MOUNTAINS OF THE MOON UNIVERSITY FACULTY OF SCIENCE, TECHNOLOGY AND INNOVATION DEPARTMENT OF COMPUTER SCIENCE INDIVIDUAL COURSEWORK LP BY MR.SAMUEL OCEN

TWEYONGYERE SARAH
2023/U/MMU/BCS/00100

12/February/2024

1 Solutions to Linear Programming problems

1.1 NO.1: Basic Resource Allocation

```
#NO.1:Basic Resource Allocation
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable

# Create a linear programming problem
model = LpProblem(name="Basic_Resource_Allocation", sense=LpMinimize)

# Define decision variables
x = LpVariable(name="x")
y = LpVariable(name="y")

# Define the objective function
model += 4 * x + 5 * y, "Objective_function"

# Define constraints
model += 2 * x + 3 * y  >=10, "CPU_Constraint"
model += x + 2 * y  >=5, "Memory_Constraint"
model += 3 * x +  y  >=8, "Storage_Constraint"

# Solve the linear programming problem
```

```
model.solve()

# Display the results
print("Optimal Solution:")
print(f"The Value of (x): {x.varValue}")
print(f"The Value of (y): {y.varValue}")
print(f"Minimum objective value (Z): {model.objective.value()}")
```

```
Optimal Solution:
The Value of (x): 2.0
The Value of (y): 2.0
Minimum objective value (Z): 18.0
```

1.1.1 no.1 Graph

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

#defining the x array
x = np.linspace(0, 16, 20000)

#converting constraints into inequalities
y1 = (10-2*x)/3
y2 = (5-x)/2
y3 = (8-3*x)

#Defining the feasible region
y4 = np.maximum.reduce([y1, y2, y3])
plt.fill_between(x, y4, 11, color="green", label="feasible region", alpha=0.3)

#Plotting cinstraits
plt.plot(x, y1, label="2 * x + 3 * y =10")
plt.plot(x, y2, label="x + 2 * y = 5")
plt.plot(x, y3, label="3 * x + y = 8")

#Defining limits
plt.xlim(0,10)
plt.ylim(0,10)

#Label and show
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.title("A graph showing Basic Resource Allocation shading the Feasible Region")
plt.legend()
plt.show()
```

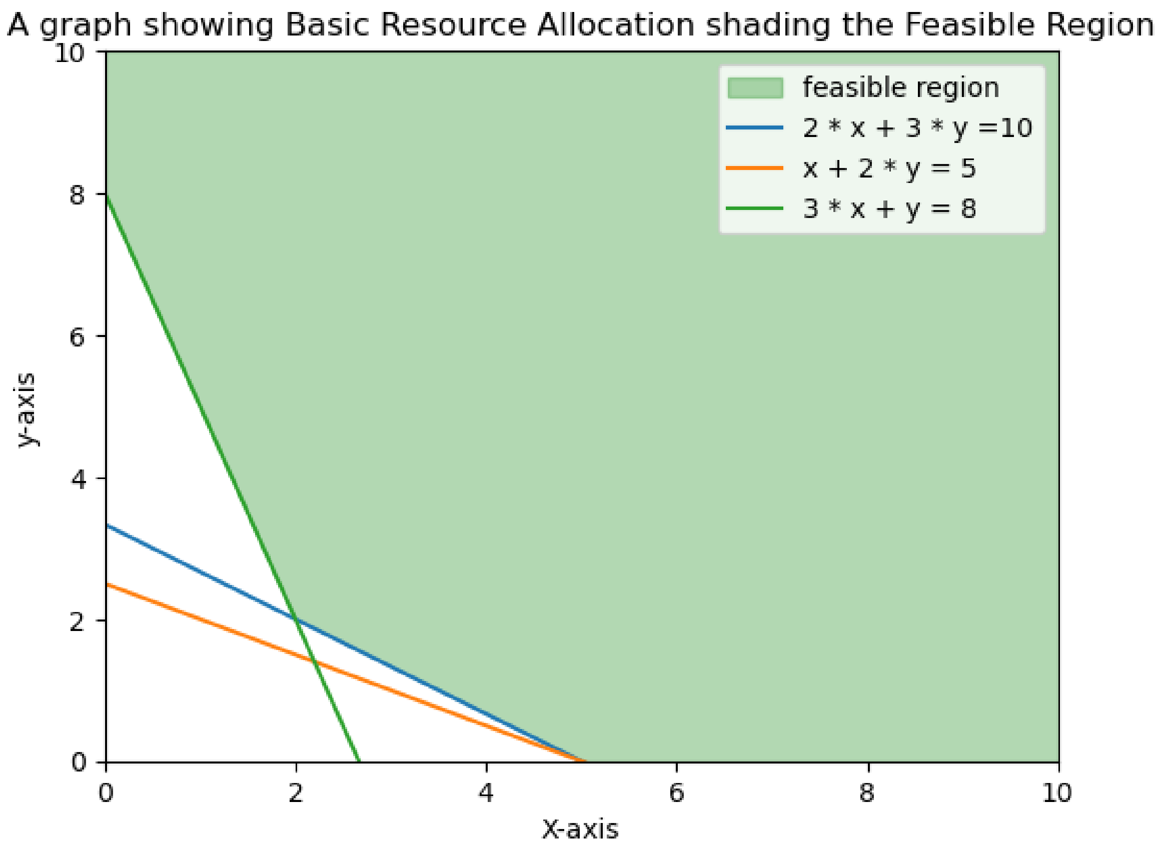

Figure 1: NO.1 Gragh

1.2 NO.2: Load Balancing

```
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable

# Create a linear programming problem
model = LpProblem(name="Overall_Time_Optimisation", sense=LpMinimize)

# Define decision variables
x = LpVariable(name="x",lowBound=0)
y = LpVariable(name="y",lowBound=0)

# Define the objective function
model += 5 * x + 4 * y, "Objective_function"
```

```
# Define constraints
model += 2 * x + 3 * y  <= 20, "Server_1_Capacity"
model += 4 * x + 2 * y  <= 15, "Server_2_Capacity"

# Solve the linear programming problem
model.solve()

# Display the results
print("Optimal Solution:")
print(f"The Value of (x): {x.varValue}")
print(f"The Value of (y): {y.varValue}")
print(f"Minimum Overall Response time (Z): {model.objective.value()}")
Optimal Solution:
The Value of (x): 0.0
The Value of (y): 0.0
Minimum Overall Response time (Z): 0.0
```

1.2.1 no.2 Gragh

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

#defining the x array
x = np.linspace(0, 16, 20000)

#converting constraints into inequalities
y1 = (20-2*x)/3
y2 = (15-4*x)/2

#Defining the feasible region
y3 = np.minimum.reduce([y1, y2])
plt.fill_between(x, y3, color="pink", label="feasible region", alpha=0.3)

#Plotting cinstraits
plt.plot(x, y1, label="2 * x + 3 * y  = 20")
plt.plot(x, y2, label="4 * x + 2 * y  = 15")

#Defining limits
plt.xlim(0,9)
plt.ylim(0,11)

#Label and show
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.title("A graph of Load Balancing shading the Feasible Region")
plt.legend()
plt.show()
```

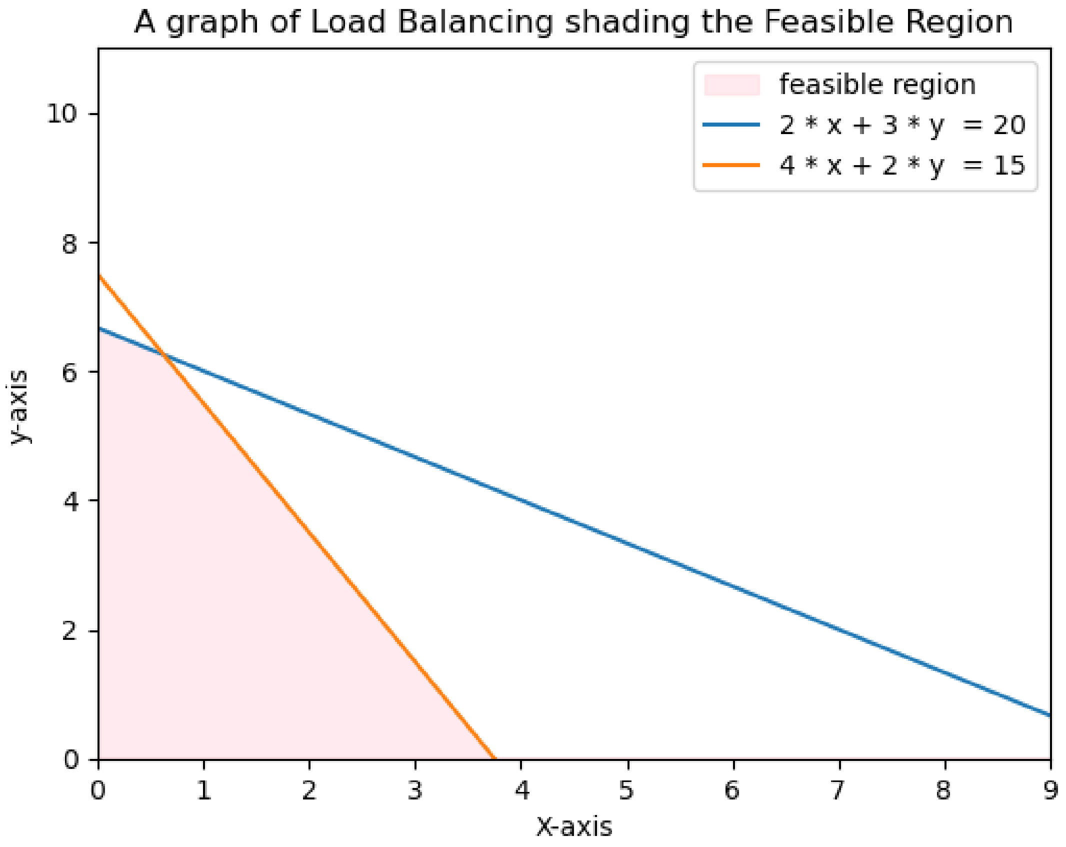

Figure 2: NO.2 Graph

1.3 Energy Efficient Resource Allocation

```
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable

# Create a linear programming problem
model = LpProblem(name="Energy_Optimization", sense=LpMinimize)

# Define decision variables
```

```
x = LpVariable(name="x", lowBound=0)
y = LpVariable(name="y", lowBound=0)

# Define the objective function
model += 3 * x + 2 * y, "Objective_function"

# Define constraints
model += 2 * x + 3 * y  >=15, "CPU_Allocation"
model += 4 * x + 2 * y  >=10, "Memory_Allocation"

# Solve the linear programming problem
model.solve()

# Display the results
print("Optimal Solution:")
print(f"The Value of (x): {x.varValue}")
print(f"The Value of (y): {y.varValue}")
print(f"Minimum Energy Cosumption value (Z): {model.objective.value()}")
Optimal Solution:
The Value of (x): 0.0
The Value of (y): 5.0
Minimum Energy Cosumption value (Z): 10.0
```

1.3.1 no.3 Graph

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

#defining the x array
x = np.linspace(0, 16, 20000)

#converting constraints into inequalities
y1 = (15-2*x)/3
y2 = (10-4*x)/2

#Defining the feasible region
y3 = np.maximum.reduce([y1, y2])
plt.fill_between(x, y3, 7, color="gray", label="feasible region", alpha=0.3)

#Plotting cinstraits
plt.plot(x, y1, label="2 * x + 3 * y =15")
plt.plot(x, y2, label="4 * x + 2 * y =10")

#Defining limits
plt.xlim(0,10)
plt.ylim(0,6)

#Label and show
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.title("A graph of Energy_Optimization shading the Feasible Region")
plt.legend()
plt.show()
```

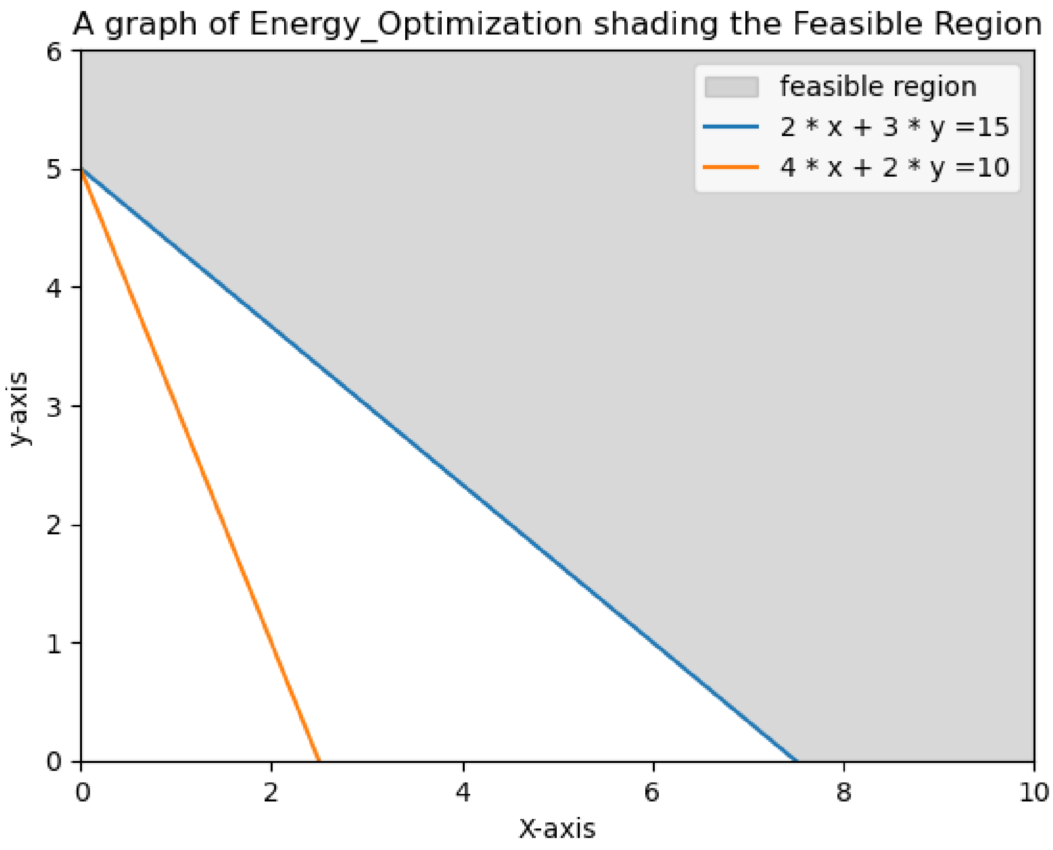

Figure 3: NO.3 Graph

1.4 NO.4 Multi$_{T}enantResourceSharing$

