Name:- Soumya Ranjan Panda Roll No:- 246PH030

SUMMARY 1(a) Done

1(b) Done

1(c) Done

1(a) Done

1(e) Done

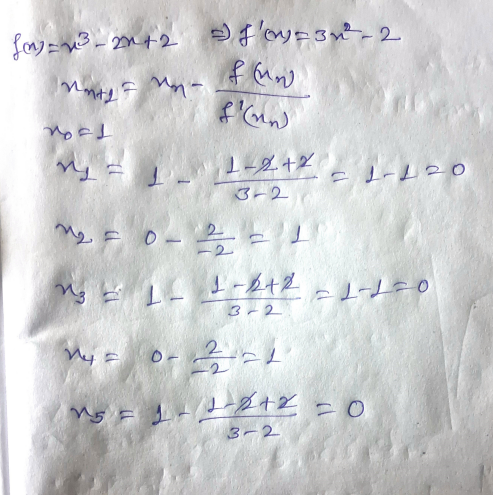
2(a) Done

2(b) Done

2(c) Done

2(d) Not done

1(A)



(B)

import numpy as np

import matplotlib.pyplot as plt

def f(x):

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

x = np.linspace(-2, 2, 4000)

y = f(x)

plt.plot(x, y, label="f(x) = x^3 - 2x + 2", color='g') plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x")

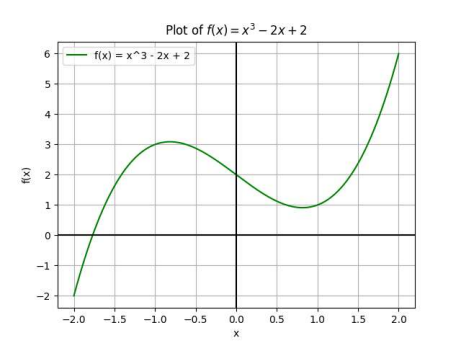
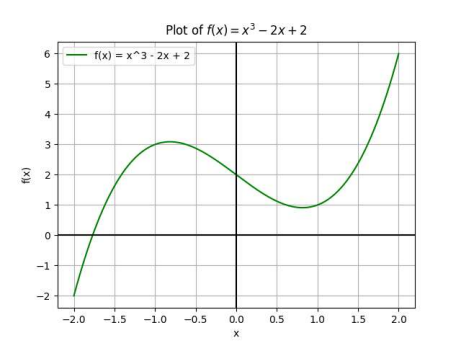
plt.ylabel("f(x)")

plt.title("Plot of $f(x) = x^3 - 2x + 2$")

plt.legend()

plt.grid(True)

plt.show()



The function we plotted doesn’t intersect positive x-axis and thus has no roots on the same. It is because f’(x) is close to zero around x = 1.

(C)

From the previous graph, we noticed that the graph intersects X-axis roughly at -1.7. Now, we take the initial value as -1.7

import numpy as np

def f(x):

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

def df(x):

return 3\*x\*\*2 - 2

def newton\_raphson():

x = float(input("Enter your guess:- "))

tol = 10\*\*-8

max\_iter = 100

for i in range(max\_iter):

fx = f(x)

dfx = df(x)

if dfx == 0:

print("Choose a different starting point.") return None

x\_1 = x - (fx / dfx)

if abs(x\_1 - x) < tol:

return x\_1, i + 1

x = x\_1

return x, max\_iter

root, iteration\_count = newton\_raphson() print("Final Root:- " , root )

print("Total Iterations:- " , iteration\_count )

OUTPUT:

Enter your guess:- -1.7

Final Root:- -1.7692923542386314 Total Iterations:- 4

(D)

import numpy as np

def f(x):

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

def bisection():

a = float(input("Enter your lower guess:- ")) b = float(input("Enter your upper guess:- ")) tol = 10\*\*-8

if f(a) \* f(b) >= 0:

print("Try a different interval") return 0

count = 0

while (b - a) / 2 > tol:

c = (a + b) / 2

if f(c) == 0:

return c, count

elif f(a) \* f(c) < 0:

b = c

else:

a = c

count += 1

return (a + b) / 2, count

root, count\_num = bisection() print("Root:- " , root)

print("Iterations:- " , count\_num)

