CL 603 Optimization **Autumn 2024**

Homework 1 : Preliminary Programming

Priyam Nayak - 214026014
priyamnayak@iitb.ac.in

14 August 2024

---

1. Consider a univariate function $f(x) = \frac{-0.1x}{(1+0.1x)(1+0.05x)}$. Plot a graph of this function for $x \in [0, 30]$.

Plot:

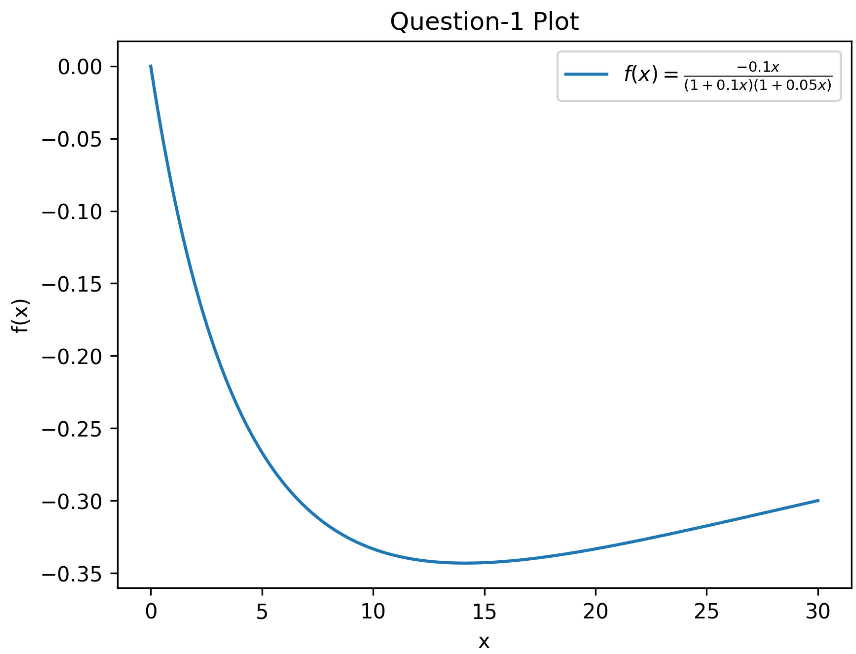

Figure 1: Plot of $f(x) = \frac{-0.1x}{(1+0.1x)(1+0.05x)}$ for $x \in [0, 30]$

2. Consider a univariate function $f(x) = cos^2(x) + 0.1x$. Plot a graph of this function for $x \in [-5, 5]$.

Plot:

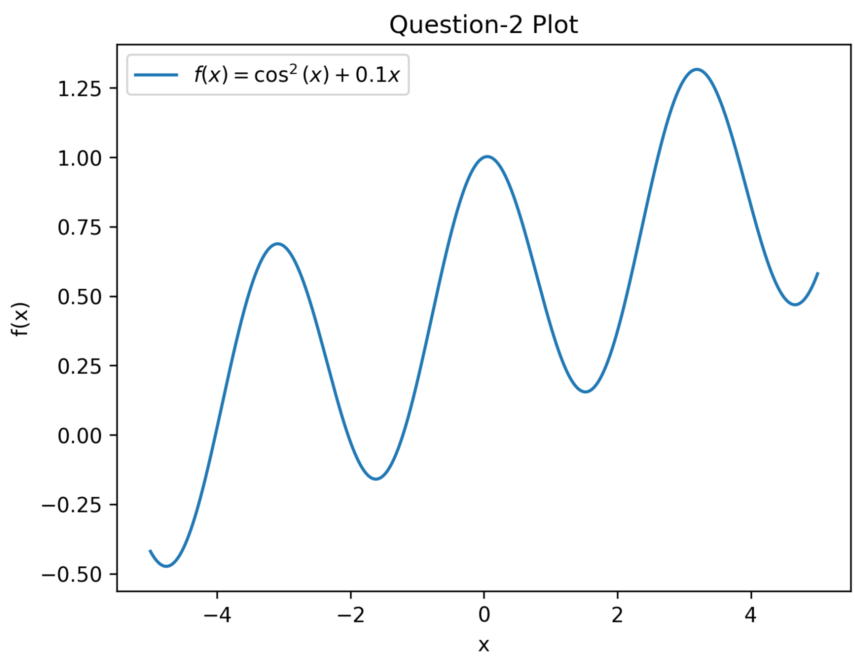

Figure 2: Plot of $f(x) = cos^2(x) + 0.1x$ for $x \in [-5, 5]$

3. Consider the following multivariate function:

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1^2 + x_2^2 - 7)^2 \tag{1}$$

Plot a contour plot of this function for $x(i) \in [-6, 6]$.

