1 BUNOOTI AGGREY 2023-U-MMU-BCS-01666 LINEAR PROGRAMMING ASSIGNMEMENT

MOUNTAINS OF THE MOON UNIVERSITY
FACULTY OF SCIENCE , TECHNOLOGY AND INNOVATION
DEPARTMENT OF COMPUTER SCIENCE
YEAR ONE
SEMESTER TWO
COURSE NAME : LINEAR PROGRAMMING
COURSE CODE: BCS 1201
LECTURER SAMEUL OCEN
INDIVIDUAL COURSE WORK

NAME: BUNOOTI AGGREY

REG NO: 2023-U-MMU-BCS-01669

NO 1 BASIC RESOURCE ALLOCATION

```
from pulp import LpProblem, LpVariable, LpMinimize#my codes

# creating a linear programming minimization problem
model = LpProblem(name="Minimize_cost_of_Resource_Allocation", sense=LpMinimize)

# definiing the objective variables of linear programming problem
x = LpVariable(name=("x"), lowBound=0)
y = LpVariable(name=("y"), lowBound=0)

#defining the objective function of the linear programming problem
model += 4 * x + 5 * y

#defining the constraints of the linear programing problem
model += 2 * x + 3 * y >= 10 # CPU
```

```
model += x + 2 * y >= 5 # memory
model += 3 * x + y >= 8 # storage

#solve the proble
model.solve()

#display the result
print("Optimal value:")
print(f"optimal value (x): {x.varValue}")
print(f"optimal value (y): {y.varValue}")
print(f"minimum_cost (Z): {model.objective.value()}")
Optimal value:
optimal value (x): 2.0
optimal value (y): 2.0
minimum_cost (Z): 18.0

#graph for no1 resource allocation*

import numpy as np
import matplotlib.pyplot as plt
#converting constraints to inqualities
x = np.linspace(0,6,400)
#convert constraints into inqualities
y1 = (10 - 2 * x)/3
y2 = (5 - x)/2
y3 = (8 - 3 * x)

#plot constraints

plt.plot(x, y1, label = "2 * x + 3 * y >= 10") #CPU

plt.plot(x, y2, label = "4 * x + 2 * y >= 5") #Memory

plt.plot(x, y3, label = "4 * x + 2 *  >= 8") #storage

#defining the unwanted region
y4 = np.maximum.reduce([y1, y2 ,y3])

plt.fill_between(x, y4, 0, color = "gray", alpha = 0.5, label="unwanted region")

#plot the optimal solution point
optimal_x=2.0
optimal_y=2.0
plt.plot(2, 2, "ro", markersize=8, label="optimal_solution")
plt.xlim(0, 6)
plt.ylim(0, 9)
```

```
plt.grid(True)
plt.legend()
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("unwanted Region and optimal solution")
```

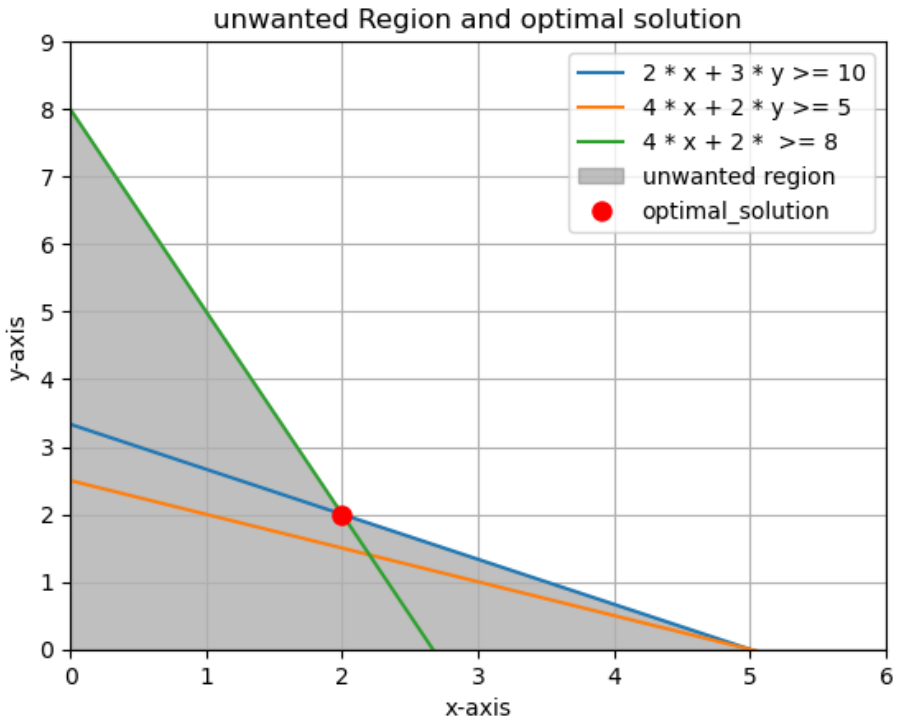

Figure 1: graph of basic resource allocation no1 unwanted region

NO2 LOAD BALANCING

```
  #importing necessary libraries

from pulp import LpProblem, LpVariable, LpMinimize

# creating a linear programming minimization problem

model = LpProblem(name="minimize_the_overall_response_time", sense=LpMinimize)

# definiing the objective variables of linear programming problem

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

#defining the objective function of the linear programming problem

model += 5 * x + 4 * y

#defining the constraints of the linear programing problem

model += 2 * x + 3 * y <= 20 #server 1 capacity
model += 4 * x + 2 * y <= 15 # server2 capcity

#solve the problem
model.solve()

#display the result
print("Optimal value:")
print(f"optimal value (x): {x.varValue}")
print(f"optimal value (y): {x.varValue}")
print(f"minimum_overall_time (Z): {model.objective.value()}")

Optimal value:
optimal value (x): 0.0
optimal value (y): 0.0
minimum_overall_time (Z): 0.0

code for graph of loading balancing

  import numpy as np
import matplotlib.pyplot as plt
#converting constraints to inqualities
x = np.linspace(0,15,400)

y1 = (20 - 2* x)/3
```

```
y2 = (15 - 4 * x)/2
#plot constraints
plt.plot(x, y1, label = "2 * x + 3 * y <= 20")
plt.plot(x, y2, label = "4 * x + 2 * y <= 15" )

#define the feasible region
y3 = np.minimum(y1, y2)

plt.fill_between(x, y3, color = "gray", alpha = 0.5, label="feasible region")

#plot the optimal solution point
optimal_x=0
optimal_y=0
plt.plot(0, 0, "ro", markersize=8, label="optimal_solution")
#define the limits
plt.xlim(0, 10)
plt.ylim(0, 10)
plt.grid(True)
plt.legend()
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Feasible Region and optimal solution")
```

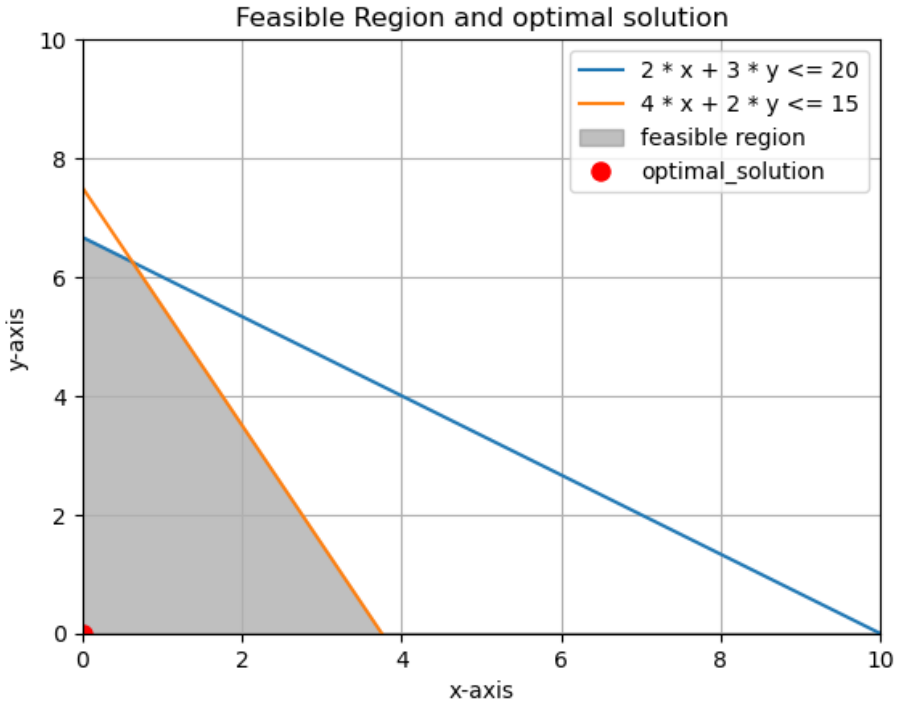

Figure 2: graph of loading Balancing no2 feasible region

NO 3 ENERGY EFFECIENCY

```
    #importing necessary libraries

from pulp import LpProblem, LpVariable, LpMinimize

# creating a linear programming minimization problem

model = LpProblem(name="minimize_the_total_energy_consumption", sense=LpMinimize)

# definiing the objective variables of linear programming problem

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

#defining the objective function of the linear programming problem

model += 3 * x + 2 * y

#defining the constraints of the linear programing problem

model += 2 * x + 3 * y >= 15 # CPU Allocation
model += 4 * x + 2 * y >= 10 # Memory Allocation

#solve the problem

model.solve()

#display the result

print("Optimal value:")
print(f"optimal value (x): {x.varValue}")
print(f"optimal value (y): {y.varValue}")
print(f"minimum_total_energy_consumption: {model.objective.value()}")

Optimal value:
optimal value (x): 0.0
optimal value (y): 5.0
minimum_total_energy_consumption: 10.0

codes for graph on energy effeciency

import numpy as np
import matplotlib.pyplot as plt

#converting constraints to inqualities
```

```
x = np.linspace(0,15,400)

y1 = (15 - 2* x)/3
y2 = (10 - 4 * x)/2

#plot constraints

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

#define the unwanted region

y3 = np.maximum(y1, y2)

plt.fill_between(x, y3, color = "gray", alpha = 0.5, label="unwanted region")

#plot the optimal solution point

optimal_x=0
optimal_y=5
plt.plot(0, 5, "ro", markersize=8, label="optimal_solution")

#define the limits

plt.xlim(0,8)
plt.ylim(0, 6)
plt.grid(True)
plt.legend()
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("unwanted Region and optimal solution")
```

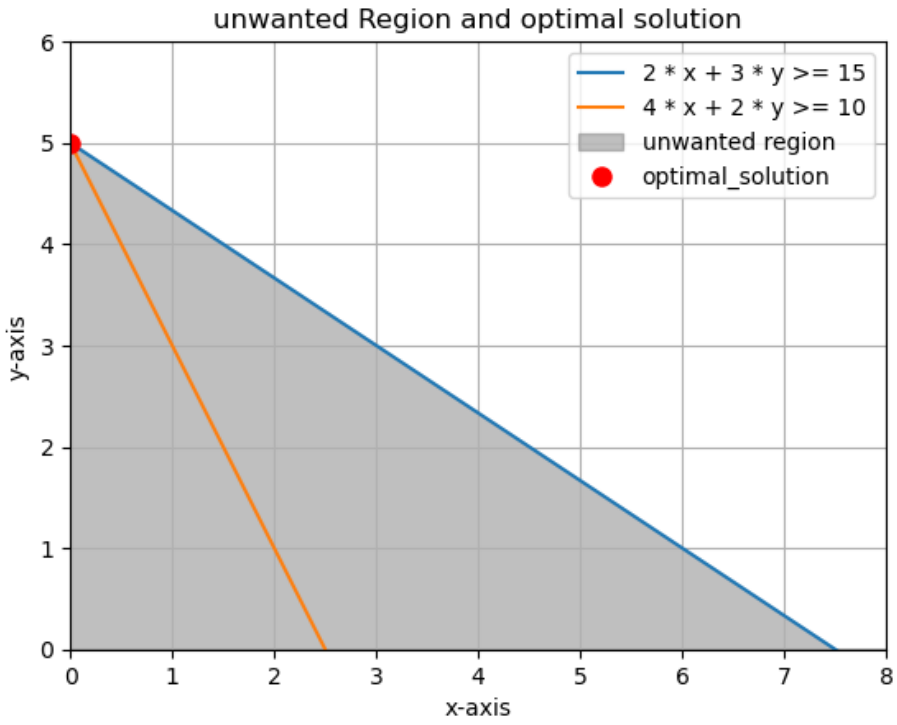

Figure 3: graph of energy effeicency no 3 unwanted region

```
    NO 4 MULTIPLE TENANTS

    #importing necessary libraries

from pulp import LpProblem, LpVariable, LpMinimize

# creating a linear programming minimization problem

model = LpProblem(name="minimize_resources_allocation", sense=LpMinimize)

# definiing the objective variables of linear programming problem

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

#defining the objective function of the linear programming problem

model += 5 * x + 4 * y

#defining the constraints of the linear programing problem

model += 2 * x + 3 * y >= 12 # Tenant 1
model += 4 * x + 2 * y >= 18 # Tenant 2

#solve the problem

model.solve()

#display the result

print("Optimal value:")
print(f"optimal value (x): {x.varValue}")
print(f"optimal value (y): {y.varValue}")
print(f"minimum_total_resource_allocation: {model.objective.value()}")

Optimal value:
optimal value (x): 3.75
optimal value (y): 1.5
minimum_total_resource_allocation: 24.75

codes of graph on no 4 multiple tenant resource
import numpy as np
import matplotlib.pyplot as plt

#converting constraints to inqualities
```

```
x = np.linspace(0,6,400)

y1 = (12 - 2* x)/3
y2 = (18 - 4 * x)/2

#plot constraints

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

#define the unwanted region

y3 = np.maximum(y1, y2)

plt.fill_between(x, y3, color = "gray", alpha = 0.5, label="unwanted region")

#plot the optimal solution point

optimal_x=3.75
optimal_y=1.5
plt.plot(3.75, 1.5, "ro", markersize=8, label="optimal_solution")

#define the limits

plt.xlim(0, 9)
plt.ylim(0, 9)
plt.grid(True)
plt.legend()
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("unwanted Region and optimal solution")
```

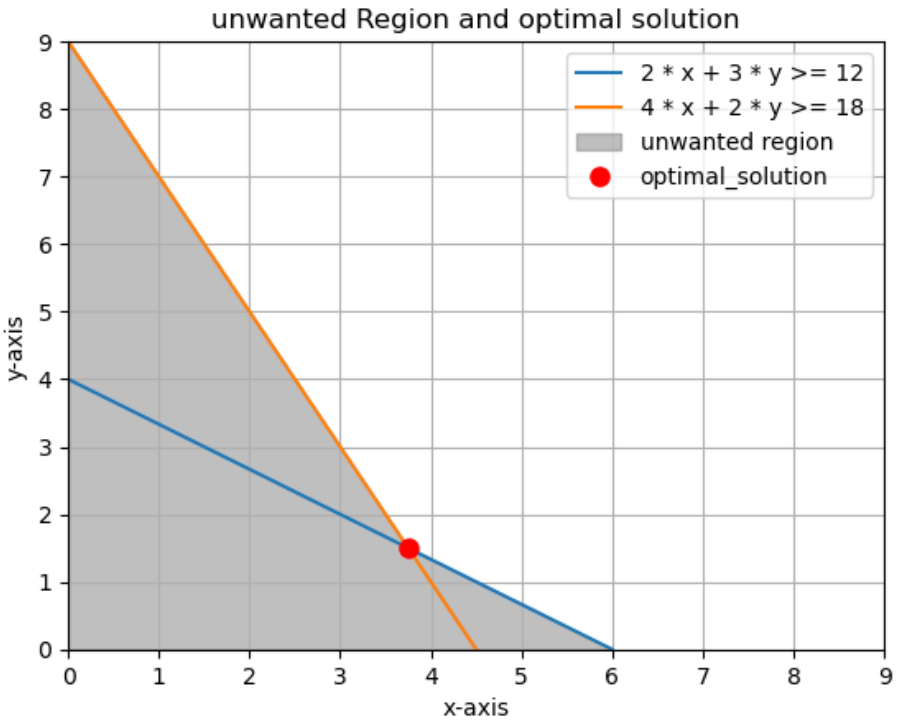

Figure 4: graph of multi tenant no 4 unwanted region

```
NO 5 PRODUCTION PLANNING MANUFACTURING

from pulp import LpProblem, LpVariable, LpMinimize

# creating a linear programming minimization problem

model = LpProblem(name="Minimizing_the_production_cost", sense=LpMinimize)

# definiing the objective variables

x1 = LpVariable(name=("x1"), lowBound=0)
x2 = LpVariable(name=("x2"), lowBound=0)
x3 = LpVariable(name=("x3"), lowBound=0)

#defining the objective function
model += 5 * x1 + 3 * x2 + 4 * x3

#defining the constraints

model += 2 * x1 +3 * x2 + x3 <= 1000    #available raw material

model += 4 * x1 + 2 * x2 + 5 * x3 <= 120 # Available hours

model += x1 >= 200 #demand constraint , minimum product constrains

model += x2 >= 300 #demand constraint , minimum product constrains

model += x3 >= 150 #demand constraint , minimum product constrains

#solve the proble

model.solve()

#display the result
print("Optimal value:")
print(f"Quantity of product (x1): {x1.varValue}")
print(f"Quantity of product (x2): {x2.varValue}")
print(f"Quantity of product (x2): {x3.varValue}")
print(f"minimum value(Z): {model.objective.value()}")

Optimal value:
Quantity of product (x1): 200.0
Quantity of product (x2): 300.0
Quantity of product (x2): 0.0
minimum value(Z): 1900.0
```

```
codes for production planning no 5

# importing necessary libraries

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

# coverting  the constraints into inequalities

def constraint1(x1):
    return (1000 - 2 * x1) / 3

def constraint2(x1):
    return (25 - 4 * x1) / 5

def constraint3(x1):
    return 300

def constraint4(x1):
    return 150

# Create arrays of x1 values for plotting

x1_values = np.linspace(0, 400, 100)

# Calculate corresponding x2 and x3 values based on the constraints

x2_values = constraint1(x1_values)
x3_values = constraint2(x1_values)

# Plot the constraints

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the feasible region surface

ax.plot_surface(x1_values.reshape(-1, 1), np.full_like(x1_values, 300), x3_values.reshape(-1

# Define the unwanted region

x1_unwanted = np.linspace(200, 40, 100)
x3_unwanted = np.full_like(x1_unwanted, 150)
x2_unwanted = constraint1(x1_unwanted)
```

```
# Plot the unwanted region surface

ax.plot_surface(x1_unwanted.reshape(-1, 1), x2_unwanted.reshape(-1, 1), x3_unwanted.reshape(

# Plot the optimal solution point

optimal_x1 = 200.0
optimal_x2 = 300.0
optimal_x3 = 0.0
ax.scatter(optimal_x1, optimal_x2, optimal_x3, color='red', s=100, label='Optimal Solution')

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
ax.set_title('Feasible Region with Optimal Solution')
plt.show()
```