```
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable
```

```
# Create a linear programming problem
model = LpProblem(name="Optimal_Tenant_Resource_Sharing", sense=LpMinimize)

# Define decision variables
x = LpVariable(name="x", lowBound=0)
y = LpVariable(name="y", lowBound=0)

# Define the objective function
model += 5 * x + 4 * y, "Objective_function"

# Define constraints
model += 2 * x + 3 * y  >=12, "Tenant 1"
model += 4 * x + 2 * y  >=18, "Tenant 2"

# Solve the linear programming problem
model.solve()

# Display the results
print("Optimal Solution:")
print(f"The Value of (x): {x.varValue}")
print(f"The Value of (y): {y.varValue}")
print(f"Minimum Tenant_Resource_Sharing value (Z): {model.objective.value()}")
```

```
Optimal Solution:
The Value of (x): 3.75
The Value of (y): 1.5
Minimum Tenant_Resource_Sharing value (Z): 24.75
```

1.4.1 no.4 Graph

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

#defining the x array
x = np.linspace(0, 16, 20000)

#converting constraints into inequalities
y1 = (12-2*x)/3
y2 = (18-4*x)/2

#Defining the feasible region
y3 = np.maximum.reduce([y1, y2])
plt.fill_between(x, y3, 11, color="blue", label="feasible region", alpha=0.2)

#Plotting cinstraits
```

```
plt.plot(x, y1, label="2 * x + 3 * y =12")
plt.plot(x, y2, label="4 * x + 2 * y =18")

#Defining limits
plt.xlim(0,10)
plt.ylim(0,10)

#Label and show
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.title("A graph of Tenant_Resource_Sharing shading the Feasible Region")
plt.legend()
plt.show()
```

