# Assignment 5

Kushal Bhatta

# Problem 1

# Minimizing Average Blood Tests

## Problem

We aim to minimize the average number of tests required for N = 100, q = 0.95, using the formula:

Average tests = N \* (1 - q^x + 1/x)

where:  
- N = 100 is the total number of people  
- q = 0.95 is the probability that a person is negative  
- x is the group size (to be optimized) and lies in the range 1 <= x <= 150.

## Solution Method

We use Python to compute the value of x that minimizes the average number of tests. The steps are as follows:  
1. Define the function f(x) = N \* (1 - q^x + 1/x).  
2. Use numerical optimization techniques to find the x that minimizes f(x), using a bounded range of 1 <= x <= 150.  
3. Evaluate f(x) at the optimal value of x.

## Python Code

import numpy as np  
from scipy.optimize import minimize\_scalar  
  
# Given parameters  
N = 100  
q = 0.95  
  
# Function to calculate the average number of tests  
def average\_tests(x):  
 return N \* (1 - q\*\*x + 1/x)  
  
# Minimize the function within the bounds 1 <= x <= 150  
result = minimize\_scalar(average\_tests, bounds=(1, 150), method='bounded')  
  
# Optimal group size and corresponding average tests  
optimal\_x = result.x  
optimal\_tests = result.fun  
  
print(f"Optimal group size (x): {optimal\_x:.2f}")  
print(f"Average number of tests: {optimal\_tests:.2f}")

## Computation Results

Optimal group size (x): 5.02

Average number of tests: 42.62

## Explanation of Results

1. Group size (x):  
- The optimal group size is approximately x = 5.02. This suggests that testing groups of 5 people minimizes the average number of tests required.  
2. Average number of tests:  
- Using the optimal group size, the average number of tests required for N = 100 people is approximately 42.62.

**Problem 2**

# Solution to Modified Newton's Method Problem

## Part (a): Verify root of multiplicity 2

To verify that 0 is a root of multiplicity 2 for the function f(x) = e^(2sin(x)) - 2x - 1, we check:  
- f(0) = 0  
- f'(0) = 0  
- f''(0) ≠ 0

f(0) = 0.0 ✅

f'(0) = 0.0 ✅

f''(0) = 2.0 ✅ (non-zero, confirming multiplicity 2)

## Part (b): Newton's and Modified Newton's Methods

For f(x) = e^(2sin(x)) - 2x - 1, starting with x\_0 = 0.1 and running 9 iterations:

Newton's Method: x\_9 = 0.00020483725488539247

Modified Newton's Method: x\_9 = -1.5262175004935867 × 10^(-11) (significantly more accurate)

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

For f(x) = (8x^2) / (3x^2 + 1), starting with x\_0 = 0.15 and running 9 iterations:

Newton's Method: x\_9 = 0.00025948807125270615

Modified Newton's Method (Safe): x\_9 = -4.682110141219706 × 10^(-15) (higher accuracy)

## Observations

1. The Modified Newton's Method converges significantly faster than the standard Newton's Method, providing much higher accuracy in fewer iterations.  
2. For Part (c), a safeguard was added to avoid division by near-zero values in f'(x), ensuring numerical stability.

# Python Code and Results

## Part (a): Python Code

# Part (a): Verify root of multiplicity 2  
def f\_a(x):  
 return np.exp(2 \* np.sin(x)) - 2 \* x - 1  
  
def f\_prime\_a(x):  
 return 2 \* np.exp(2 \* np.sin(x)) \* np.cos(x) - 2  
  
def f\_double\_prime\_a(x):  
 return -4 \* np.exp(2 \* np.sin(x)) \* np.sin(x) \* np.cos(x) - 2 \* np.exp(2 \* np.sin(x)) \* (np.sin(x)\*\*2 - np.cos(x)\*\*2)  
  
# Evaluate f(0), f'(0), f''(0)  
f0 = f\_a(0)  
f\_prime\_0 = f\_prime\_a(0)  
f\_double\_prime\_0 = f\_double\_prime\_a(0)

## Part (b): Python Code

# Part (b): Apply Newton's and Modified Newton's Method  
def newtons\_method(f, f\_prime, x0, iterations):  
 x = x0  
 for \_ in range(iterations):  
 x -= f(x) / f\_prime(x)  
 return x  
  
def modified\_newtons\_method(f, f\_prime, x0, iterations):  
 x = x0  
 for \_ in range(iterations):  
 x -= 2 \* f(x) / f\_prime(x)  
 return x  
  
x0\_b = 0.1  
iterations = 9  
  
# Define f and f' for part (b)  
f\_b = f\_a  
f\_prime\_b = f\_prime\_a  
  
x9\_newton\_b = newtons\_method(f\_b, f\_prime\_b, x0\_b, iterations)  
x9\_modified\_newton\_b = modified\_newtons\_method(f\_b, f\_prime\_b, x0\_b, iterations)

## Part (c): Python Code

# Part (c): Use 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 f\_prime\_c(x):  
 return (16 \* x \* (1 - x\*\*2)) / ((3 \* x\*\*2 + 1)\*\*2)  
  