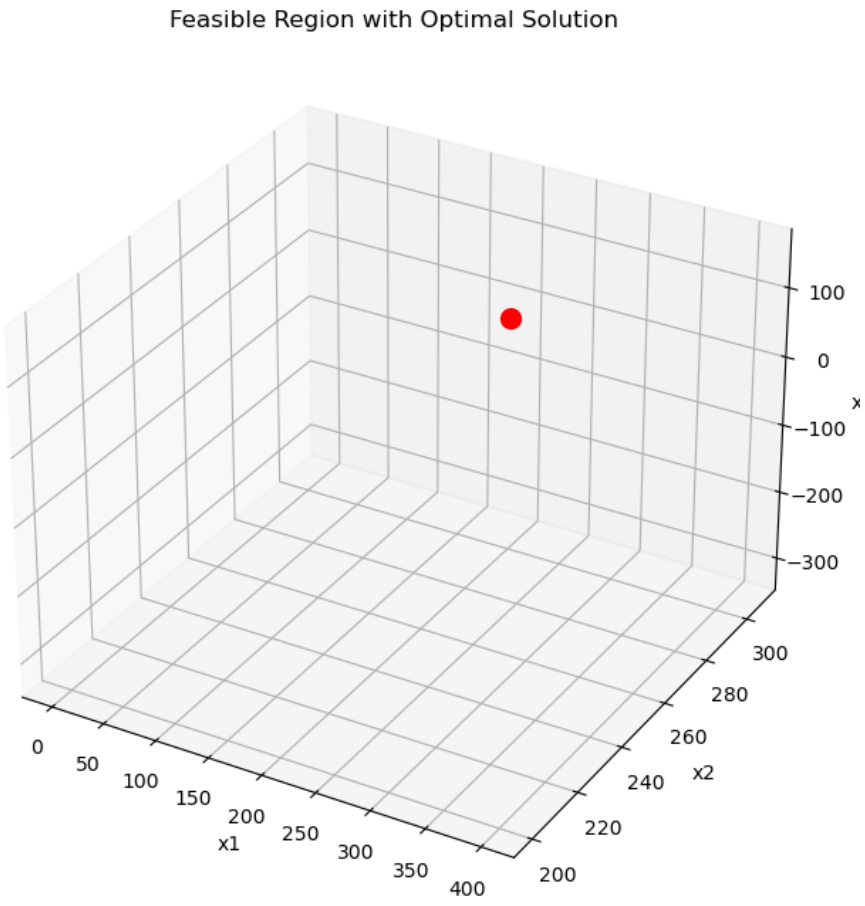

Figure 5: graph for production planning no 5 feasible region

```
    NO 6 FINANCIAL PORTFOLIO OPTIMIZATION

    #importing necessary libraries
from pulp import LpProblem, LpVariable, LpMaximize
# creating a linear programming minimization problem
model = LpProblem(name="Maximize_the_return_on_investment", sense=LpMaximize)

# definiing the objective variables

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

#defining the constraints

model += 2 * x1 + 3 * x2 + x3 <= 10000  # bubget constraint (maximum Budget for investment)
model += x1 >= 2000                      # minimum investment constraints
model += x2 >= 1500                      # minimum investment constraints
model += x3 >= 1000                      # minimum investment constraints

#solve the problem

model.solve()

#display the result

print("Optimal value:")
print(f"optimal value (x1): {x1.varValue}")
print(f"optimal value (x2): {x2.varValue}")
print(f"optimal value (x3): {x3.varValue}")
print(f"maximum_return_on_investment (Z): {model.objective.value()}")

Optimal value:
optimal value (x1): 2000.0
optimal value (x2): 1500.0
optimal value (x3): 1500.0
maximum_return_on_investment (Z): 490.0

codes for  no 6 in 3D

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

```
from mpl_toolkits.mplot3d import Axes3D

# Define the constraints
def constraint1(x1):
    return (10000 - 2 * x1) / 3

def constraint2(x1):
    return (10000 - 2 * x1) / 3

def constraint3(x1):
    return (10000 - 2 * x1) / 3

# Create arrays of x1 values for plotting
x1_values = np.linspace(0, 5000, 1000)

# Calculate corresponding x2 and x3 values based on the constraints
x2_values = constraint1(x1_values)
x3_values = constraint2(x1_values)

# Plot the constraints
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x1_values, x2_values, x3_values, color='b', alpha=0.3)

# Plot the feasible region surface
X1, X3 = np.meshgrid(x1_values, x3_values)
X2 = constraint1(x1_values)
ax.plot_surface(X1, X2, X3, color='gray', alpha=0.5, label='Feasible Region')

# Plot the optimal solution point

optimal_x1 = 2000  # Optimal value for x1
optimal_x2 = 1500  # Optimal value for x2
optimal_x3 = 1500  # Optimal value for x3
ax.scatter(optimal_x1, optimal_x2, optimal_x3, color='red', s=100, label='Optimal Solution')

# Plot the point of intersection

ax.scatter(2000, 1500, 1500, color='red', s=100, label='Point of Intersection (2000, 1500, 1

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
ax.set_title('Feasible Region with Optimal Solution')
plt.show()
```

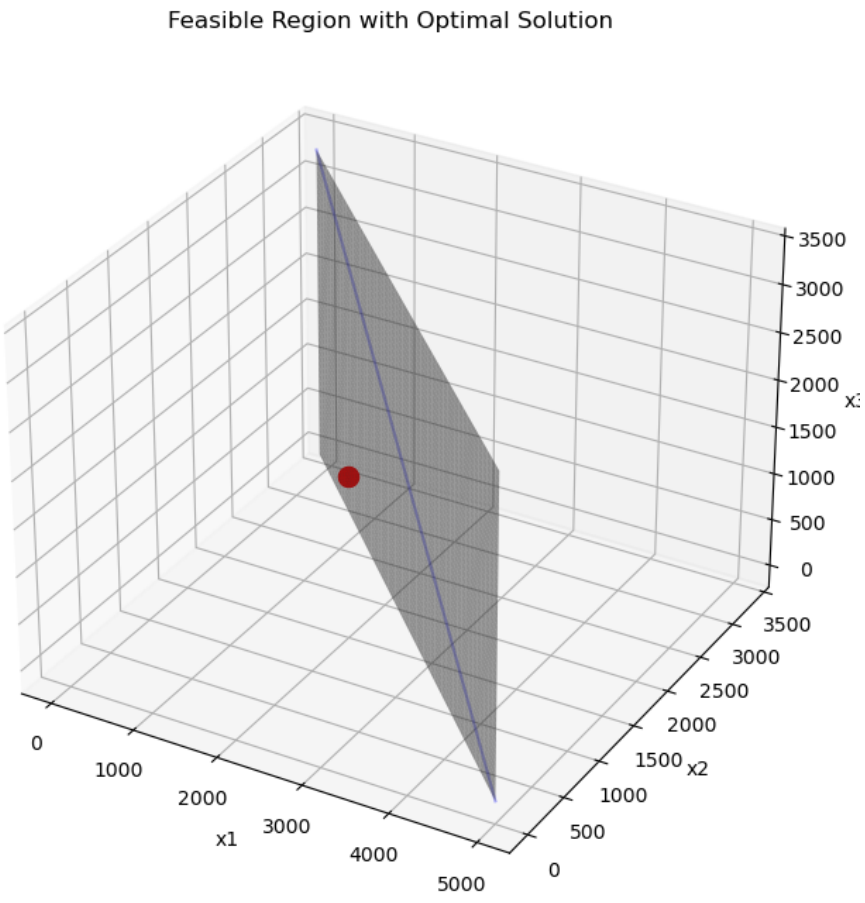

Figure 6: graph for portfolio optimization no 6 feasible region

