KUSEMERERWA FRANK 2022/U/MMU/BCS/020

In [1]:

```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("basic_resource_allocation", sense=LpMinimize)

# Variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')

# Objective function
model += 4*x + 5*y  # Corrected the objective function

# Constraints
model += 2*x + 3*y >= 10, "CPU constraint"
model += x + 2*y >= 5, "memory constraint"
model += 3*x + y >= 8, "storage constraint"

# Solving
model.solve()

# Results
optimal_x = x.varValue
optimal_y = y.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x:", optimal_x)
print("y:", optimal_y)
print("Objective value:", optimal_value)
```

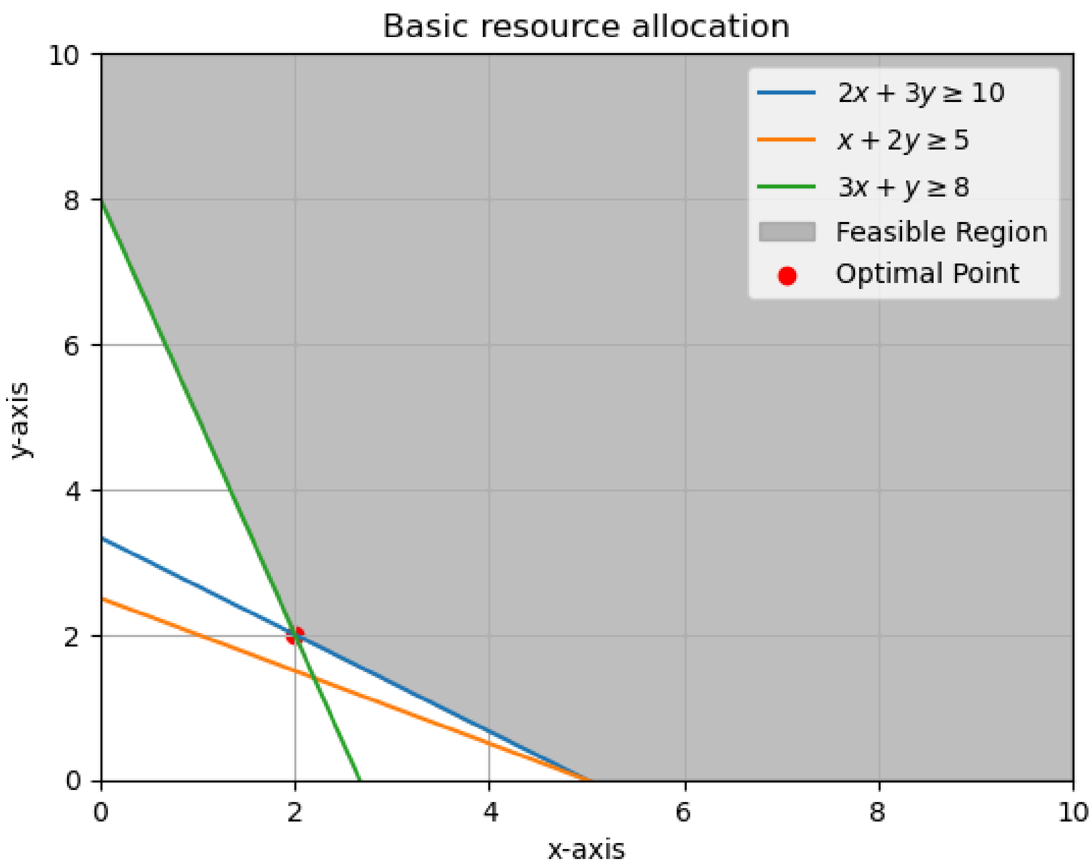
```
Optimal solution:
x: 2.0
y: 2.0
Objective value: 18.0
```

In [2]:

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

# LpProblem
model = LpProblem("basic_resource_allocation", sense=LpMinimize)

# Variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')

# Objective function
model += 4*x + 5*y

# Constraints
model += 2*x + 3*y >= 10, "CPU constraint"
model += x + 2*y >= 5, "memory constraint"
model += 3*x + y >= 8, "storage constraint"

# Solving
model.solve()

# Results
optimal_x = x.varValue
optimal_y = y.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x:", optimal_x)
print("y:", optimal_y)
print("Objective value:", optimal_value)

# Plotting the feasible region and optimal point
x_vals = np.linspace(0, 10, 100)
y1_vals = (10 - 2*x_vals) / 3
y2_vals = (5 - x_vals) / 2
y3_vals = (8 - 3*x_vals)

plt.plot(x_vals, y1_vals, label=r'$2x + 3y \geq 10$')
plt.plot(x_vals, y2_vals, label=r'$x + 2y \geq 5$')
plt.plot(x_vals, y3_vals, label=r'$3x + y \geq 8$')

plt.fill_between(x_vals, np.maximum.reduce([y1_vals, y2_vals, y3_vals, np.