Plot:

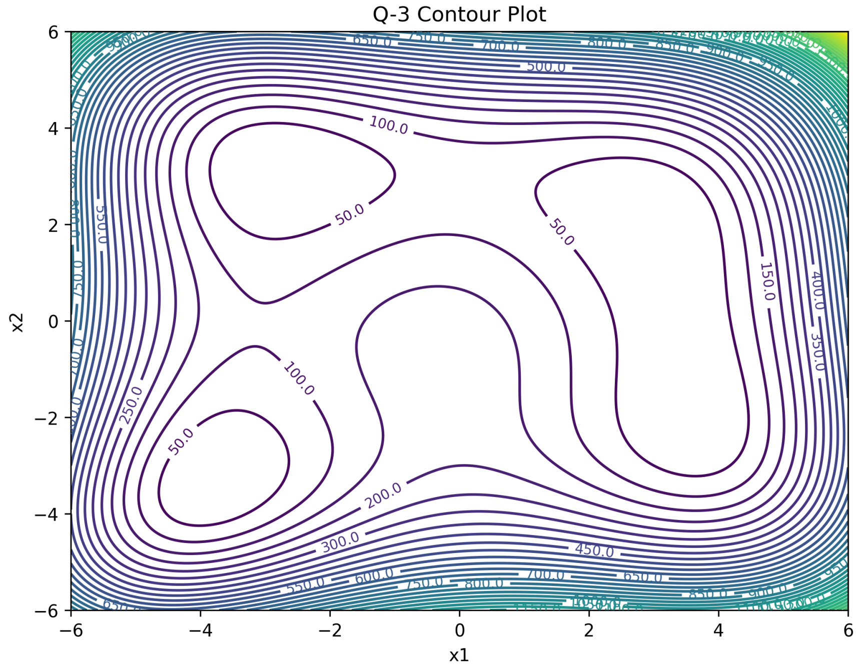

Figure 3: Plot of $f(x) = (x_1^2 + x_2 - 11)^2 + (x_1^2 + x_2^2 - 7)^2$ for $x_i \in [-6, 6]$

4. Given a multivariate function $f(x)$, write a code to compute its gradient and Hessian at a point. Use the center difference formula to compute the numerical derivative. You can test the code using the function given in Eq. (1). At point (1, 2), the following answer is expected:

$$\nabla f = \begin{bmatrix} -36 \\ -32 \end{bmatrix}$$

$$\nabla^2 f = \begin{bmatrix} -22 & 12 \\ 12 & 26 \end{bmatrix}$$

Answer:
Gradient at (1, 2): [-36, -32]

Hessian at (1, 2): [[-22.00000182, 12.00000099], [12.00000099, 25.9999311]]

5. Given a symmetric matrix, write a code to determine its definiteness using a Gauss elimination-based approach. Use this code to determine if the Hessian obtained in the above question is positive definite or not.

Answer:
Hessian Matrix: [[-22, 12], [12, 26]] is Indefinite

A Python Code

A.1 Code for plotting of $f(x) = \frac{-0.1x}{(1+0.1x)(1+0.05x)}$ for $x \in [0, 30]$

```
#CL 603 - Optimization
#Homework 1 - Problem 1
#Priyam Nayak
#214026014


import numpy as np # Import the numpy library and assign it the name 'np'
import matplotlib.pyplot as plt # Import the matplotlib.pyplot module and assign it the name 'plt'


x = np.linspace(0,30,600) # Create an array of 600 evenly spaced values from 0 to 30
fx = -0.1*x/((1+0.1*x)*(1+0.05*x)) # Calculate the function values fx for each x, using the given mathematical expression

plt.plot(x,fx,label=r'$f(x) = \frac{-0.1x}{(1+0.1x)(1+0.05x)}$') # Create a line plot of fx against x
plt.title("Question-1 Plot") # Set the title of the plot
plt.xlabel("x") # Set the label for the x-axis
plt.ylabel("f(x)") # Set the label for the y-axis
plt.legend() #Create legend
plt.savefig('Q1_Plot.png', dpi=300, bbox_inches='tight') #Save the plot as png file
plt.show() # Display the plot
```