```
codes for  graph for no 6
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the constraints
def constraint1(x1):
    return (10000 - 2 * x1) / 3

def constraint2(x1):
    return (8000 - x1) / 2

def constraint3(x1):
```

```
    return (12000 - 3 * x1) / 4

# Create arrays of x1 and x2 values for plotting
x1_values = np.linspace(0, 5000, 1000)
x2_values = np.linspace(0, 5000, 1000)

# Create meshgrid for x1 and x2
X1, X2 = np.meshgrid(x1_values, x2_values)

# Calculate corresponding x3 values based on the constraints
X3 = constraint3(X1)

# Plot the constraints
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x1_values, constraint1(x1_values), np.zeros_like(x1_values), color='b', label='Const
ax.plot(x1_values, constraint2(x1_values), np.zeros_like(x1_values), color='g', label='Const
ax.plot(x1_values, constraint3(x1_values), np.zeros_like(x1_values), color='r', label='Const

# Plot the feasible region surface (changed color to yellow)
ax.plot_surface(X1, X2, X3, color='yellow', alpha=0.5, label='Feasible Region')

# Plot the optimal solution point
optimal_x1 = 2000  # Optimal value for x1
optimal_x2 = constraint1(optimal_x1)  # Optimal value for x2
optimal_x3 = constraint3(optimal_x1)  # Optimal value for x3
ax.scatter(optimal_x1, optimal_x2, optimal_x3, color='red', s=100, label='Optimal Solution')

# Plot the point of intersection
intersection_x1 = 2000
intersection_x2 = 1500
intersection_x3 = 1500
ax.scatter(intersection_x1, intersection_x2, intersection_x3, color='orange', s=100, label=

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
ax.set_title('Feasible Region with Optimal Solution and Point of Intersection')

plt.show()
```

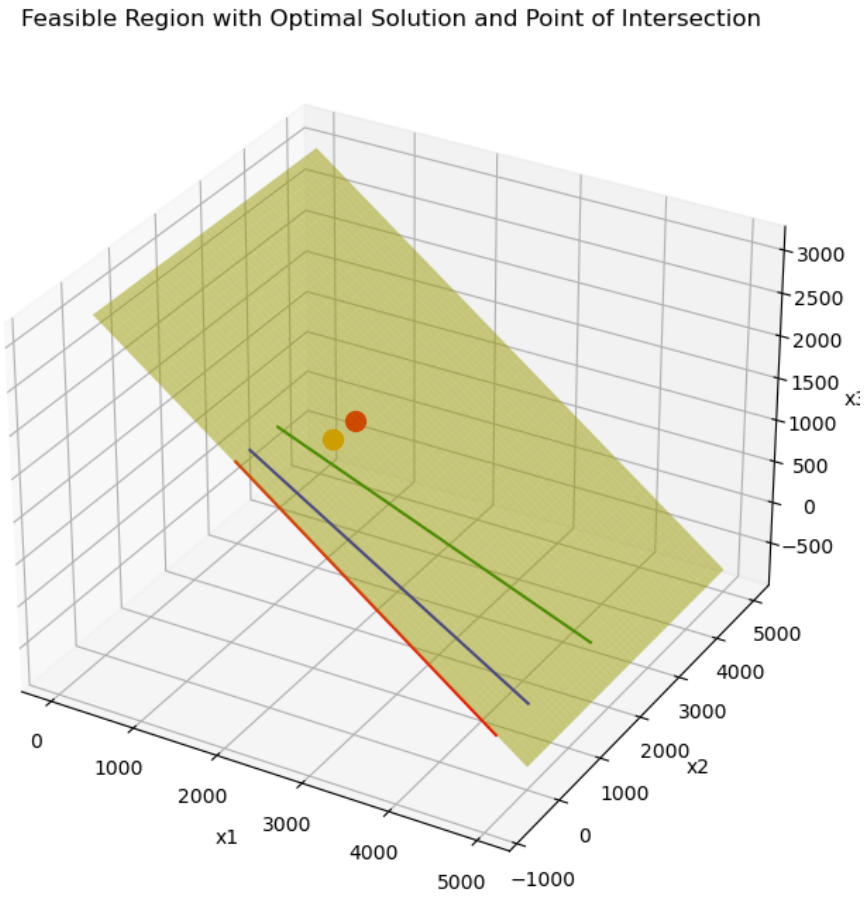

Figure 7: graph for no 6 showing feasible region 2

