by iterating by a 0.001 until the root no longer can be found*

*# (this method did not work, however, and returned an epsilon of 0 around p\_0 = 1.5)*

*# Find right endpoint*

    i = 0

    while i < 100:

        root\_temp = newton(p\_0 + i - 0.001, TOL, N\_0)

        root = newton(p\_0 + i, TOL, N\_0)

        if root != root\_true:

            min\_a = root\_temp

            break

        i += 0.001

*# Find left endpoint*

    i = 0

    while i < 100:

        root\_temp = newton(p\_0 - i + 0.001, TOL, N\_0)

        root = newton(p\_0 - i, TOL, N\_0)

        if root != root\_true:

            max\_a = root\_temp

            break

        i -= 0.001

*# Output endpoints found*

    print("Left endpoint:", min\_a,"\nRight endpoint:", max\_a)

*# =======================================================================*

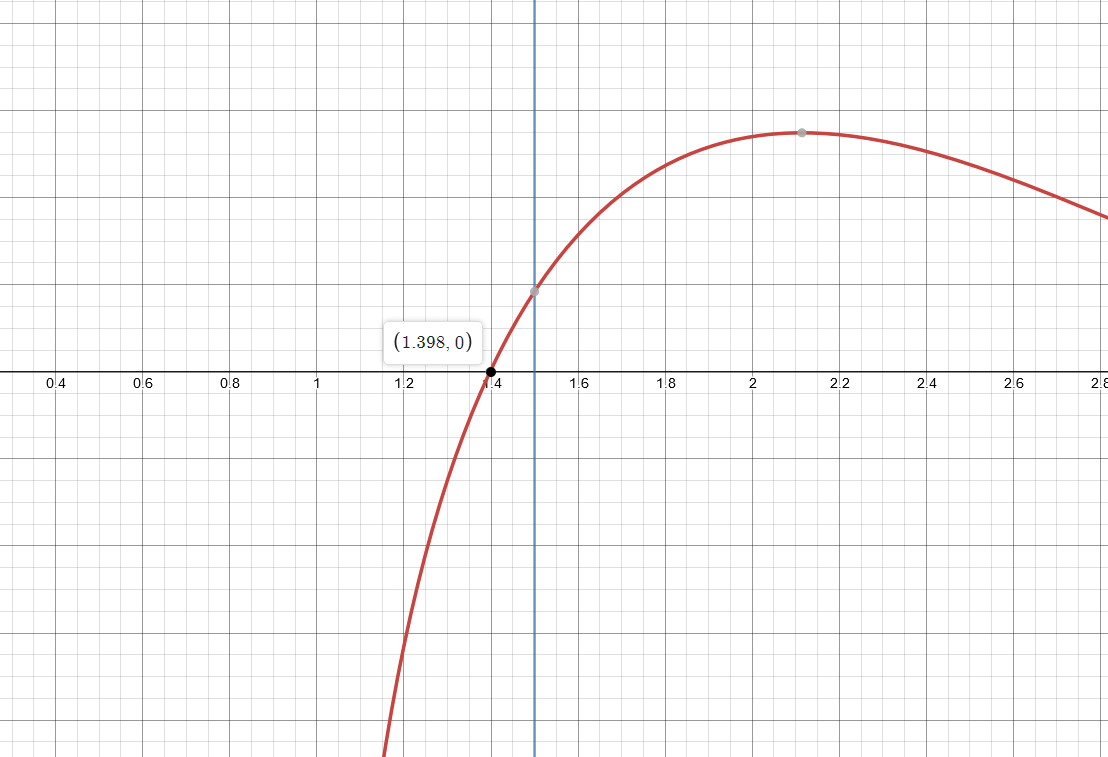
if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:



Below you can see the graph of the function. Here, if you select x\_0 = 1.5, then the root is approached from the right via newtons method. This is denoted by the blue line x = 1.5.



Input:

import math

*# ====================================================*

def g(x):

    return 2 + math.sin(x)

*# ====================================================*

def steff(p\_0, TOL, N\_0, ptrue):

*# Define and declare i*

    i = 1

*# Use Steffensen method and fixed point to approximate quadratically and output absolute error for each step*

    while i <= N\_0:

*# Call function*

        p\_1 = g(p\_0)

        p\_2 = g(p\_1)

*# Approximate via steffensen method*

        p = p\_0 - (p\_1 - p\_0)\*\*2 / (p\_2 - 2\*p\_1 + p\_0)

*# Find error*

        abs\_error = abs(ptrue - p)

        print('p value:', p, 'Absolute Error:', abs\_error)

*# Return approximation if below tolerence for accuracy*

        if abs(p - p\_0) < TOL:

            return p, i

        i += 1

        p\_0 = p

    return 'Error'

*# ====================================================*

def main():

*# Define constants and criteria*

    p\_0 = 2

    N\_0 = 100

    TOL = 10e-9

    ptrue = 2.554195952837043

*# Steffensen method*

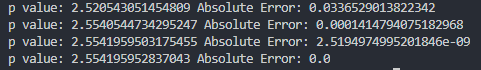
    pt, i = steff(p\_0, TOL, N\_0, ptrue)

*# ====================================================*

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:



Input:

import math

import numpy as np

*# ===================================================*

def g\_func(x):

    return x\*\*3 + 2\*x\*\*2 + 10\*x - 20

*# ===================================================*

def g\_func\_prime(x):

    return 3\*x\*\*2 + 4\*x + 10

*# ===================================================*

def f\_true():

*# Pisa's approxmiation*

    x = 1 + 22\*(1/60) + 7\*(1/60)\*\*2 + 42\*(1/60)\*\*3 + \

            33\*(1/60)\*\*4 + 4\*(1/60)\*\*5 + 40\*(1/60)\*\*6

    return x

*# ===================================================*

def newton(p\_0, TOL, N\_0, ptrue):

    i = 1

    while i <= N\_0:

        p = p\_0 - g\_func(p\_0)/g\_func\_prime(p\_0)

        if abs(p - p\_0) < TOL:

            print('Absolute Error:', abs(p - ptrue))

            return p, i

        i += 1

        p\_0 = p

    return 'Not enough Iterations'

*# ===================================================*

def main():

*# Define constants and criteria*

    p\_0 = 2

    TOL = 10e-14

    N\_0 = 100

*# Find true value (or at least what Pisa approximated)*

    ptrue = f\_true()

*# Find approimation via newton and compare to Pisa*

    pt, j = newton(p\_0, TOL, N\_0, ptrue)

*# ===================================================*

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:



Input:

import math

*# =====================================*

def g\_func(x):

    return math.pi + 0.5\*math.sin(x/2)

*# =====================================*

def fixed\_pt(p\_0, TOL, N\_0):

*# Define and declare i*

    i = 1

*# Iterate through by calling on g and using fixed point method*

    while i <= N\_0:

        p = g\_func(p\_0)

        if abs(p\_0 - p) < TOL:

            return p, i

        i += 1

        p\_0 = p

    return 'Error'

*# =====================================*

def main():

*# Define constants and criteria*

    p\_0 = 1

    TOL = 10e-10

    N\_0 = 100

*# Find approximation via fixed point method*

    pt, it\_num = fixed\_pt(p\_0, TOL, N\_0)

    print("Approximation:", pt, "Number of Iterations:", it\_num)

*# =====================================*

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:

