Department Of
CREATIVE
TECHNOLOGIES
AIR UNIVERSITY
FCAI, ISLAMABAD

Multi Variable Calculus

Spring Semester Project

BSDS-IV

Date of Submission: 29/5/2025

Submitted To: Dr. Javeria Akram

Members:	
ID	**Name**
232542	Abdul Wahab
232546	Arhum Ahmad
232550	Abdullah Hassan

Table of Content

Table of Content **2**
1. Group Member Contributions **3**
2. Problem Statements **3**
Question #1 3
Question #2 4
Question #3 4
3. Problem Solving Using Software **5**
Question #1 5
CODE 5
OUTPUT 7
GRAPH 7
Question #2 8
CODE 8
OUTPUT 9
GRAPH 10
Question #3 10
CODE 10
OUTPUT 12
GRAPH 12
4. Hand Calculations **13**
5. Comparison Between Software and Hand Calculations Results **13**
5. Difficulties Faced and How They Were Overcome **13**

1. Group Member Contributions

Member	Contribution
Abdul Wahab	Performed all **hand calculations** for the given problems, including step-by-step derivations and final results.
Abdullah Hassan	Helped with **writing and debugging the Python code,** generating **plots**, and validating results where needed.
Arhum Ahmad	Provided **line-by-line explanations** of the code, prepared the **report structure**, and handled **comparison & formatting**.

2. Problem Statements

Question #1

The total latency (response time) in a system with two servers is modeled by:

$$L(x,y) = \frac{x^2 + y^2}{x + y + 1}$$

where x and y are the number of concurrent connections on Server 1 and Server 2, respectively. Domain: $1 \le x \le 20$, $1 \le y \le 20$

Use a any suitable software (MATLAB, Python) to perform the following steps:

A. a. Plot a 3D surface plot of the latency function over the given rectangle.
B. b. Calculate the function's first partial derivatives and use the equation solver to find

C. the critical points.
D. c. Find the critical points along the boundaries by considering the boundaries one by
E. one. (Mainly do this step by hand. Only find the derivatives by using software)
F. d. Perform the steps (a-c) by hand as well.
G. e. Find the optimal distribution of load (x, y) that minimizes latency.

Question #2

The Cobb-Douglas production function for software manufacturer is given by

$$f(x,y) = 100x^{\frac{3}{4}}y^{\frac{1}{4}}$$

Where x represents the unit of labor (at $150 per unit) and y represents the unit of capital (at $250 per unit). The total cost of labor and capital is limited to $50000. Find the maximum production level for this manufacturer using the Lagrange multiplier.

A. Where x represents the unit of labor (at $150 per unit) and y represents the unit of capital (at $250 per unit). The total cost of labor and capital is limited to $50000. Find the maximum production level for this manufacturer using the Lagrange multiplier.
B. Show the calculations of ∇f and ∇g using any suitable software.
C. Plot a 3D surface plot of f(x, y) using any suitable software.

Question #3

An electronics manufacturer determines that the profit P (in dollars) obtained by producing and selling x units of a DVD player and y units of a DVD recorder is

approximated by the model

$$P(x, y) = 8x + 10y - 0.001(x^2 + xy + y^2) - 10000$$

A. Plot a 3D surface plot of profit function.
B. Calculate the first order and second order partial derivatives using any software.
C. Find the production level that produces a maximum profit. What is the maximum profit?

3. Problem Solving Using Software

Question #1

CODE

```
#Importing required libraries to perform calculus operations
#Numpy for array and quick computations
#Matplotlib for plotting the graphs of functions
#Sympy for working on calculus questions
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, diff, Eq, solve, lambdify

#Creating symbols for solving equations. As python doesnt treat x, y, etc as mathematical variables
#Symbols function define that x, y are mathematical variables. Real means that x, y are real numbers
x, y = symbols('x y', real=True)
#Function from question
L = (x**2 + y**2) / (x + y + 1)

# Part (a): 3D Surface Plot
#Creating 100 x and y values spaced evenly for plotting. Range from 25 to 50
X_vals = np.linspace(25, 50, 100)
Y_vals = np.linspace(25, 50, 100)
#Creates a grid of x and y vals
X, Y = np.meshgrid(X_vals, Y_vals)
```