```
    NO 7 DIET OPTIMIZATION

    from pulp import LpProblem, LpVariable, LpMinimize#my codes

# creating a linear programming minimization problem

model = LpProblem(name="Minimizing_the_cost_of_daily", sense=LpMinimize)

# definiing the objective variableS

x1 = LpVariable(name=("x1"), lowBound=0)

x2 = LpVariable(name=("x2"), lowBound=0)

#defining the objective function

model += 3 * x1 + 2 * x2

#defining the constraints of the linear programing problem

model += 2 * x1 + x2 >= 20  #proteins in grams

model += 3 * x1 + 2 * x2 >= 25 # vitamin in units

#solve the problem

model.solve()

#display the result

print("Optimal value:")
print(f"Quantity of product (x1): {x1.varValue}")
print(f"Quantity of product (x2): {x2.varValue}")
print(f"minimum value(Z): {model.objective.value()}")

Optimal value:
Quantity of product (x1): 10.0
Quantity of product (x2): 0.0
minimum value(Z): 30.0

codes for graph of no 7 unwanted region

from pulp import LpProblem, LpVariable, LpMinimize
import numpy as np
import matplotlib.pyplot as plt
```

```
# creating a linear programming minimization problem

model = LpProblem(name="Minimizing_the_cost_of_daily", sense=LpMinimize)

# defining the objective variables

x1 = LpVariable(name="x1", lowBound=0)
x2 = LpVariable(name="x2", lowBound=0)

# defining the objective function
model += 3 * x1 + 2 * x2

# defining the constraints of the linear programming problem

model += 2 * x1 + x2 >= 20      # proteins in grams
model += 3 * x1 + 2 * x2 >= 25  # vitamin in units

# solve the problem

model.solve()

# display the result

print("Optimal value:")
print(f"Quantity of product (x1): {x1.varValue}")
print(f"Quantity of product (x2): {x2.varValue}")
print(f"Minimum value(Z): {model.objective.value()}")

# Create a grid of points

x1_values = np.linspace(0, 15, 400)
x2_values = np.linspace(0, 15, 400)

#X1, X2 = np.meshgrid(x1_values, x2_values)

# Plot the constraints

plt.figure(figsize=(8, 6))
plt.plot(x1_values, 20 - 2*x1_values, label="2x1 + x2 >= 20")
plt.plot(x1_values, (25 - 3*x1_values)/2, label="3x1 + 2x2 >= 25")

# Shade the unwanted region

plt.fill_between(x1_values, 0,(20 - 2*x1_values),where=((20-2*x1_values) >= 0),color='gray'
plt.fill_between(x1_values, 0,(25 - 3*x1_values)/2, where=((25 - 3*x1_values)/2 >= 0),color=
```

```
# Plot the optimal solution point

plt.scatter(x1.varValue, x2.varValue, color='red', label="Optimal Solution")

plt.xlabel("servings of food item 1: (eg grains):x1")
plt.ylabel("servings of food items 2: (eg vegatables):x2")
plt.title("Unwanted regiion with Optimal Solution")
plt.legend()
plt.grid(True)
plt.xlim(0, 15)
plt.ylim(0, 20)
plt.show()

Optimal value:
Quantity of product (x1): 10.0
Quantity of product (x2): 0.0
Minimum value(Z): 30.0
```

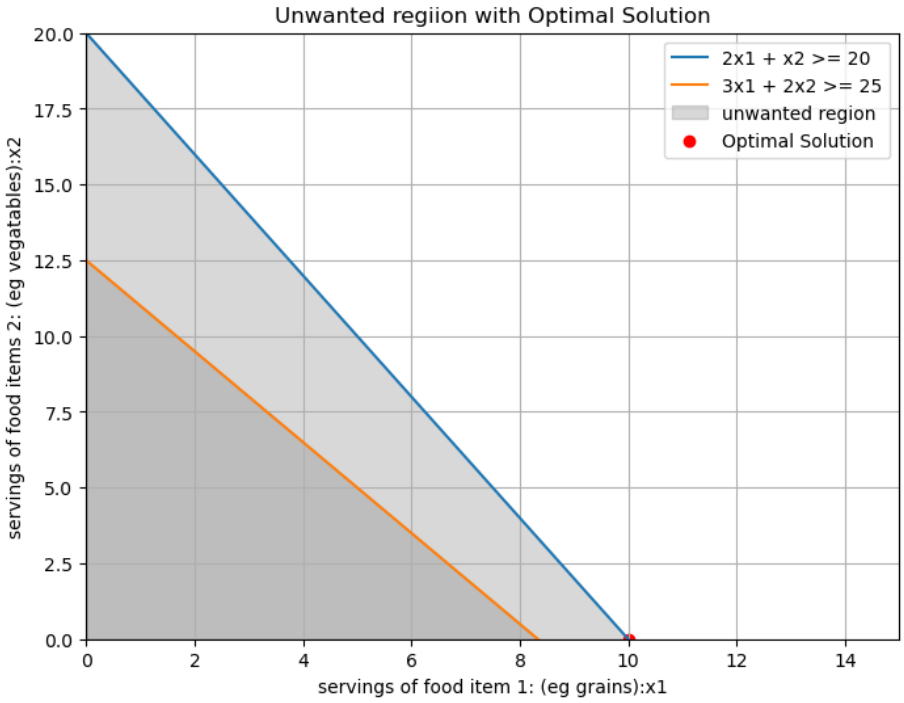

Figure 8: graph for diet optimization no 7 unwanted region