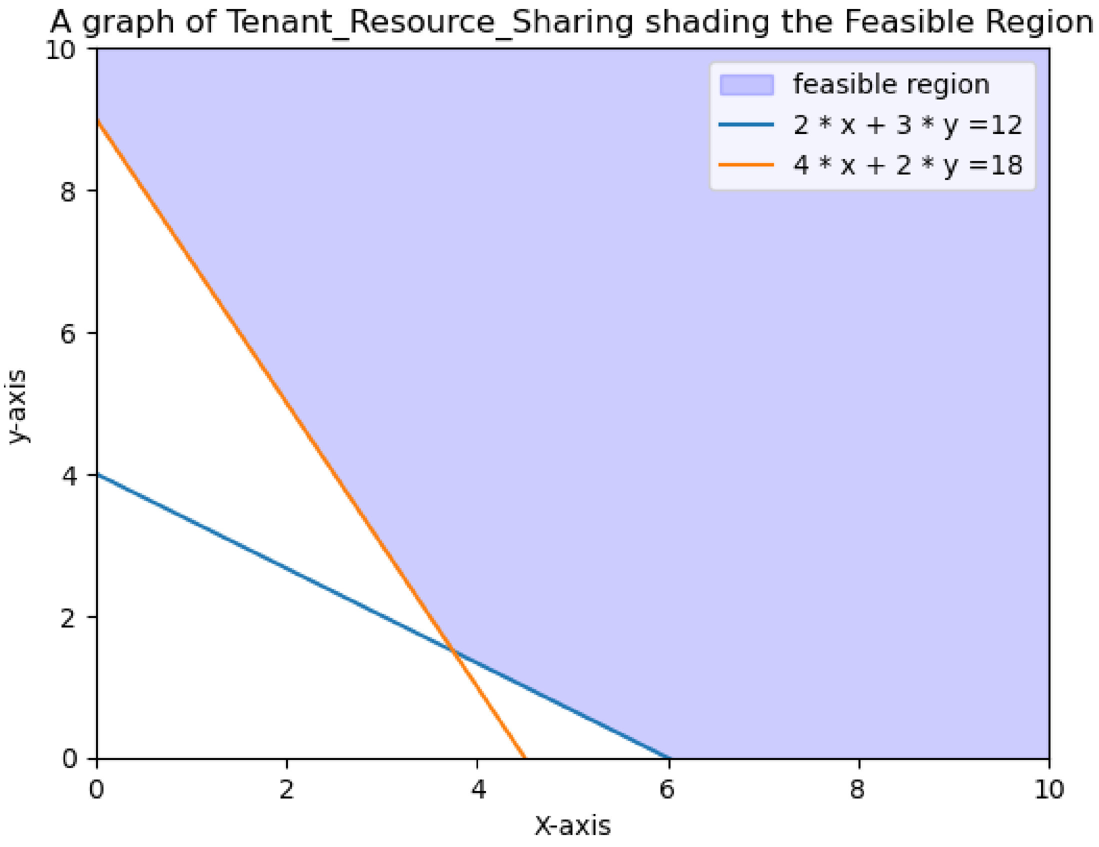

Figure 4: NO.4 Graph

1.5 NO.5 Production planning in manufacturiing

```
from pulp import*
model = LpProblem(name="product_optimisation", sense=LpMinimize)#problem
#defining variables
x1=LpVariable(name="Quantity_of_Product_1 x1", lowBound=0)
x2=LpVariable(name="Quantity_of_Product_2 x2", lowBound=0)
x3=LpVariable(name="Quantity_of_Product_3 x3", lowBound=0)
#defining the objective function
model+=5*x1 + 3*x2 + 4*x3, "objective_Function"
#defining constraints
model+=5*x1 + 3*x2 + 4*x3 <=1000, "Raw_Material"
model+=5*x1 + 3*x2 + 4*x3 <=120, "Labour_Hours"
model+=x1 >= 200, "x1_Minimum_Investment"
model+=x2 >= 300, "x2_Minimum_Investment"
model+=x3 >= 150, "x3_Minimum_Investment"
#solve
model.solve()
#print results
print("Optimal_Results")
print(f"The_value_of_x1, (x1): {x1.varValue}")
print(f"The_value_of_x2, (x2): {x2.varValue}")
print(f"The_value_of_x3, (x3): {x3.varValue}")
print(f"The_Optimal_Value (M): {model.objective.value()}")
```

```
Optimal_Results
The_value_of_x1, (x1): 200.0
The_value_of_x2, (x2): 300.0
The_value_of_x3, (x3): 0.0
The_Optimal_Value (M): 1900.0
```

1.5.1 no.5 Graph

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the inequalities
def inequality1(x1, x2, x3):
    return 2*x1 + 3*x2 + x3 <= 1000

def inequality2(x1, x2, z):
    return 4*x1 + 2*x2 + 5*x3 <= 120

def inequality3(x):
    return x1 >= 200

def inequality4(y):
    return x2 >= 3000

def inequality5(z):
    return x3 >= 150

# Create a meshgrid for plotting
x1_vals = np.linspace(200, 1000, 100)
x2_vals = np.linspace(3000, 5000, 100)
x3_vals = np.linspace(150, 300, 100)
x1, x2, x3 = np.meshgrid(x1_vals, x2_vals, x3_vals)

# Evaluate the inequalities
feasible_region = (inequality1(x1, x2, x3) & inequality2(x1, x2, x3) & inequality3(x1) & ine

# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1[feasible_region], x2[feasible_region], x3[feasible_region], color='red')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
plt.show()
```

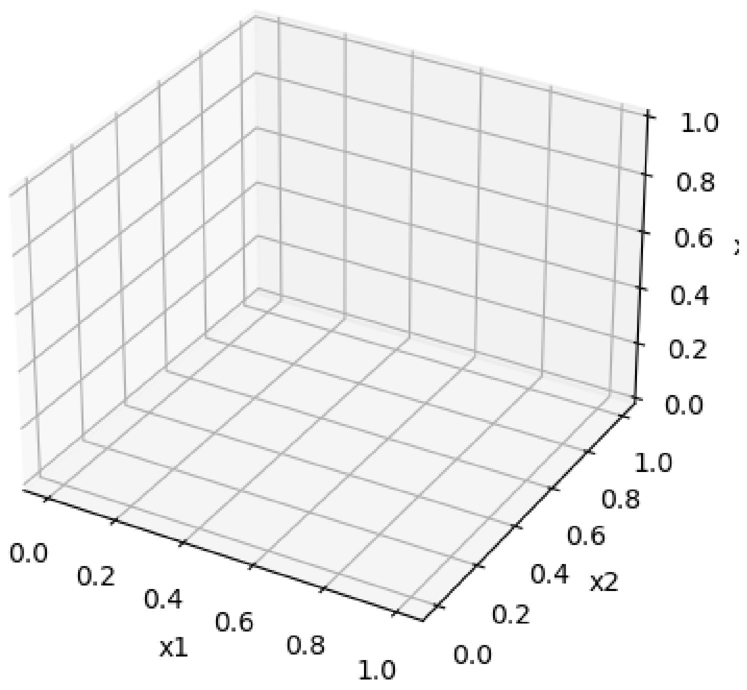

Figure 5: NO.5 Graph

1.6 NO.6 Financial Portfolio Optimisation