OUTPUT:

Enter your lower guess:- -2.5 Enter your upper guess:- -1.5 Root:- -1.7692923471331596 Iterations:- 26

(E)

import numpy as np

import scipy.optimize as opt def f(x):

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

def df(x):

return 3\*x\*\*2 - 2

initial\_guess = -1.7

root =opt.newton(f, initial\_guess, fprime=df) print("Root:- " , root)

OUTPUT:

Root:- -1.7692923542386314

2(A)

import numpy as np

import matplotlib.pyplot as plt

# Define function

def f(x):

return x \* np.cos(x) - np.sin(x)

x = np.linspace(-20, 20, 1000)

y = f(x)

plt.plot(x, y, label="xcos(x) - sin(x)", color='g') plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.grid()

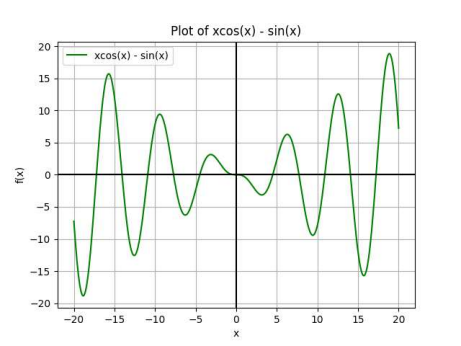
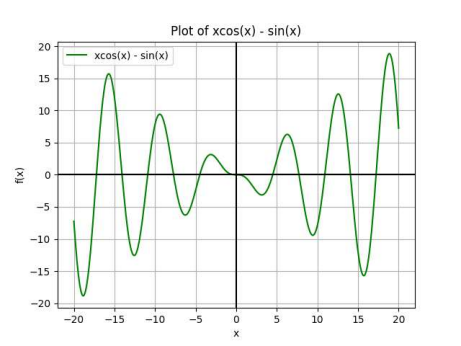
plt.legend()

plt.title("Plot of xcos(x) - sin(x)")

plt.xlabel("x")

plt.ylabel("f(x)")

plt.show()

There are 11 roots between -20 to 20

Those are roughly -17, -14, -11, -8, -4.5, 0, 4.5, 7.5, 11, 14, 17

(B)

import numpy as np

def f(x):

return x \* np.cos(x) - np.sin(x)

def df(x):

return -x \* np.sin(x)

def newton\_raphson():

x = float(input("Enter your guess:- ")) tol = 10\*\*-8

max\_iter = 100

for i in range(max\_iter):

fx = f(x)

dfx = df(x)

if dfx == 0:

print("Take a new initial point…") return None

x\_1 = x - (fx / dfx)

if abs(x\_1 - x) < tol:

return x\_1, i + 1

x = x\_1

return x, max\_iter

root, iteration\_count = newton\_raphson() print(f"Final Root:- " , root )

print("Total Iterations:- " , iteration\_count ) OUTPUT:

Enter your guess:- 0

Choose a different starting point.

Enter your guess:- 2

Final Root:- 1.8749296567218637e-08 Total Iterations:- 45

Enter your guess:- 3

Final Root:- -4.493409457909064 Total Iterations:- 5

Enter your guess:- 3.8

Final Root:- 4.493409457909064 Total Iterations:- 5

Enter your guess:- 5

Final Root:- 4.493409457909064 Total Iterations:- 4

(C)

import numpy as np

import scipy.optimize as opt

def f(x):

return x\*np.cos(x)-np.sin(x)

def df(x):

return -x \* np.sin(x)

initial\_guess = float(input(“Enter your initial guess:- ”)) root = opt.newton(f, initial\_guess, fprime=df)

print("Root:- " , root)

Result:-

For 0, Root:- 0.0

For 2, Root:- 2.4336055780755966e-08

For 3, Root:- -4.493409457909064

For 3.8, Root:- 4.493409457909064

For 5, Root:- 4.493409457909064

Except for initial value 2, roots for other initial values are same in both by N-R method and Scipy optimise method.