```
    NO 8 PRODUCTION PLANNING

    from pulp import LpProblem, LpVariable, LpMaximize

# creating a linear programming minimization problem

model = LpProblem(name="Maximize_the_profit_in_production_process", sense=LpMaximize)

# definiing the objective variables

x1 = LpVariable(name=("x1"), lowBound=0)
x2 = LpVariable(name=("x2"), lowBound=0)

#defining the objective function

model += 5 * x1 + 3 * x2

#defining the constraints

model += 2 * x1 + 3 * x2 <= 60 #labors in hours
model += 4 * x1 + 2 * x2 <= 80 # Raw materials in units

#solve the probleM

model.solve()

#display the result

print("Optimal value:")
print(f"Quantity of product 1 to produce (x1): {x1.varValue}")
print(f"Quantity of product 2 to produce (x2): {x2.varValue}")
print(f"maximum value(Z): {model.objective.value()}")

Optimal value:
Quantity of product 1 to produce (x1): 15.0
Quantity of product 2 to produce (x2): 10.0
maximum value(Z): 105.0

CODES FOR GRAPH NO 8


import matplotlib.pyplot as plt
import numpy as np

# defining the x array
```

```
x = np.linspace(0,50,20000)

#  convert the constraints to inequalities

y1 = (60-2*x)/3
y2 = (80-4*x)/2

# plot the the constraints

plt.plot(x,y1,label="2 * x1 + 3 * x2 <= 60")
plt.plot(x,y2,label="4 * x1 + 2 * x2 <= 80")

# shading the feasible region

y3 = np.minimum.reduce([y1,y2])
plt.fill_between(x,y3,0 ,label="feasible region", color="gray",alpha=0.5)

#optimal solution

optimal_x1 = 15.0
optimal_x2 = 10.0
plt.plot(optimal_x1, optimal_x2, "ro", markersize=8, label="optimal_solution")

# plt.plot(15,10,"ro", markersize=8, color="red", label="optimal point")

plt.xlabel("Quantity of product 1 to produce: x-axis")
plt.ylabel("Quantity of product 1 to produce: y-axis")
plt.title("feasible region and optimal solution")
plt.xlim(0,40)
plt.ylim(0,50)
plt.grid(True)
plt.legend()
plt.show()
```

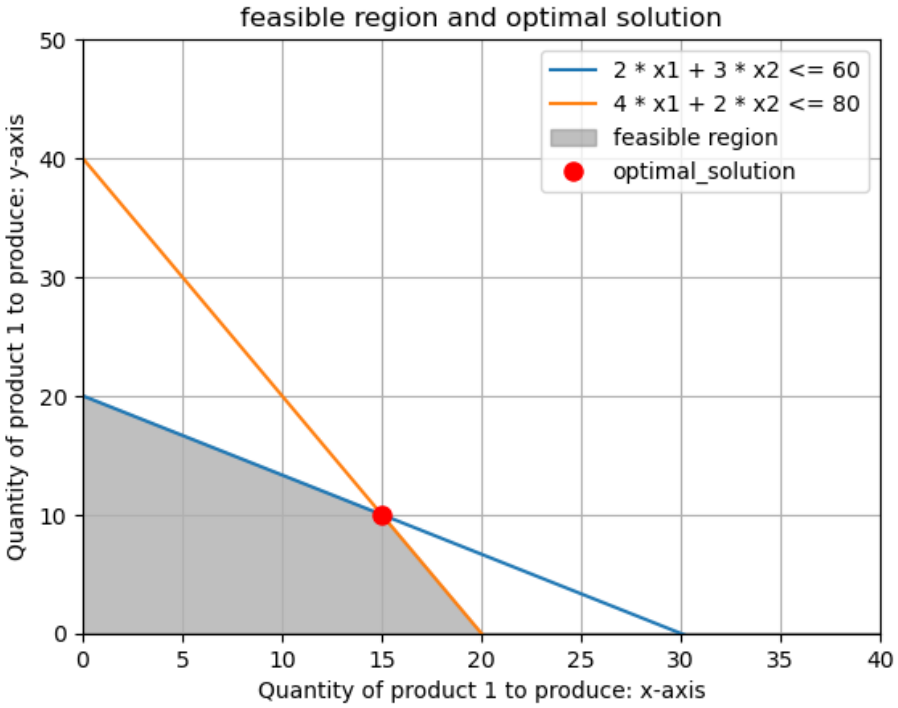

Figure 9: graph for NO 8 feasible region