A.2 Code for plotting of $f(x) = \cos^2(x) + 0.1x$ for $x \in [-5, 5]$

```
#CL 603 - Optimization
#Homework 1 - Problem 2
#Priyam Nayak
#214026014

import numpy as np # Import the numpy library and assign it the name 'np'
import matplotlib.pyplot as plt # Import the matplotlib.pyplot module and assign it the name 'plt'

x = np.linspace(-5,5,1000) # Create an array of 1000 evenly spaced values from -5 to 5
fx = np.cos(x)**2 + 0.1*x # Calculate the function values fx for each x, using the given mathematical expression

plt.plot(x,fx,label=r'$f(x)=\cos^2(x)+0.1x$') # Create the plot of fx against x
plt.title("Question-2 Plot") # Set the title of the plot
plt.xlabel("x") # Set the label for the x-axis
plt.ylabel("f(x)") # Set the label for the y-axis
plt.legend() #Create legend
plt.savefig('Q2_Plot.png', dpi=300, bbox_inches='tight') #Save the plot as png file
plt.show() # Display the plot
```

A.3 Code for plotting of $f(x) = (x_1^2 + x_2 - 11)^2 + (x_1^2 + x_2^2 - 7)^2$ for $x_i \in [-6, 6]$

```
#CL 603 - Optimization
#Homework 1 - Problem 3
#Priyam Nayak
#214026014

import numpy as np # Import the numpy library and assign it the name 'np'
import matplotlib.pyplot as plt # Import the matplotlib.pyplot module and assign it the name 'plt'

# Define the function
def f(x1, x2):
    return (x1**2 + x2 - 11)**2 + (x1 + x2**2 - 7)**2

# Generate the grid of values
x1 = np.linspace(-6, 6, 1000) # Create an array of 1000 evenly spaced values from -6 to 6 for x1
```

```python
x2 = np.linspace(-6, 6, 1000) # Create an array of 1000 evenly spaced values from -6 to 6 for x1
X1, X2 = np.meshgrid(x1, x2) # Create a meshgrid from x1 and x2 for plotting
Z = f(X1, X2) # Compute the function values over the grid

# Plot the contour
plt.figure(figsize=(8, 6)) # Create a new figure with a specific size
contour = plt.contour(X1, X2, Z, levels=50, cmap='viridis') # Plot the contour with 50 levels and 'viridis' colormap
plt.clabel(contour, inline=True, fontsize=8, fmt="%.1f")  # Display function values along contours
plt.title('Q-3 Contour Plot') # Set the title of the plot
plt.xlabel('x1') # Set the label for the x-axis
plt.ylabel('x2') # Set the label for the y-axis
#plt.legend() #Create legend
plt.savefig('Q3_Plot.png', dpi=300, bbox_inches='tight') #Save the plot as png file
plt.show() # Display the plot
```