```
from pulp import*
# defining the linear programing problem
model = LpProblem(name="product_optimisation", sense=LpMaximize)
# defining the variable function
x1=LpVariable(name="x1", lowBound=0) # Investment in Stock A
x2=LpVariable(name="x2", lowBound=0) # Investment in Stock B
x3=LpVariable(name="x3", lowBound=0) # Investment in Stock C
# defining the objective function
model+=0.08 * x1 + 0.1 * x2 + 0.12 * x3, "objective_Function"
model+=2 * x1 + 3  * x2 + x3 <=10000, "Raw_Material"
model += x1 >= 2000 # x1_Minimum Investment
model += x2 >= 1500 # x2 Minimum Investment
model += x3 >= 1000 # x3 Minimum Investment
# solve
model.solve()
#print results
print("Optimal Results")
print(f"The value of x1, (x1): {x1.varValue}")
print(f"The value of x2, (x2): {x2.varValue}")
print(f"The value of x3, (x3): {x3.varValue}")
print(f"The Optimal Value (M): {model.objective.value()}")
```

```
Optimal Results
The value of x1, (x1): 2000.0
The value of x2, (x2): 1500.0
The value of x3, (x3): 1500.0
The Optimal Value (M): 490.0
```

1.6.1 no.6 Graph

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the inequalities
def inequality1(x1, x2, x3):
    return 2*x1 + 3*x2 + x3 <= 10000

def inequality3(x):
    return x1 >= 2000

def inequality4(y):
    return x2 >= 1500
```

```
def inequality5(z):
    return x3 >= 1000

# Create a meshgrid for plotting
x1_vals = np.linspace(500, 3000, 300)
x2_vals = np.linspace(5000, 7000, 300)
x3_vals = np.linspace(300, 600, 200)
x1, x2, x3 = np.meshgrid(x1_vals, x2_vals, x3_vals)

# Evaluate the inequalities
feasible_region = (inequality1(x1, x2, x3) & inequality2(x1, x2, x3) & inequality3(x1) & ine

# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1[feasible_region], x2[feasible_region], x3[feasible_region], color='red')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
plt.show()
```

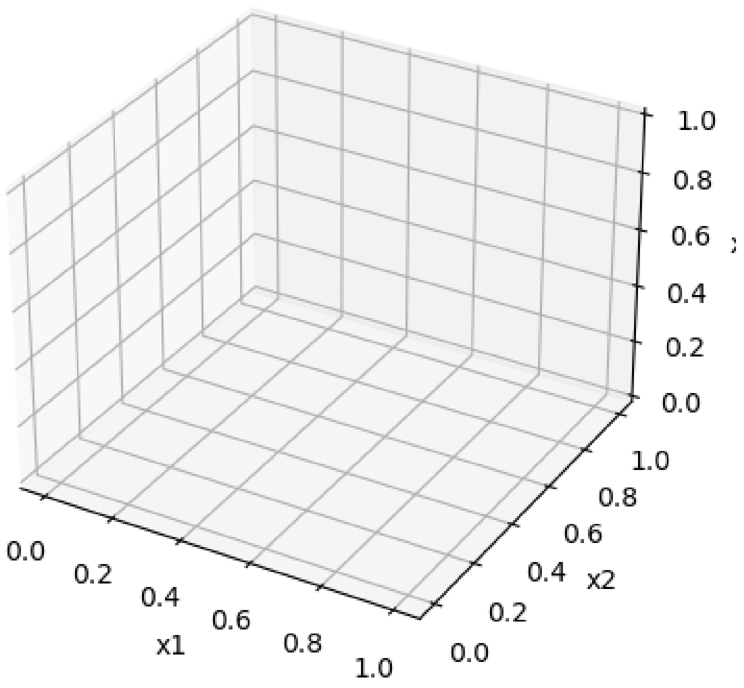

Figure 6: No.6 Graph

1.7 NO.7 Diet Optimization

```
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable

# Create a linear programming problem
model = LpProblem(name="Diet_Cost_Optimisation", sense=LpMinimize)

# Define decision variables
x = LpVariable(name="x1=services of food item 1", lowBound=0)
y = LpVariable(name="x2=services of food item 2", lowBound=0)

# Define the objective function
model += 3 * x + 2* y, "Objective_function"

# Define constraints
model += 2 * x + y  >=20, "Proteins_Constraint"
model += 3 * x + 2 * y  >=25, "Vitamin_Constraint"

# Solve the linear programming problem
model.solve()

# Display the results
print("Optimal Solution:")
print(f"The Cost for services of of item 1 (x1): {x.varValue}")
print(f"The Cost for services of of item 2 (x2): {y.varValue}")
print(f"Optimal Diet_cost value (Z): {model.objective.value()}")
```

```
Optimal Solution:
The Cost for services of of item 1 (x1): 10.0
The Cost for services of of item 2 (x2): 0.0
Optimal Diet_cost value (Z): 30.0
```

