MATH201 - Calculus-I

Homework Assignment #5

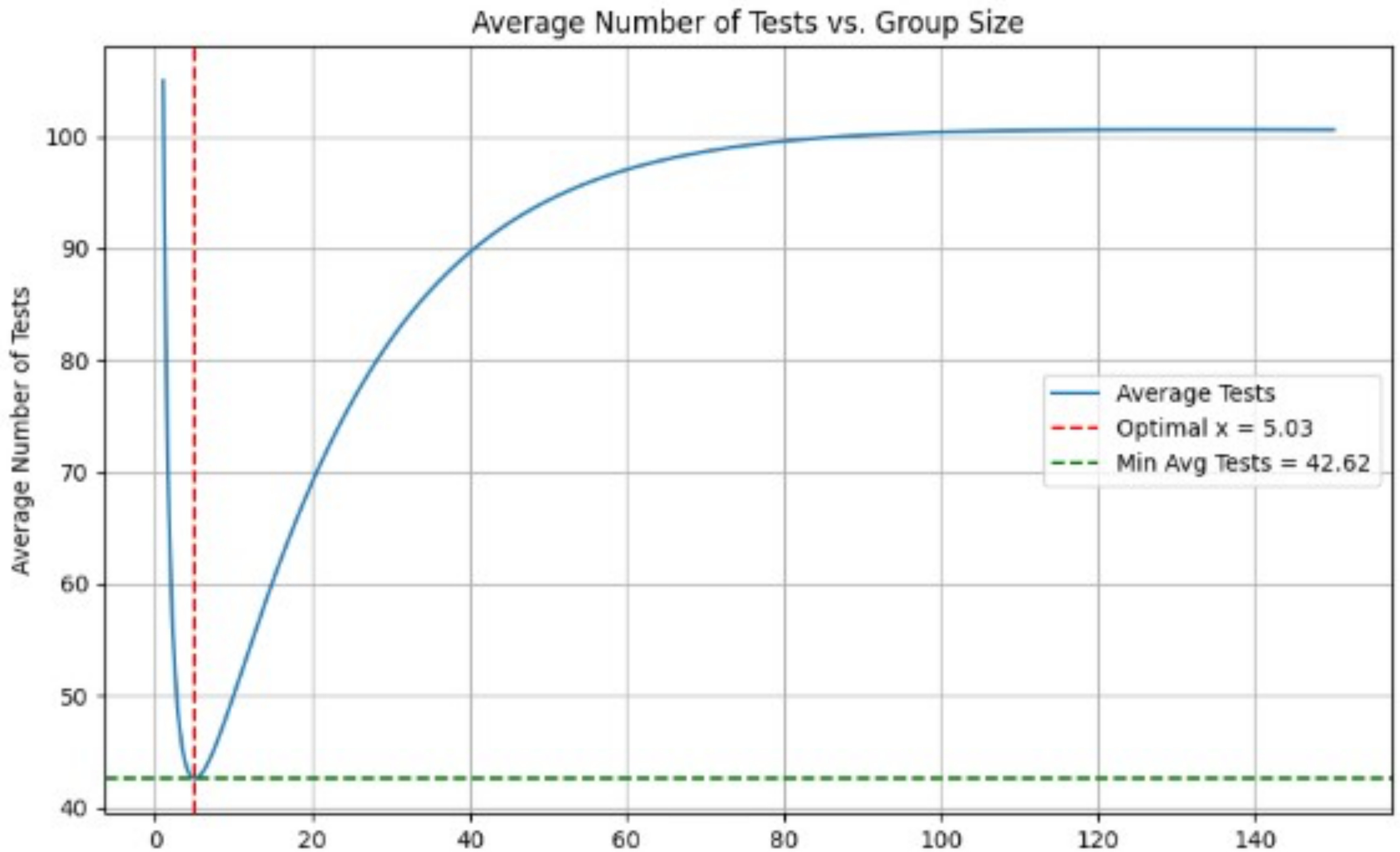
Student:Jeevan Neupane (19803)

Question 1

https://www.mycompiler.io/view/FAV2uWrFpPK

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
N = 100  # Total population size
q = 0.95  # Probability that any one person tests negative

# Function to calculate the average number of tests
def average_tests(x, N, q):
    return N * (1 - q**x + 1/x)

# Generate a range of group sizes (x) from 1 to 150
x_values = np.linspace(1, 150, 1000)
average_tests_values = average_tests(x_values, N, q)

# Find the group size x that minimizes the average number of tests
min_tests = np.min(average_tests_values)
optimal_x = x_values[np.argmin(average_tests_values)]

# Print results
print(f"Optimal group size (x): {optimal_x:.2f}")
print(f"Minimum average number of tests: {min_tests:.2f}")