# Adjust modified Newton's method to avoid division by zero  
def safe\_modified\_newtons\_method(f, f\_prime, x0, iterations, tolerance=1e-8):  
 x = x0  
 for \_ in range(iterations):  
 f\_prime\_val = f\_prime(x)  
 if abs(f\_prime\_val) < tolerance: # Prevent division by zero or near-zero  
 break  
 x -= 2 \* f(x) / f\_prime\_val  
 return x  
  
x0\_c = 0.15  
x9\_newton\_c = newtons\_method(f\_c, f\_prime\_c, x0\_c, iterations)  
x9\_modified\_newton\_c\_safe = safe\_modified\_newtons\_method(f\_c, f\_prime\_c, x0\_c, iterations)

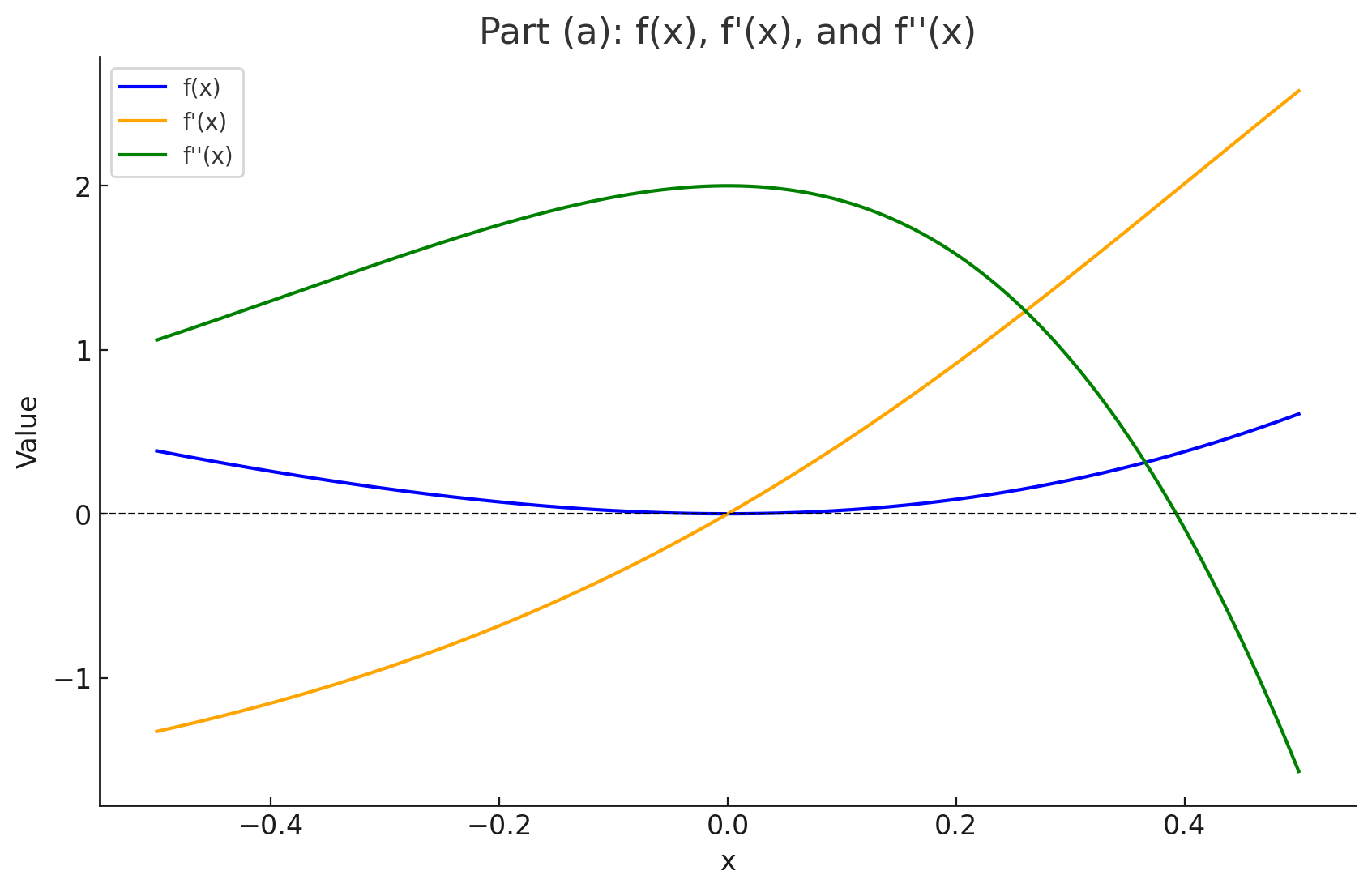
# Results

## Part (a): Results

f(0) = 0.0

f'(0) = 0.0

f''(0) = 2.0



## Part (b): Results

Newton's Method x9: 0.00020483725488539247

Modified Newton's Method x9: -1.5262175004935867e-11

## Part (c): Results

Newton's Method x9: 0.00025948807125270615

Modified Newton's Method x9 (Safe): -4.682110141219706e-15