1.7.1 no.7 Graph

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

#defining the x array
x = np.linspace(0, 30, 20000)

#converting constraints into inequalities
y1 = (20-2*x)
y2 = (25-3*x)/2

#Defining the feasible region
```

```
y3 = np.maximum.reduce([y1, y2])
plt.fill_between(x, y3, 26, color="yellow", label="feasible region", alpha=0.3)

#Plotting cinstraits
plt.plot(x, y1, label="2 * x1 + x2 =20")
plt.plot(x, y2, label="3 * x1 + 2 * x2 =25")

#Defining limits
plt.xlim(0,15)
plt.ylim(0,25)

#Label and show
plt.xlabel("Services_of_food_item_1 (x1)")
plt.ylabel("Services_of_food_item_2 (x2)")
plt.title("A graph showing Diet_Optimisation shading the Feasible Region")
plt.legend()
plt.show()
```

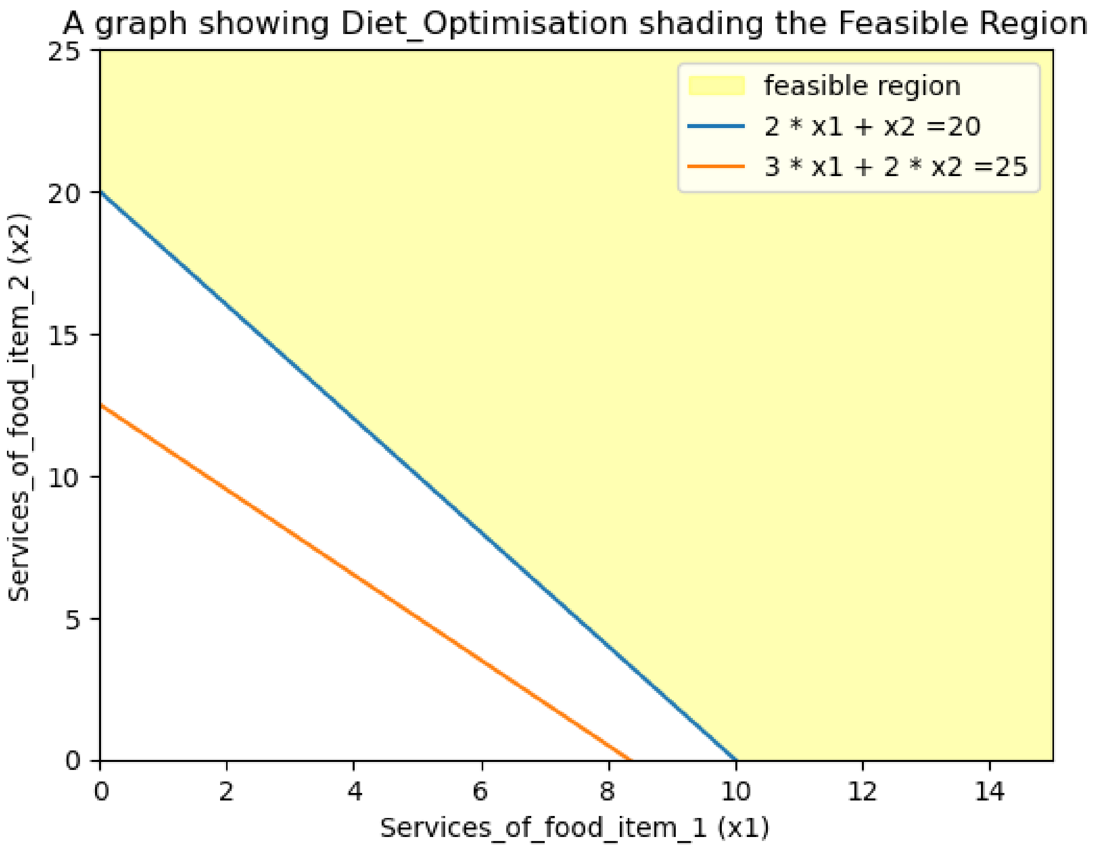

Figure 7: NO.7 Graph

1.8 NO.8 Production Planning

```
#Importing necessary libraries
from pulp import LpProblem, LpMinimize, LpVariable

# Create a linear programming problem
model = LpProblem(name="Production_Profit_Optimization", sense=LpMinimize)

# Define decision variables
x = LpVariable(name="x1=Quantity of Product A", lowBound=0)
y = LpVariable(name="x2=Quantity of Product B", lowBound=0)

# Define the objective function
model += 5 * x + 3 * y, "Objective_function"

# Define constraints
model += 2 * x + 3 * y  <=60, "Labour_Constraint"
model += 4 * x + 2 * y  <=80, "Raw_Material_Constraint"

# Solve the linear programming problem
model.solve()

# Display the results
print("Optimal Solution:")
print(f"Quantity of Product A (x1): {x.varValue}")
print(f"Quantity of Product B (x2): {y.varValue}")
print(f"Maximum Profit value (Z): {model.objective.value()}")
```

```
Optimal Solution:
Quantity of Product A (x1): 0.0
Quantity of Product B (x2): 0.0
Maximum Profit value (Z): 0.0
```

1.8.1 no.8 Graph

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

#defining the x array
x = np.linspace(0, 60, 20000)

#converting constraints into inequalities
y1 = (60-2*x)/3
y2 = (80-4*x)/2

#Defining the feasible region
```

```
y3 = np.minimum.reduce([y1, y2])
plt.fill_between(x, y3, color="maroon", label="feasible region", alpha=0.3)

#Plotting cinstraits
plt.plot(x, y1, label="2 * x + 3 * y =60")
plt.plot(x, y2, label="4 * x + 2 * y =80")

#Defining limits
plt.xlim(0,40)
plt.ylim(0,50)

#Label and show
plt.xlabel("Quantity Of Product A")
plt.ylabel("Quantity Of Product B")
plt.title("A graph of Production planning shading the Feasible Region")
plt.legend()
plt.show()
```

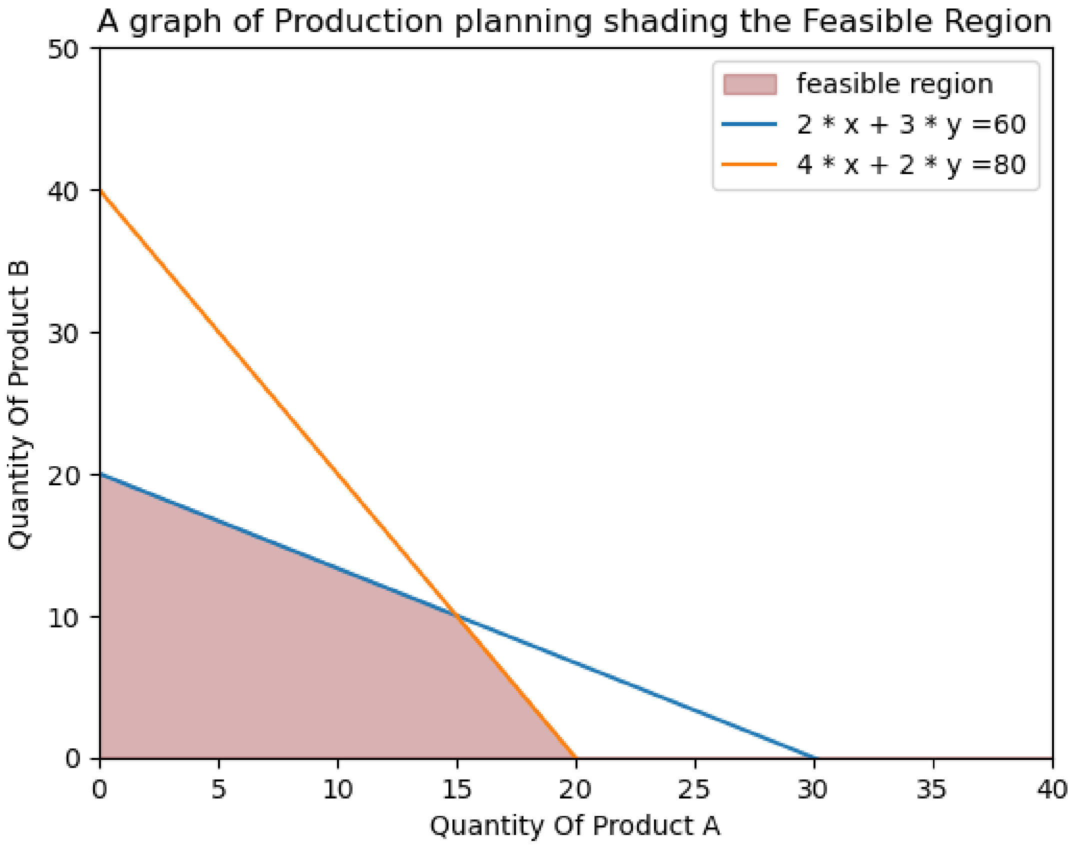

Figure 8: No.8 Graph