

#Question-1:

import numpy as np

from scipy.optimize import minimize\_scalar

# Defining constants

N = 100

q = 0.95

# Defining the function to minimize

def average\_tests(x):

return N \* (1 - q\*\*x + 1/x)

# Using a bounded minimization method to find the optimal x

result = minimize\_scalar(average\_tests, bounds=(1, 150), method='bounded')

# Printing the optimal group size and the corresponding average number of tests

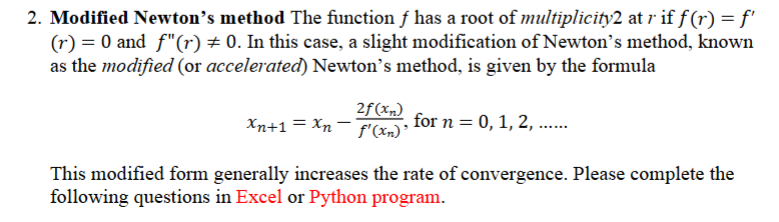
optimal\_x = result.x

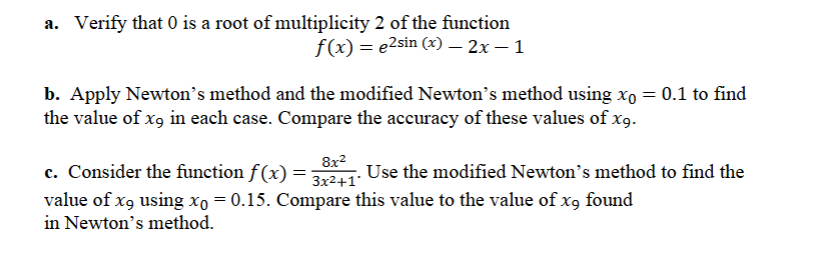
optimal\_tests = result.fun

print(f"Optimal group size (x): {optimal\_x:.2f}")

print(f"Minimum average number of tests: {optimal\_tests:.2f}")

Ans = 





#Question-2

import numpy as np

import sympy as sp

# Part (a): Verifying that 0 is a root of multiplicity 2 of the given function

x = sp.symbols('x')

f\_sym = sp.exp(2 \* sp.sin(x)) - 2 \* x - 1

f\_prime\_sym = sp.diff(f\_sym, x)

f\_double\_prime\_sym = sp.diff(f\_prime\_sym, x)

# Evaluating f(0), f'(0), f''(0)

root\_value = f\_sym.subs(x, 0)

first\_derivative = f\_prime\_sym.subs(x, 0)

second\_derivative = f\_double\_prime\_sym.subs(x, 0)

print("Part (a): Verifying root at x = 0")

print(f"f(0) = {root\_value}")

print(f"f'(0) = {first\_derivative}")

print(f"f''(0) = {second\_derivative}")

print("Conclusion: 0 is a root of multiplicity 2 since f(0) = 0, f'(0) = 0, and f''(0) ≠ 0.")

print("\n")

# Part (b): Newton's and Modified Newton's methods for f(x) = e^(2sin(x)) - 2x - 1

def f\_b(x):

return np.exp(2 \* np.sin(x)) - 2 \* x - 1

def df\_b(x):

return 2 \* np.exp(2 \* np.sin(x)) \* np.cos(x) - 2

# Newton's Method

def newtons\_method(func, dfunc, x0, iterations, epsilon=1e-10):

for i in range(iterations):

derivative = dfunc(x0)

if abs(derivative) < epsilon: # Avoid division by zero

print("Derivative near zero. Stopping iterations.")

break

x0 = x0 - func(x0) / derivative

return x0

# Modified Newton's Method

def modified\_newtons\_method(func, dfunc, x0, iterations, epsilon=1e-10):

for i in range(iterations):

derivative = dfunc(x0)

if abs(derivative) < epsilon: # Avoid division by zero

print("Derivative near zero. Stopping iterations.")

break

x0 = x0 - 2 \* func(x0) / derivative

return x0

# Initial guess and iterations for part (b)

x0\_b = 0.1

iterations = 9

x9\_newton = newtons\_method(f\_b, df\_b, x0\_b, iterations)

x9\_modified\_newton = modified\_newtons\_method(f\_b, df\_b, x0\_b, iterations)

print("Part (b): Finding x9 for both methods")

print(f"x9 (Newton's Method): {x9\_newton}")

print(f"x9 (Modified Newton's Method): {x9\_modified\_newton}")

print("\n")

# Part (c): Modified Newton's method for f(x) = (8x^2) / (3x^2 + 1)

def f\_c(x):

return (8 \* x\*\*2) / (3 \* x\*\*2 + 1)

def df\_c(x):

numerator = 16 \* x \* (3 \* x\*\*2 + 1) - 48 \* x\*\*3

denominator = (3 \* x\*\*2 + 1)\*\*2

return numerator / denominator

x0\_c = 0.15

x9\_part\_c = modified\_newtons\_method(f\_c, df\_c, x0\_c, iterations)

print("Part (c): Finding x9 using Modified Newton's Method for Part (c)")

print(f"x9 (Modified Newton's Method, Part c): {x9\_part\_c}")

Ans =