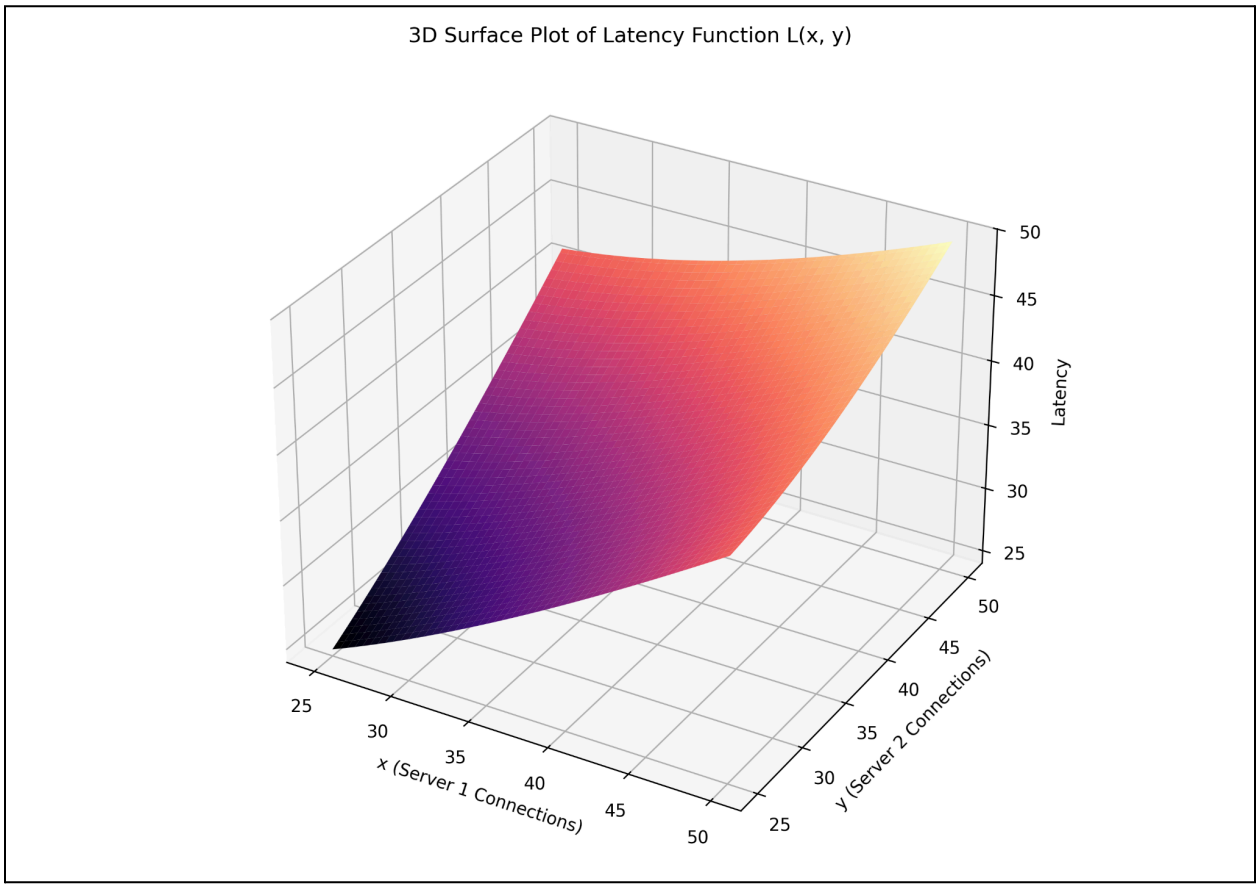
```
#Converting the mathematical function to a python function for evaluation
L_func = lambdify((x, y), L, modules='numpy')
#Calculating all values for every point in gridspace
Z = L_func(X, Y)
#Value of function for x=3 , y=5
print(f'{L_func(3,5):.2f}')

#Plotting the Function Graph
fig = plt.figure(figsize=(10, 7))
#Adds a 3d subplot
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='magma')
ax.set_title('3D Surface Plot of Latency Function L(x, y)')
ax.set_xlabel('x (Server 1 Connections)')
ax.set_ylabel('y (Server 2 Connections)')
ax.set_zlabel('Latency')
plt.tight_layout()
plt.show()

# Part (b): Partial derivatives and critical points
#diff calculates partial derivative. E.g. 1st one calculates partial L w.r.t x
Lx = diff(L, x)
Ly = diff(L, y)
#Calculating all C.Ps
critical_points = solve([Eq(Lx, 0), Eq(Ly, 0)], (x, y), dict=True)
print("Critical points from partial derivatives (may be outside domain):")
print(critical_points)

# Part (c): Check boundary values
boundary_points = []

#Calculates and stores the latency for all y-values when x = 25 and x = 50.
for y_val in range(25, 51):
    boundary_points.append((25, y_val, L_func(25, y_val)))
    boundary_points.append((50, y_val, L_func(50, y_val)))

#Calculates and stores the latency for all x-values when y = 25 and y = 50
for x_val in range(25, 51):
    boundary_points.append((x_val, 25, L_func(x_val, 25)))
    boundary_points.append((x_val, 50, L_func(x_val, 50)))

#Finding points with minimum latency
min_boundary_point = min(boundary_points, key=lambda item: item[2])
print("Minimum latency on the boundary:")
```

```
print(f"x = {min_boundary_point[0]}, y = {min_boundary_point[1]}, L =
{min_boundary_point[2]:.4f}")

# Part (e): Full search over domain
#Searching for all points. Just because we can
full_domain_points = [(xv, yv, L_func(xv, yv)) for xv in range(25, 56) for yv in range(25, 56)]
min_latency_point = min(full_domain_points, key=lambda item: item[2])
print("Global minimum latency in domain:")
print(f"x = {min_latency_point[0]}, y = {min_latency_point[1]}, L = {min_latency_point[2]:.4f}")
```