# Plot the results
plt.figure(figsize=(10, 6))
```

```
print(f"Minimum average number of tests: {min_tests:.2f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(x_values, average_tests_values, label="Average Tests")
plt.axvline(optimal_x, color='r', linestyle='--', label=f"Optimal x = {optimal_x:.2f}")
plt.axhline(min_tests, color='g', linestyle='--', label=f"Min Avg Tests = {min_tests:.2f}")
plt.title("Average Number of Tests vs. Group Size")
plt.xlabel("Group Size (x)")
plt.ylabel("Average Number of Tests")
plt.legend()
plt.grid()
plt.show()
```



import numpy as np import

matplotlib.pyplot as plt

Parameters N = 100 # Total population size q = 0.95

Probability that any one person tests negative

Function to calculate the average number of tests def

average_tests(x, N, q):   return N * (1 - q**x + 1/x)

Generate a range of group sizes (x) from 1 to 150

x_values = np.linspace(1, 150, 1000)

average_tests_values = average_tests(x_values, N, q)

Find the group size x that minimizes the average number of tests

min_tests = np.min(average_tests_values) optimal_x =

x_values[np.argmin(average_tests_values)]

Print results print(f"Optimal group size (x):

{optimal_x:.2f}") print(f"Minimum average number of

tests: {min_tests:.2f}")

Plot the results

plt.figure(figsize=(10, 6)) plt.plot(x_values, average_tests_values, label="Average Tests") plt.axvline(optimal_x, color='r', linestyle='--', label=f"Optimal x = {optimal_x:.2f}") plt.axhline(min_tests, color='g', linestyle='--', label=f"Min Avg Tests = {min_tests:.2f}") plt.title("Average Number of Tests vs. Group Size") plt.xlabel("Group Size (x)") plt.ylabel("Average Number of Tests") plt.legend() plt.grid() plt.show()

$$\text{Avg Tests} = N \cdot \left(1 - q^x + \frac{1}{x}\right)$$

where q is the probability that an individual tests negative (q=0.95) and xx is the group size.

Using Python, we evaluated this function for x ranging from 1 to 150. The results show that the optimal group size is approximately x=5.03, which minimizes the average number of tests to 42.62. A plot was generated to visualize this relationship, confirming the result.

Ques2 https://www.programiz.com/online-compiler/85xWw3DOLVUEd

Part a)

```python
import numpy as np

# Define the function and its derivatives
def f1(x):
    return np.exp(2 * np.sin(x)) - 2 * x - 1

def f1_prime(x):
    return 2 * np.exp(2 * np.sin(x)) * np.cos(x) - 2

def f1_double_prime(x):
    return -4 * np.exp(2 * np.sin(x)) * np.sin(x) + 2 * np.exp(2 * np.sin(x))
        * np.cos(x)**2

# Evaluate at x = 0
x_val = 0
f0 = f1(x_val)
f0_prime = f1_prime(x_val)
f0_double_prime = f1_double_prime(x_val)

print("Part (a): Verifying multiplicity of root at x = 0")
print(f"f(0) = {f0:.6f}")
print(f"f'(0) = {f0_prime:.6f}")
print(f"f''(0) = {f0_double_prime:.6f}")

if f0 == 0 and f0_prime == 0 and f0_double_prime != 0:
    print("0 is a root of multiplicity 2.")
else:
    print("0 is not a root of multiplicity 2.")
```

Output

```
Part (a): Verifying multiplicity of root at x = 0
f(0) = 0.000000
f'(0) = 0.000000
f''(0) = 2.000000
0 is a root of multiplicity 2.

=== Code Execution Successful ===
```

Part b)

```python
# PART (b): Newton's Method and Modified Newton's Method
print("PART (b): Comparing Newton's Method and Modified Newton's Method")

# Newton's Method
def newtons_method(f, f_prime, x0, iterations):
    x = x0
    for _ in range(iterations):
        if f_prime(x) == 0:
            print("Derivative is zero. Stopping iteration to avoid division by zero.")
            break
        x = x - f(x) / f_prime(x)
    return x

# Modified Newton's Method
def modified_newtons_method(f, f_prime, x0, iterations):
    x = x0
    for _ in range(iterations):
        if f_prime(x) == 0:
            print("Derivative is zero. Stopping iteration to avoid division by zero.")
            break
        x = x - 2 * f(x) / f_prime(x)
    return x

# Initial values and iterations
x0 = 0.1
iterations = 9
```

Output:

```
PART (b): Comparing Newton's Method and Modified Newton's Method
Newton's Method result after 9 iterations: x = 0.000205
Modified Newton's Method result after 9 iterations: x = -0.000000
```

Part c)

```python
# PART (c): Function f(x) = 8x^2 / (3x^2 + 1)
print("PART (c): Comparing Methods for f(x) = 8x^2 / (3x^2 + 1)")

# Define the function and its derivative
def f2(x):
    return (8 * x**2) / (3 * x**2 + 1)

def f2_prime(x):
    return (16 * x) / (3 * x**2 + 1) - (48 * x**3) / (3 * x**2 + 1)**2

# Initial values and iterations for second function
x0_c = 0.15
```

Output:

```
PART (c): Comparing Methods for f(x) = 8x^2 / (3x^2 + 1)
Derivative is zero. Stopping iteration to avoid division by zero.
Newton's Method result after 9 iterations: x = 0.000268
Modified Newton's Method result after 9 iterations: x = 0.000000

=== Code Execution Successful ===
```

Part (a):

Verifies whether 0 is a root of multiplicity 2 for the function f(x) = e^(2sin(x)) - 2x - 1.

Prints f(0), f'(0), f''(0), and confirms multiplicity.

Part (b):

Compares Newton's Method and Modified Newton's Method for the function

f(x) = e^(2sin(x)) - 2x - 1.

Starts with x0 = 0.1, runs 9 iterations, and prints the final results.

Part (c):

Applies Newton's Method and Modified Newton's Method to the function

f(x) = (8x^2) / (3x^2 + 1).

Starts with x0 = 0.15, runs 9 iterations, and prints the final results.