plt.scatter(optimal_x, optimal_y, color='red', label='Optimal Point')

plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Basic resource allocation')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()

plt.grid(True)
```

```
plt.show()
```

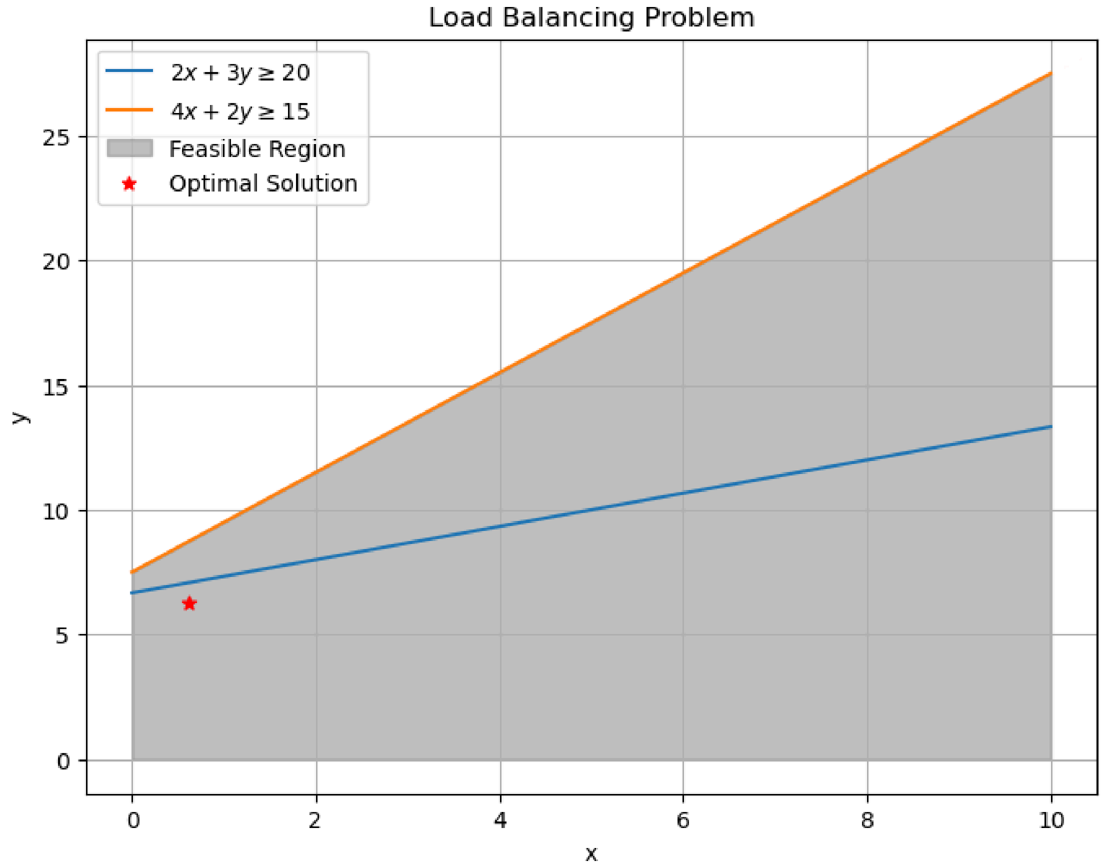
```
Optimal solution:
x: 2.0
y: 2.0
Objective value: 18.0
```



In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

#define the objective function coeffcoefficients
c = [5, 4]

#coefficients of the inequality constraints : left hand side
A = np.array([
    [-2, -3],
    [-4, -2],
])

#right hand side
b = [-20, -15]

#Solving the problem
result = linprog(c, A_ub=A, b_ub=b)

#display the results
print("Optimal values: ")
print("x =", result.x[0])
print("y =", result.x[1])
print("Optimal Objective Function value (z) = ", result.fun)


#plotting
x_values = np.linspace(0, 10, 100)
y1_values = (20 + 2*x_values) / 3
y2_values = (15 + 4*x_values) / 2

plt.figure(figsize=(8, 6))
plt.plot(x_values, y1_values, label=r'$2x + 3y \geq 20$')
plt.plot(x_values, y2_values, label=r'$4x + 2y \geq 15$')

plt.fill_between(x_values, np.maximum(y1_values, y2_values), color="gray",
plt.scatter(result.x[0], result.x[1], color='red', marker='*', label="Opti

plt.xlabel('x')
plt.ylabel('y')
plt.title("Load Balancing Problem")
plt.legend()
plt.grid(True)
plt.show()
```

```
Optimal values:
x = 0.6250000000000003
y = 6.25
Optimal Objective Function value (z) =  28.125
```


Load Balancing Problem
$2x + 3y \geq 20$
$4x + 2y \geq 15$
Feasible Region
Optimal Solution
y
x
0
5
10
15
20
25
0
2
4
6
8
10