A.4 Code for Computing Gradient and Hessian using Center Difference Formula

```python
#CL 603 - Optimization
#Homework 1 - Problem 4
#Priyam Nayak
#214026014

import numpy as np # Import the numpy library and assign it the name 'np'

# Define the function
def f(x):
    return (x[0]**2 + x[1] - 11)**2 + (x[0] + x[1]**2 - 7)**2

# Gradient calculation using the center difference method
def gradient(f, x, h=1e-5):
    grad = np.zeros_like(x)  # Initialize the gradient vector with zeros, same shape as x
    for i in range(len(x)):  # Loop over each variable in x
        x_f = x.copy()  # Create a copy of x to modify for forward difference
        x_b = x.copy()  # Create a copy of x to modify for backward difference
        x_f[i] += h  # Increment the i-th variable by h for forward difference
        x_b[i] -= h  # Decrement the i-th variable by h for backward difference
        grad[i] = (f(x_f) - f(x_b)) / (2 * h)  # Compute the central difference for the i-th variable
    return grad  # Return the gradient vector


# Hessian using center difference method
def hessian(f, x, h=1e-5):
    n = len(x)  # Determine the number of variables in x
    hess = np.zeros((n, n))  # Initialize the Hessian matrix with zeros, shape (n, n)
    for i in range(n):  # Loop over each row index of the Hessian matrix
        for j in range(n):  # Loop over each column index of the Hessian matrix
            x_ijpp = x.copy()  # Create a copy of x to modify for both i and j incremented
            x_ijpp[i] += h
            x_ijpp[j] += h
            x_ijnp = x.copy()  # Create a copy of x to modify for i decremented and j incremented
            x_ijnp[i] -= h
            x_ijnp[j] += h
            x_ijpn = x.copy()  # Create a copy of x to modify for i incremented and j decremented
            x_ijpn[i] += h
            x_ijpn[j] -= h
            x_ijnn = x.copy()  # Create a copy of x to modify for both i and j decremented
            x_ijnn[i] -= h
            x_ijnn[j] -= h
            hess[i, j] = (f(x_ijpp) - f(x_ijnp) - f(x_ijpn) + f(x_ijnn)) / (4 * h**2)  # Compute the second partial derivative for f with respect to x_i and x_j
    return hess  # Return the Hessian matrix
```

```

x = np.array([1.0, 2.0]) # Test at point (1, 2)
grad_f = gradient(f, x)
hess_f = hessian(f, x)
#np.savetxt('Q4_Result.txt', grad_f, fmt='%.6f', header='Gradient Matrix:', comments='')
# Write results to a text file with rounded integers
with open("Q4_Result.txt", "w") as file:
    file.write("Gradient at (1, 2):\n")
    file.write(np.array2string(np.round(grad_f).astype(int), separator=', '))
    file.write("\n\nHessian at (1, 2):\n")
    file.write(np.array2string(hess_f, separator=', '))

print("Gradient at (1, 2):", np.round(grad_f).astype(int))
print("Hessian at (1, 2):\n", hess_f)
```

A.5 Code to determine the Definiteness of a Symmetric Matrix using Gauss Elimination-based Approach

```
#CL 603 - Optimization
#Homework 1 - Problem 5
#Priyam Nayak
#214026014

import numpy as np # Import the numpy library and assign it the alias 'np'

def to_ut(A): #Function to convert a symmetric matrix A to its upper triangular form using Gaussian elimination.
    A = A.astype(float)  # Ensure the matrix is treated as floating-point
    n = A.shape[0]  # Get the number of rows (and columns) in the matrix
    for i in range(n):
        # Perform row elimination to create zeros below the diagonal
        for j in range(i + 1, n):
            if A[i, i] == 0:  # If the pivot element is zero, skip to avoid division by zero
                continue
            factor = A[j, i] / A[i, i]  # Calculate the factor for row elimination
            A[j, i:] -= factor * A[i, i:]  # Subtract the factor times the pivot row from the current row
    return A

def cons_dm(utm): # Function to construct a diagonal matrix from the pivot elements of the upper triangular matrix.
    n = utm.shape[0]  # Get the number of rows (and columns)
    dm = np.zeros((n, n))  # Initialize an n x n zero matrix
    # Extract the diagonal elements (pivot elements) and place them in the diagonal matrix
    for i in range(n):
        dm[i, i] = utm[i, i]
    return dm

def check_def(dm): # Function to check the definiteness of a matrix based on its pivot elements
    pivots = np.diag(dm)  # Extract the diagonal elements (pivot elements)
    if all(p > 0 for p in pivots):
        return "Positive definite"
    elif all(p < 0 for p in pivots):
        return "Negative definite"
    elif all(p >= 0 for p in pivots):
        return "Positive semi-definite"
    elif all(p <= 0 for p in pivots):
        return "Negative semi-definite"
    else:
        return "Indefinite"

# Hessian matrix from Q4
Hessian = np.array([[-22, 12],
                    [12, 26]])

utm = to_ut(Hessian.copy()) # Convert to upper triangular matrix
dm = cons_dm(utm) # Construct diagonal matrix using the pivot elements
```

```
definiteness = check_def(dm) # Check definiteness based on the diagonal matrix

with open("Q5_Result.txt", "w") as file:
    file.write("Hessian Matrix:\n")
    file.write(np.array2string(Hessian, separator=', ') +"\n")
    file.write(f"is {definiteness}\n")

print("Hessian Matrix from Q4:")
print(Hessian)
print(f"is {definiteness}")
```