OUTPUT

```
Critical points from partial derivatives (may be outside domain):
[{x: -1, y: -1}, {x: -1, y: -1}, {x: 0, y: 0}, {x: 0, y: 0}]
Minimum latency on the boundary:
x = 25, y = 25, L = 24.5098
Global minimum latency in domain:
x = 25, y = 25, L = 24.5098
```

GRAPH



Question #2

CODE

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, diff, Eq, solve, lambdify

#Defining symbols. Here true means that symbols represent positive value only. Like in case of length,etc
x, y, λ = symbols('x y lambda', real=True, positive=True)

#Defining production function and constraint
f = 100 * x**(3/4) * y**(1/4)                    #Cobb-Douglas production function
g = 150 * x + 250 * y - 50000                      #Budget constraint

#Calculating partial derivatives w.r.t x & y
df_dx = diff(f, x)
df_dy = diff(f, y)
dg_dx = diff(g, x)
dg_dy = diff(g, y)

#Storing partial derivatives in pairs
grad_f = (df_dx, df_dy)
grad_g = (dg_dx, dg_dy)

#Lagrange multiplier equations
eq1 = Eq(df_dx, λ * dg_dx)
eq2 = Eq(df_dy, λ * dg_dy)
eq3 = Eq(g, 0)

#Solving system of equations
solution = solve((eq1, eq2, eq3), (x, y, λ), dict=True)[0]

#Extracting values
opt_x = solution[x]
opt_y = solution[y]
opt_production = f.subs({x: opt_x, y: opt_y})