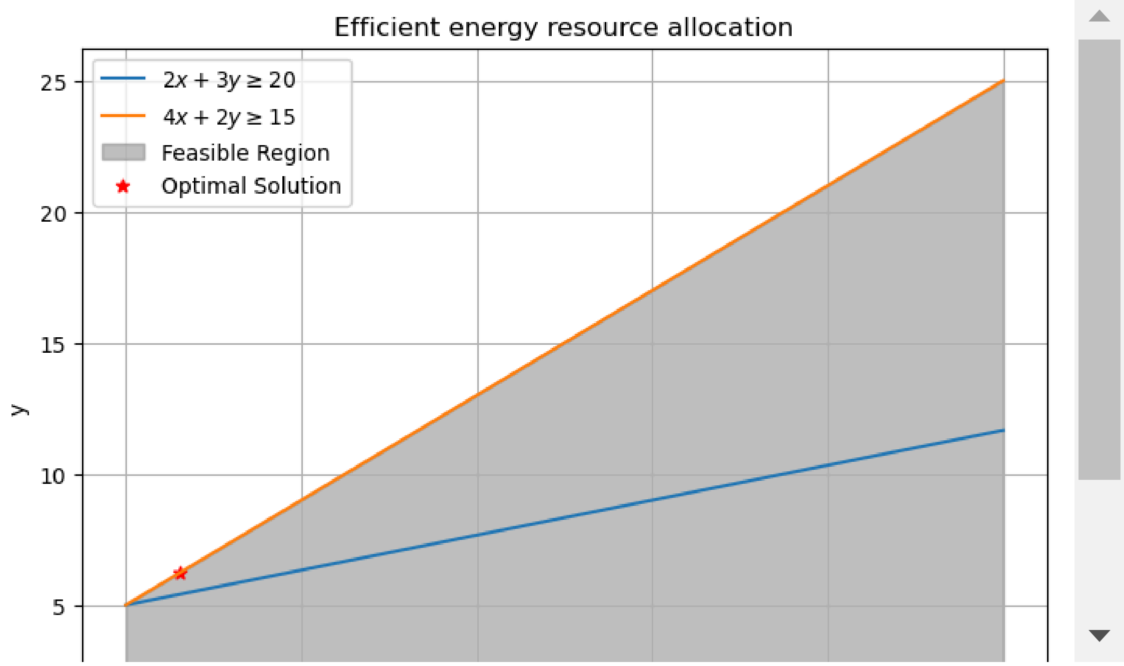
In [1]:

```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("Energy-Effecient_resource_Allocation", sense=LpMinimize

# Variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')

# Objective function
model += 3*x + 2*y  # Corrected the objective function

# Constraints
model += 2*x + 3*y >= 15, "CPU allocation constraint"
model += 4*x + 2*y >= 10, "Memory allocation constraint"

# Solving
model.solve()

# Results
optimal_x = x.varValue
optimal_y = y.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x:", optimal_x)
print("y:", optimal_y)
print("Objective value:", optimal_value)
```

```
Optimal solution:
x: 0.0
y: 5.0
Objective value: 10.0
```

In [13]:

```
#plotting
x_values = np.linspace(0, 10, 100)
y1_values = (15 + 2*x_values) / 3
y2_values = (10 + 4*x_values) / 2

plt.figure(figsize=(8, 6))
plt.plot(x_values, y1_values, label=r'$2x + 3y \geq 20$')
plt.plot(x_values, y2_values, label=r'$4x + 2y \geq 15$')

plt.fill_between(x_values, np.maximum(y1_values, y2_values), color="gray",
plt.scatter(result.x[0], result.x[1], color='red', marker='*', label="Opti

plt.xlabel('x')
plt.ylabel('y')
plt.title("Efficient energy resource allocation")
plt.legend()
plt.grid(True)
plt.show()
```



In [6]:

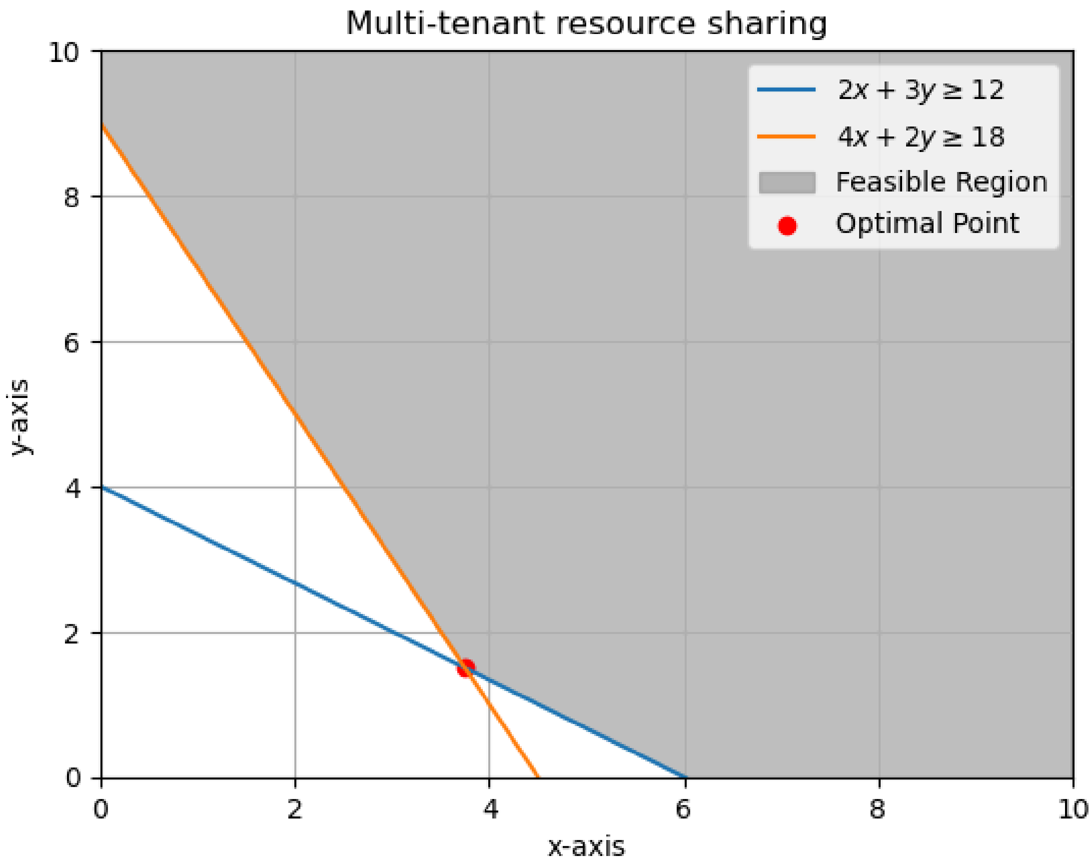
```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("Multi-Tenant_resource_sharing", sense=LpMinimize)

# Variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')

# Objective function
model += 5*x + 4*y  # Corrected the objective function

# Constraints
model += 2*x + 3*y >= 12, "Tenant1 constraint"
model += 4*x + 2*y >= 18, "Tenant2 constraint"

# Solving
model.solve()

# Results
optimal_x = x.varValue
optimal_y = y.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x:", optimal_x)
print("y:", optimal_y)
print("Objective value:", optimal_value)
```

```
Optimal solution:
x: 3.75
y: 1.5
Objective value: 24.75
```

In [11]:

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

# LpProblem
model = LpProblem("Multi-Tenant_resource_sharing", sense=LpMinimize)

# Variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')

# Objective function
model += 5*x + 4*y

# Constraints
model += 2*x + 3*y >= 12, "Tenant1 constraint"
model += 4*x + 2*y >= 18, "Tenant2 constraint"

# Solving
model.solve()

# Results
optimal_x = x.varValue
optimal_y = y.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x:", optimal_x)
print("y:", optimal_y)
print("Objective value:", optimal_value)

# Plotting the feasible region and optimal point
x_vals = np.linspace(0, 10, 100)
y1_vals = (12 - 2*x_vals) / 3
y2_vals = (18 - 4*x_vals) / 2

plt.plot(x_vals, y1_vals, label=r'$2x + 3y \geq 12$')
plt.plot(x_vals, y2_vals, label=r'$4x + 2y \geq 18$')

plt.fill_between(x_vals, np.maximum(y1_vals, y2_vals),11, color='gray', al

plt.scatter(optimal_x, optimal_y, color='red', label='Optimal Point')

plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Multi-tenant resource sharing')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()

plt.grid(True)
plt.show()
```

```
Optimal solution:
x: 3.75
y: 1.5
Objective value: 24.75
```



In [16]:

```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("production_planning", sense=LpMinimize)

