**Logo

Description automatically generated San Francisco Bay University**

**MATH201 - Calculus-I**

**Homework Assignment #5**

**Due day: 11/27/2024**

**Instruction:**

1. **Push the answer sheet to GitHub in word file**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. **Blood testing** Suppose a blood test for a disease is given to a population of *N* people, where *N* is large. At most, *N* individual blood tests must be done. The following strategy reduces the number of tests. Suppose 100 people are selected from the population and their blood samples are pooled. One test determines whether any of the 100 people test positive. If that test is positive, those 100 people are tested individually, making 101 tests necessary. However, if the pooled sample tests negative, then 100 people have been tested with one test. This procedure is then repeated. Probability theory shows that if the group size is *x* (for example, *x* = 100, as described here), then the average number of blood tests required to test *N* people is *N\**(1-), where *q* is the probability that any one person tests negative. What group size *x* minimizes the average number of tests in the case that *N* = 100 and *q* = 0.95? Assume *x* is a real number between 1 and 150 in Excel or Python program.

A screenshot of a computer program

Description automatically generated

N = 100 # Total number of people to test

q = 0.95 # Probability that one person tests negative

best\_group\_size = 0 # We'll store the group size with the fewest tests

min\_tests = float('inf') # Start with a very high number for comparison

# Loop through all possible group sizes (from 1 to 150)

for group\_size in range(1, 151): # Group sizes to test

# Calculate the average number of tests for this group size

# Formula:Tests = N \* (1 - (1 - q^group\_size) / group\_size)

tests = N \* (1 - (1 - q\*\*group\_size) / group\_size)

# Check if this group size gives fewer tests than we've seen so far

if tests < min\_tests:

min\_tests = tests # Update the minimum number of tests

best\_group\_size = group\_size # Update the best group size

print(f"Best group size: {best\_group\_size}")

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

1. **Modified Newton’s method** The function *ƒ* has a root of *multiplicity*2 at *r* if and . In this case, a slight modification of Newton’s method, known as the *modified* (or *accelerated*) Newton’s method, is given by the formula

, for

This modified form generally increases the rate of convergence. Please complete the following questions in Excel or Python program.

**a.**Verify that 0 is a root of multiplicity 2 of the function

**Step 1: Verify f(0)=0**

Plug x=0 into f(x):

f(0)=0^2(0−3)=0

So, x=0 is a root of f(x)

**Step 2: Find the First Derivative f′(x)**

The first derivative is:

f′(x)=d/dx[x^2(x−3)]=2x(x−3)+x^2

Simplify:

f′(x)=3x^2−6x

Now, substitute x=0 into f′(x):

f′(0)=3(0)^2−6(0)=0

So, f′(0)=0. This means x=0 is at least a **double root**.

**Step 3: Find the Second Derivative f′′(x)**

The second derivative is:

f′′(x)=d/dx[3x6^2−6x]=6x−6

Now, substitute x=0 into f′′(x):

f′′(0)=6(0)−6=−6

Since f′′(0)≠0, this confirms x=0 is a **root of multiplicity 2**.

**Final Answer for Part a:**

* f(0)=0: x=0 is a root.
* f′(0)=0: x=0 is at least a double root.
* f′′(0)≠0: Confirms x=0 is a **root of multiplicity 2**.

**b.** Apply Newton’s method and the modified Newton’s method using to find the value of in each case. Compare the accuracy of these values of .

The function is f(x)=x^2(x−3), and its derivative is f′(x)=3x^2−6x

1. **Newton’s Method Formula:**

xn+1=xn−[f(xn)/f′(xn)]

1. **Modified Newton’s Method Formula:**

xn+1=xn−[f(xn)/f′(xn)]⋅1/2

**Python Code for Calculation:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Define the function and its derivative

def f(x):

return x\*\*2 \* (x - 3)

def f\_prime(x):

return 3\*x\*\*2 - 6\*x

# Newton's Method

def newton\_method(x0, iterations):

x = x0

for \_ in range(iterations):

x = x - f(x) / f\_prime(x)

return x

# Modified Newton's Method

def modified\_newton\_method(x0, iterations):

x = x0

for \_ in range(iterations):

x = x - (f(x) / f\_prime(x)) \* 0.5

return x

# Initial guess and iterations

x0 = 2.5 # Starting point

iterations = 5 # Number of steps

# Calculate roots

root\_newton = newton\_method(x0, iterations)

root\_modified = modified\_newton\_method(x0, iterations)

print("Root using Newton's Method:", root\_newton)

print("Root using Modified Newton's Method:", root\_modified)

**c.** Consider the function . Use the modified Newton’s method to find the value of using = 0.15. Compare this value to the value of found

in Newton’s method.

**Steps:**

1. Use the formula for the modified method:

xn+1=xn−[f(xn)/f′(xn)]⋅1/2

Where f(x)=cos(x)−x and f′(x)=−sin(x)−1

1. Start with x0=0.15

**Python Code for Calculation:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

import math

# Define the function and its derivative

def g(x):

return math.cos(x) - x

def g\_prime(x):

return -math.sin(x) - 1

# Modified Newton's Method

def modified\_newton\_cos(x0, iterations):

x = x0

for \_ in range(iterations):

x = x - (g(x) / g\_prime(x)) \* 0.5

return x

# Initial guess and iterations

x0 = 0.15

iterations = 5

# Calculate root

root\_cos = modified\_newton\_cos(x0, iterations)

print("Root using Modified Newton's Method for cos(x) - x:", root\_cos)