#Printing Solution
print("1. Optimal solution using Lagrange multiplier:")
print(f"   x = {opt_x:.2f}, y = {opt_y:.2f}, max production ≈ {opt_production.evalf():.2f}\n")

print("2. Gradients:")
```

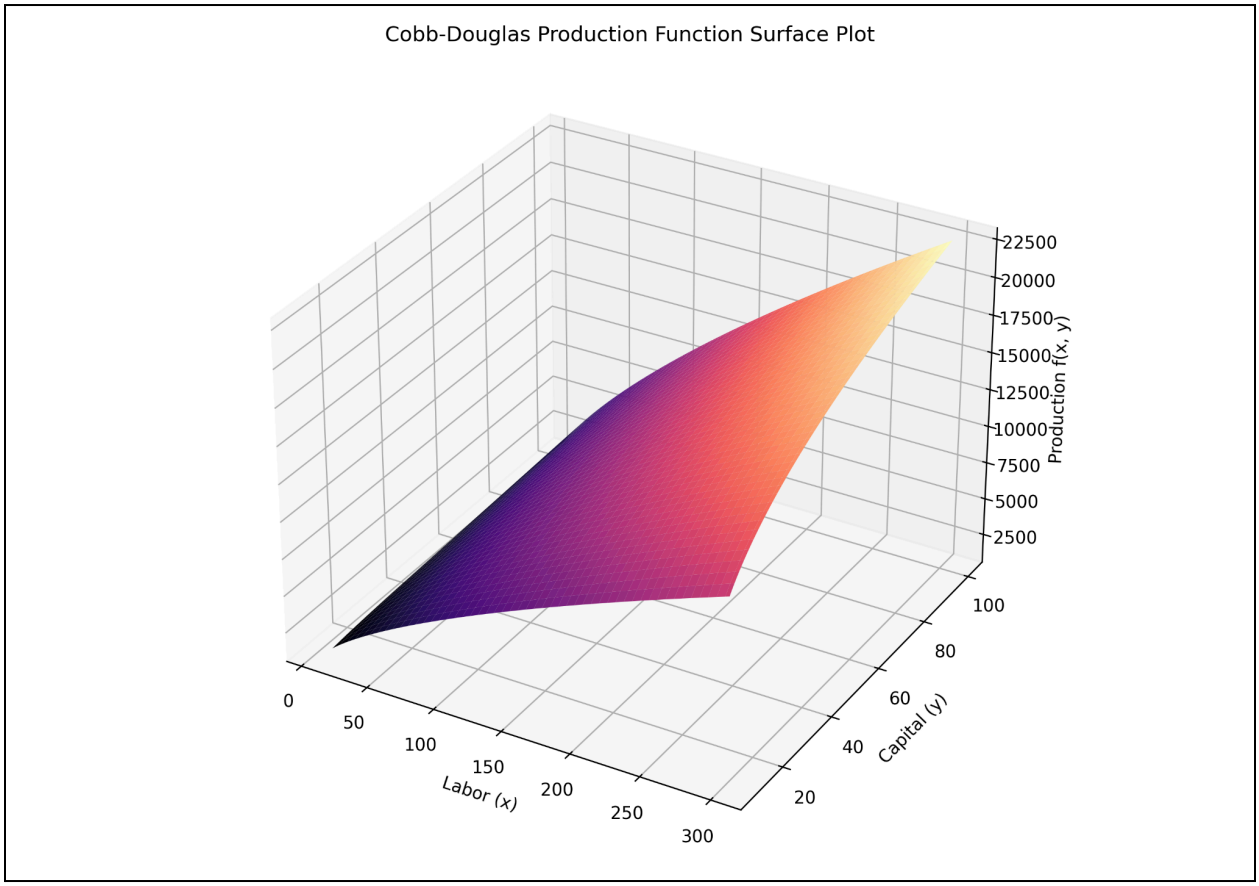
```
print(f"   ∇f = ({df_dx}, {df_dy})")
print(f"   ∇g = ({dg_dx}, {dg_dy})\n")

# Part 3: 3D Surface Plot
#Converting the mathematical function to a python function
f_func = lambdify((x, y), f, modules='numpy')
#Creating 100 x and y values spaced evenly for plotting. Range from 10 to 300
x_vals = np.linspace(10, 300, 100)
y_vals = np.linspace(10, 100, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f_func(X, Y)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='magma', edgecolor='none')
ax.set_title('Cobb-Douglas Production Function Surface Plot')
ax.set_xlabel('Labor (x)')
ax.set_ylabel('Capital (y)')
ax.set_zlabel('Production f(x, y)')
plt.tight_layout()
plt.savefig('plot2.png', dpi=300)
plt.show()
```

OUTPUT

1. Optimal solution using Lagrange multiplier:
   x = 250.00, y = 50.00, max production ≈ 16718.51

2. Gradients:
   ∇f = (75.0*y**0.25/x**0.25, 25.0*x**0.75/y**0.75)
   ∇g = (150, 250)

GRAPH



Question #3

CODE

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, diff, Eq, solve, lambdify

#Defining variables and profit function
x, y = symbols('x y', real=True)
P = 8*x + 10*y - 0.001*(x**2 + x*y + y**2) - 10000

# Part 1: 3D Surface Plot
```

```
P_func = lambdify((x, y), P, modules='numpy')
x_vals = np.linspace(0, 30000, 100)
y_vals = np.linspace(0, 30000, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = P_func(X, Y)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='magma', edgecolor='none')
ax.set_title('Profit Function Surface Plot')
ax.set_xlabel('DVD Players (x)')
ax.set_ylabel('DVD Recorders (y)')
ax.set_zlabel('Profit P(x, y)')
plt.tight_layout()
plt.savefig('plot3.png', dpi=300)
plt.show()

# Part 2: Calculating First and second order partial derivatives
P_x = diff(P, x)
P_y = diff(P, y)
P_xx = diff(P_x, x)
P_yy = diff(P_y, y)
P_xy = diff(P_x, y)

print("First-order partial derivatives:")
print(f" ∂P/∂x = {P_x}")
print(f" ∂P/∂y = {P_y}")

print("\nSecond-order partial derivatives:")
print(f" ∂²P/∂x² = {P_xx:.4f}")
print(f" ∂²P/∂y² = {P_yy:.4f}")
print(f" ∂²P/∂x∂y = {P_xy:.4f}")

# Part 3: Solving ∂P/∂x = 0 and ∂P/∂y = 0 for critical point
solutions = solve([Eq(P_x, 0), Eq(P_y, 0)], (x, y), dict=True)
opt = solutions[0]
opt_x = opt[x]
opt_y = opt[y]
opt_profit = P.subs(opt)

print("\nCritical point (possible max profit):")
print(f"x = {opt_x:.4f}, y = {opt_y:.4f}")
print(f"Maximum profit ≈ ${opt_profit.evalf():.2f}")
```

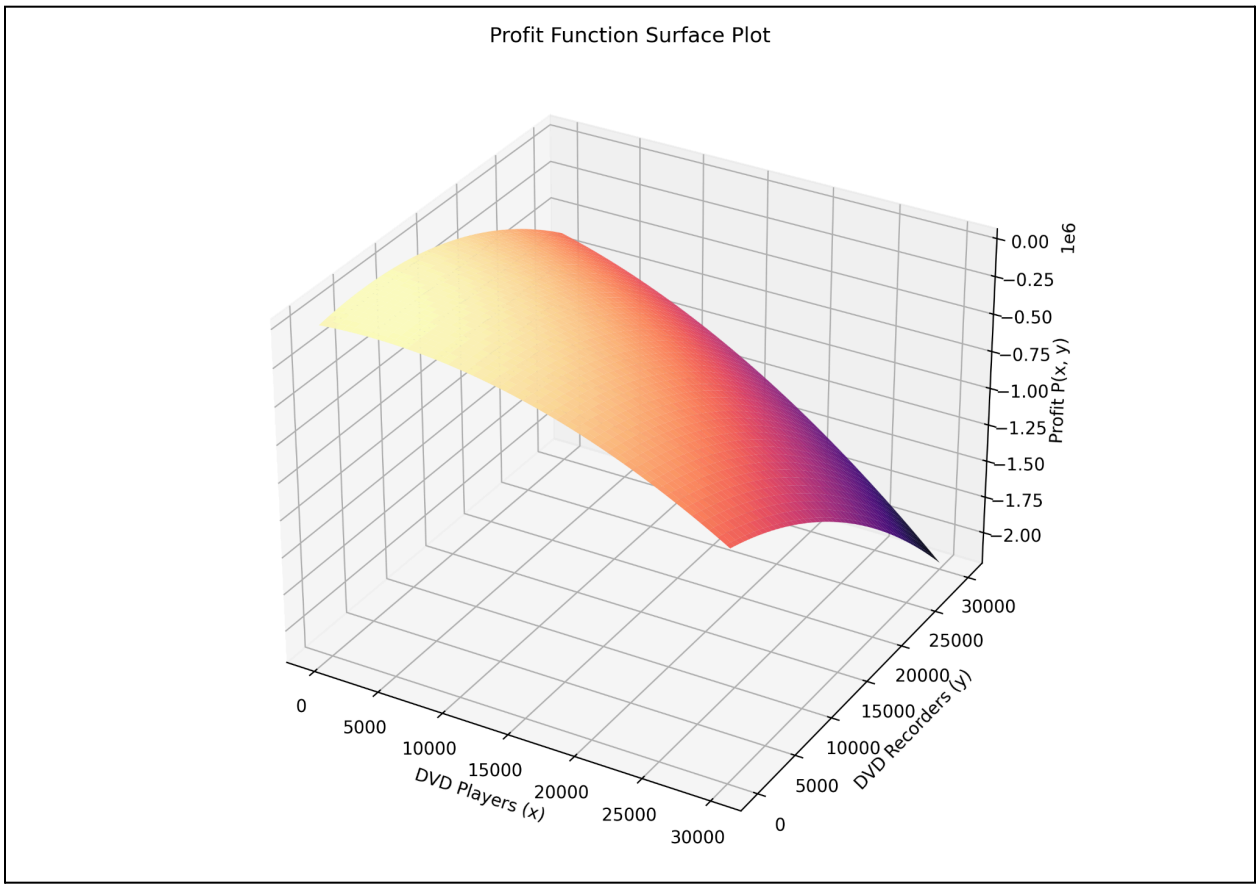
OUTPUT

First-order partial derivatives:
$\partial P/\partial x = -0.002*x - 0.001*y + 8$
$\partial P/\partial y = -0.001*x - 0.002*y + 10$

Second-order partial derivatives:
$\partial^2 P/\partial x^2 = -0.0020$
$\partial^2 P/\partial y^2 = -0.0020$
$\partial^2 P/\partial x \partial y = -0.0010$

Critical point (possible max profit):
x = 2000.0000, y = 4000.0000
Maximum profit ≈ $18000.00

GRAPH



4. Hand Calculations

Attached separately

5. Comparison Between Software and Hand Calculations Results

Question No.	Software Result	Hand Calculation Result
Question #1	X = 0, Y = 0	X = 0 , Y = 0
Question #2	X = 250.00, Y = 50.00 Max production ≈ 16718.51	X = 250, Y = 50 Max Production = 16730
Question #3	X = 2000.0000 Y = 4000.0000 Maximum profit ≈ $18000.00	X = 2000 Y = 4000 Maximum Profit = 18000

5. Difficulties Faced and How They Were Overcome

At the beginning, it became clear that basic Python functions weren't quite enough for the types of calculations required in this project. While exploring possible solutions, many libraries came up — including the **math** library — but most didn't offer the symbolic tools that were needed. Eventually, **sympy** was found to be more suitable for the task. It handled things like symbolic differentiation and equation solving much more effectively. There was a bit of a learning curve at first, but things became more manageable with time and a bit of trial and error.
Overall, a few adjustments were made along the way, and most issues were gradually resolved as the project moved forward.

There were definitely moments when laziness and procrastination showed up uninvited — because, well, who doesn't put things off sometimes? But after a little push (and maybe some snacks), we found the right angle to get back in gear and made steady progress—no need to derive any more excuses! Maam to be honest, laziness to the point that even this paragraph is chatgpt. Just for fun :)