# Variables
x1 = LpVariable("x1", lowBound=0, cat='Continuous')
x2 = LpVariable("x2", lowBound=0, cat='Continuous')

# Objective function
model += 5*x1 + 3*x2  # Corrected the objective function

# Constraints
model += 2*x1 + 3*x2 <= 60, "labour constraint"
model += 4*x1 + 2*x2 <= 80, "raw materials constraint"

# Solving
model.solve()

# Results
optimal_x1 = x1.varValue
optimal_x2 = x2.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x1:", optimal_x1)
print("x2:", optimal_x2)
print("Objective value:", optimal_value)
```

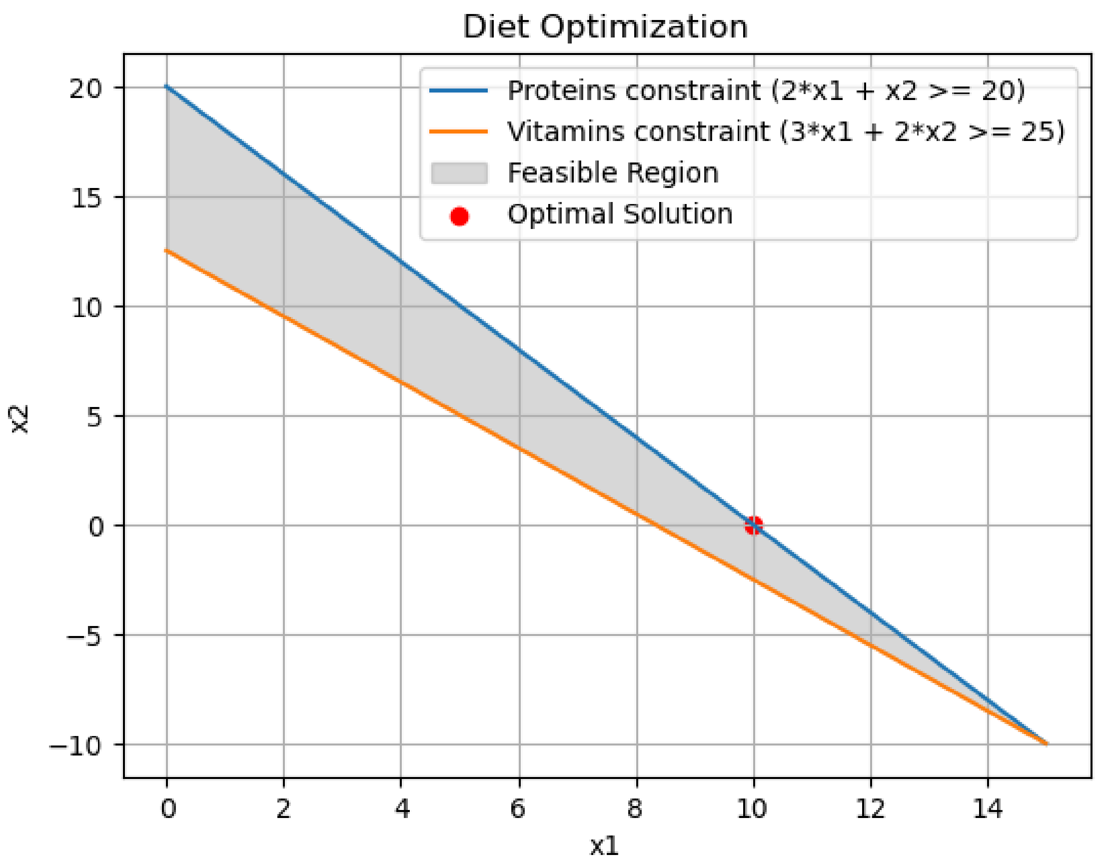
```
Optimal solution:
x1: 0.0
x2: 0.0
Objective value: 0.0
```

In [1]:

```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("Diet_Optimisation", sense=LpMinimize)

# Variables
x1 = LpVariable("x1", lowBound=0, cat='Continuous')
x2 = LpVariable("x2", lowBound=0, cat='Continuous')

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

# Constraints
model += 2*x1 + x2 >= 20, "Proteins constraint"
model += 3*x1 + 2*x2 >= 25, "Vitamins constraint"


# Solving
model.solve()

# Results
optimal_x1 = x1.varValue
optimal_x2= x2.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x1:", optimal_x1)
print("x2:", optimal_x2)
print("Objective value:", optimal_value)
import matplotlib.pyplot as plt
import numpy as np

# Define the constraints
x1_values = np.linspace(0, 15, 100)  # Adjust the range based on your prob
constraint1 = (20 - 2*x1_values)  # 2*x1 + x2 >= 20
constraint2 = (25 - 3*x1_values) / 2  # 3*x1 + 2*x2 >= 25

# Plot the constraints
plt.plot(x1_values, constraint1, label="Proteins constraint (2*x1 + x2 >= 
plt.plot(x1_values, constraint2, label="Vitamins constraint (3*x1 + 2*x2 >

# Highlight the feasible region
plt.fill_between(x1_values, constraint1, constraint2, where=(constraint1 >

# Highlight the optimal solution
plt.scatter(optimal_x1, optimal_x2, color='red', label='Optimal Solution')

# Set labels and title
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Diet Optimization')
plt.legend()

# Show the plot
plt.grid(True)
```

```
plt.show()
```

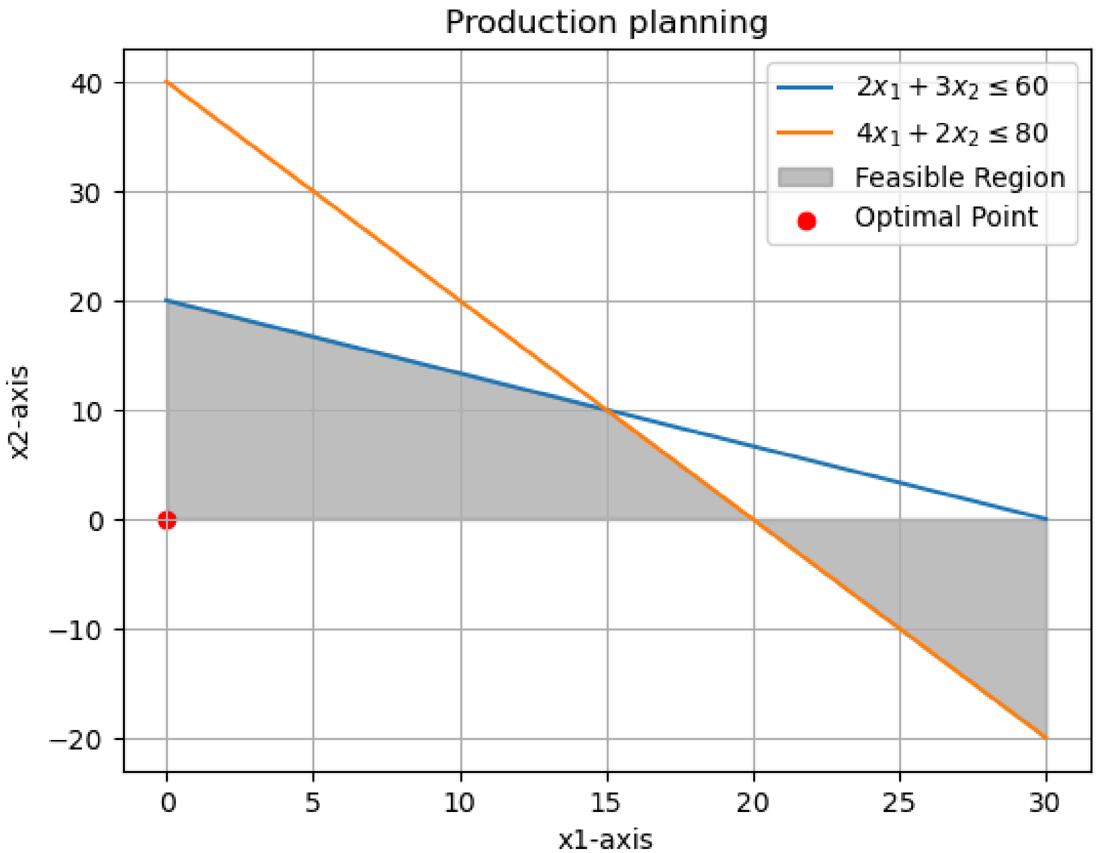
```
Optimal solution:
x1: 10.0
x2: 0.0
Objective value: 30.0
```



In [24]:

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

# LpProblem
model = LpProblem("production_planning", sense=LpMinimize)

# Variables
x1 = LpVariable("x1", lowBound=0, cat='Continuous')
x2 = LpVariable("x2", lowBound=0, cat='Continuous')

# Objective function
model += 5*x1 + 3*x2

# Constraints
model += 2*x1 + 3*x2 <= 60, "labour constraint"
model += 4*x1 + 2*x2 <= 80, "raw materials constraint"

# Solving
model.solve()

# Results
optimal_x1 = x1.varValue
optimal_x2 = x2.varValue
optimal_value = model.objective.value()

print("Optimal solution:")
print("x1:", optimal_x1)
print("x2:", optimal_x2)
print("Objective value:", optimal_value)

# Plotting the feasible region and optimal point
x1_vals = np.linspace(0, 30, 100)
x2_vals1 = (60 - 2*x1_vals) / 3  # From the first constraint
x2_vals2 = (80 - 4*x1_vals) / 2  # From the second constraint

plt.plot(x1_vals, x2_vals1, label=r'$2x_1 + 3x_2 \leq 60$')
plt.plot(x1_vals, x2_vals2, label=r'$4x_1 + 2x_2 \leq 80$')

plt.fill_between(x1_vals, 0, np.minimum(x2_vals1, x2_vals2), color='gray',

plt.scatter(optimal_x1, optimal_x2, color='red', label='Optimal Point')

plt.xlabel('x1-axis')
plt.ylabel('x2-axis')
plt.title('Production planning')
plt.legend()
plt.grid(True)
plt.show()
```

```
Optimal solution:
x1: 0.0
x2: 0.0
Objective value: 0.0
```


Production planning
$2x_1 + 3x_2 \leq 60$
$4x_1 + 2x_2 \leq 80$
Feasible Region
Optimal Point
x2-axis
x1-axis
40
30
20
10
0
−10
−20
0
5
10
15
20
25
30

In [4]:

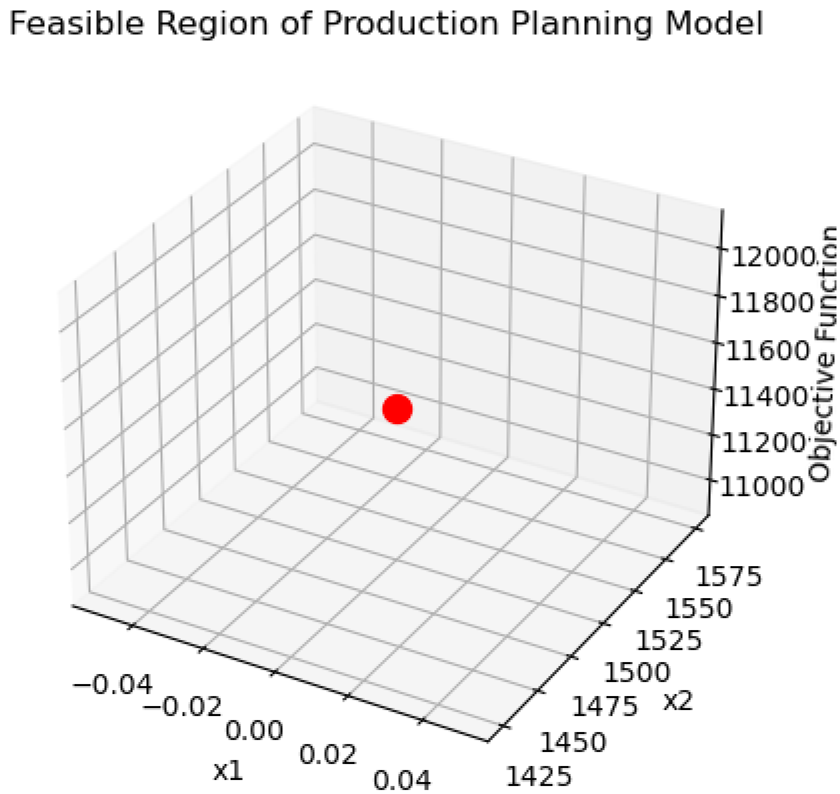
```
# Plotting the feasible region and optimal point
x1_vals = np.linspace(0, 30, 100)
x2_vals1 = (60 - 2*x1_vals) / 3  # From the first constraint
x2_vals2 = (80 - 4*x1_vals) / 2  # From the second constraint

plt.plot(x1_vals, x2_vals1, label=r'$2x_1 + 3x_2 \leq 60$')
plt.plot(x1_vals, x2_vals2, label=r'$4x_1 + 2x_2 \leq 80$')

plt.fill_between(x1_vals, 0, np.minimum(x2_vals1, x2_vals2), color='gray',

plt.scatter(optimal_x1, optimal_x2, color='red', label='Optimal Point')

plt.xlabel('x1-axis')
plt.ylabel('x2-axis')
plt.title('Production planning')
plt.legend()
plt.grid(True)
plt.show()# Plotting the feasible region and optimal point
x1_vals = np.linspace(0, 30, 100)
x2_vals1 = (60 - 2*x1_vals) / 3  # From the first constraint
x2_vals2 = (80 - 4*x1_vals) / 2  # From the second constraint

plt.plot(x1_vals, x2_vals1, label=r'$2x_1 + 3x_2 \leq 60$')
plt.plot(x1_vals, x2_vals2, label=r'$4x_1 + 2x_2 \leq 80$')

plt.fill_between(x1_vals, 0, np.minimum(x2_vals1, x2_vals2), color='gray',

plt.scatter(optimal_x1, optimal_x2, color='red', label='Optimal Point')

plt.xlabel('x1-axis')
plt.ylabel('x2-axis')
plt.title('Production planning')
plt.legend()
plt.grid(True)
plt.show()

# Plotting the feasible region and optimal point
x1_vals = np.linspace(0, 30, 100)
x2_vals1 = (60 - 2*x1_vals) / 3  # From the first constraint
x2_vals2 = (80 - 4*x1_vals) / 2  # From the second constraint

plt.plot(x1_vals, x2_vals1, label=r'$2x_1 + 3x_2 \leq 60$')
plt.plot(x1_vals, x2_vals2, label=r'$4x_1 + 2x_2 \leq 80$')

plt.fill_between(x1_vals, 0, np.minimum(x2_vals1, x2_vals2), color='gray',

plt.scatter(optimal_x1, optimal_x2, color='red', label='Optimal Point')

plt.xlabel('x1-axis')
plt.ylabel('x2-axis')
plt.title('Production planning')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

```
Optimal solution:
x1: 200.0
x2: 300.0
x3: 0.0
```

In [17]:

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

# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Optimal solution values
optimal_x1 = x1.varValue
optimal_x2 = x2.varValue
optimal_x3 = x3.varValue

# Define a grid
x1_vals = np.linspace(0, optimal_x1, 50)
x2_vals = np.linspace(0, optimal_x2, 50)
x3_vals = np.linspace(0, optimal_x3, 50)

# Create a meshgrid
X1, X2, X3 = np.meshgrid(x1_vals, x2_vals, x3_vals)

# Calculate the objective function values for each combination of variable
Z = 5*X1 + 5*X2 + 4*X3

# Constraints
constraint1 = 2*X1 + 3*X2 + X3
constraint2 = 4*X1 + 2*X2 + 5*X3

# Create a mask for the feasible region
feasible_region = np.logical_and(constraint1 <= 1000, constraint2 <= 120)

# Scatter plot the optimal solution point
ax.scatter(optimal_x1, optimal_x2, 5*optimal_x1 + 5*optimal_x2 + 4*optimal

# Set labels and title
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Objective Function')
ax.set_title('Feasible Region of Production Planning Model')

# Show the plot
plt.show()
```


Feasible Region of Production Planning Model
12000
11800
11600
11400
11200
11000
Objective Function
1575
1550
1525
1500
1475
1450
1425
x2
−0.04
−0.02
0.00
0.02
0.04
x1

In [19]:

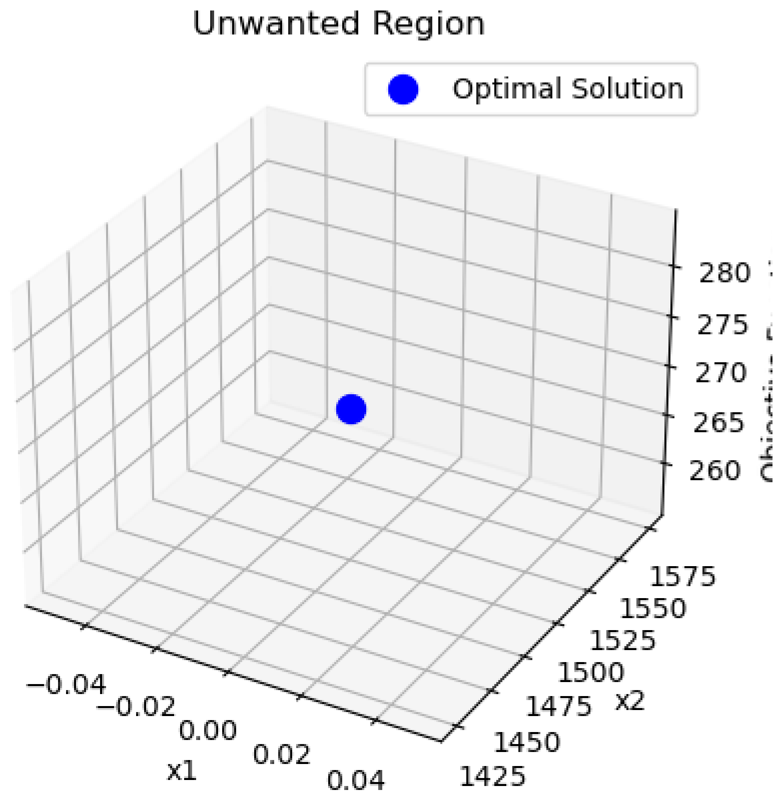
```
from pulp import LpProblem, LpMinimize, LpVariable

# LpProblem
model = LpProblem("Financial_portfolio_optimisation", sense=LpMinimize)

# Variables
x1 = LpVariable("x1", lowBound=0, cat='Continuous')
x2 = LpVariable("x2", lowBound=0, cat='Continuous')
x3 = LpVariable("x3", lowBound=0, cat='Continuous')  # Added missing line

# Objective function
model += 0.08*x1 + 0.1*x2 + 0.12*x3  # Corrected the objective function

# Constraints
model += 2*x1 + 3*x2 + x3 <= 10000, "investment constraint"
model += x1 <= 2000, "mininimum_investment constraint"
model += x2 >= 1500, "x1 lower bound"
model += x3 >= 1000, "x2 lower bound"

# Solving
model.solve()

# Results
optimal_x1 = x1.varValue
optimal_x2 = x2.varValue
optimal_x3 = x3.varValue

print("Optimal solution:")
print("x1:", optimal_x1)
print("x2:", optimal_x2)
print("x3:", optimal_x3)
```

```
Optimal solution:
x1: 0.0
x2: 1500.0
x3: 1000.0
```

In [18]:

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

# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Optimal solution values
optimal_x1 = x1.varValue
optimal_x2 = x2.varValue
optimal_x3 = x3.varValue

# Define a grid
x1_vals = np.linspace(0, optimal_x1 + 500, 50)  # Adjust the range for vis
x2_vals = np.linspace(0, optimal_x2 + 500, 50)  # Adjust the range for vis
x3_vals = np.linspace(0, optimal_x3 + 500, 50)  # Adjust the range for vis

# Create a meshgrid
X1, X2, X3 = np.meshgrid(x1_vals, x2_vals, x3_vals)

# Calculate the objective function values for each combination of variable
Z = 0.08 * X1 + 0.1 * X2 + 0.12 * X3

# Constraints
investment_constraint = 2 * X1 + 3 * X2 + X3
min_investment_constraint = X1
lower_bound_x2 = -X2 + 1500  # To flip the inequality
lower_bound_x3 = -X3 + 1000  # To flip the inequality

# Create a mask for the unwanted region
unwanted_region = np.logical_or(investment_constraint > 10000, np.logical_

# Scatter plot the optimal solution point
ax.scatter(optimal_x1, optimal_x2, 0.08 * optimal_x1 + 0.1 * optimal_x2 +

# Set labels and title
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Objective Function')
ax.set_title('Unwanted Region')

# Add a legend
ax.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x25ab46e9690>


Unwanted Region
Optimal Solution
280
275
270
265
260
1575
1550
1525
1500
1475
1450
1425
x2
−0.04
−0.02
0.00
0.02
0.04
x1

In [ ]: