# Linear Programming Problems

Model each one of the following problems as a linear programming problem. Make a list indicating clearly the variables and their meaning, the objective function, and the constraints. Once each of the problems is clearly stated, solve it using both the Excel Solver, and the functions “linprog” or “intlinprog” in the MATLAB optimization toolbox. You can choose to use another tool instead of MATLAB, for example Python. If that is the case, indicate the name of the optimization library used, and provide a link where it can be downloaded or purchased.

# Problem 1

A manufacturing firm has discontinued the production of a certain unprofitable article. This has created considerable excess production capacity. The management is considering devoting this excess capacity to the production of one or more of three possible different articles: call them article 1, article 2 and article 3. The available capacity on the machines, which might limit production, is summarized in the following table:

|  |  |
| --- | --- |
| Machine type | Available machine time (hours per week). |
| Milling machine | 200 |
| Lathe | 100 |
| Grinder | 50 |

The number of machine hours required by each one of the possible articles is give in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Machine type | Article 1 | Article 2 | Article 3 |
| Milling machine | 8 | 2 | 3 |
| Lathe | 4 | 3 |  |
| Grinder | 2 |  | 1 |

The sales department indicates that the sales potential for products 1 and 2 exceeds the maximum production rate, and that the sales potential for products 3 is 20 units per week.

The unit profit would be $200, $60 and $80, respectively, on articles 1, 2 and 3.

Determine how many pieces of each new article the firm should produce in order to maximize profit.

Notice that the numbers of articles must be integers (i.e., we cannot produce a fraction of an article).

## Python solution

import numpy as np

import scipy.optimize as spo

from scipy.optimize import linprog

from gekko import GEKKO

m = GEKKO(remote = False)

#m.options.Linear = 1 #Tells the solver that the problem in linear

#Problem 1

# variables with bounds

x1 = m.Var(lb=0, ub=None, integer=True)

x2 = m.Var(lb=0, ub=None, integer=True)

x3 = m.Var(lb=0, ub=20, integer=True)

x = [x1,x2,x3]

# inequality constraints

# A[i][j]: i is the index of machine and j is the index of article

A = [[8,2,3],[4,3,0],[2,0,1]] #needed hours per machine per article

milling\_machine\_time = 200

lathe\_time = 100

grinder\_time = 50

#maximum available hours per machine

b = [milling\_machine\_time,lathe\_time,grinder\_time]

#Maximum available time on each machine

m.Equation(x1\*A[0][0]+x2\*A[0][1]+x3\*A[0][2]<=b[0])

m.Equation(x1\*A[1][0]+x2\*A[1][1]+x3\*A[1][2]<=b[1])

m.Equation(x1\*A[2][0]+x2\*A[2][1]+x3\*A[2][2]<=b[2])

# Objective

c = [200,60,80] #profit per unit

m.Maximize(c[0]\*x1+c[1]\*x2+c[2]\*x3)

#solution

m.options.SOLVER=1

m.solve(disp=True)

print(f'product 1: {x1[0]}')

print(f'product 2: {x2[0]}')

print(f'product 3: {x3[0]}')

## Console print

Graphical user interface, application

Description automatically generated

# Problem 2

Find out what it the cheapest diet based on milk, beef and eggs, such that the daily requirements of vitamins A, C and D are satisfied, according to the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vitamin:  milligrams/ration of food | Ration of milk | Ration of beef | Ration of egg | Minimum daily |
| A | 1 | 1 | 10 | 1 mg |
| C | 100 | 10 | 10 | 50 mg |
| D | 10 | 100 | 10 | 10 mg |
| Cost per ration | $1.0 | $2.0 | $0.50 |  |

Consider two scenarios:

1. Fractions of rations are allowed, for example, 0.7 ration of egg.
2. Fraction of rations are not allowed (the diet must consist of integer numbers of rations).

## Python solution, scenario a)

import numpy as np

from gekko import GEKKO

m = GEKKO(remote = False)

integer\_solutions = False

A = [[1,1,10],[100,10,10],[10,100,10]] *#vitamin mg/ration*

b = [1,50,10] *#minimum daily vitamin*

c = [1.0,2.0,0.5] *#cost per ration*

*#decision variables*

rations\_milk = m.Var(lb=0,ub=None, integer=integer\_solutions)

rations\_beef = m.Var(lb=0,ub=None, integer=integer\_solutions)

rations\_egg = m.Var(lb=0,ub=None, integer=integer\_solutions)

x = [rations\_milk,rations\_beef,rations\_egg]

*#constraints*

m.Equation(A[0][0]\*x[0]+A[0][1]\*x[1]+A[0][2]\*x[2] >= 1)

m.Equation(A[1][0]\*x[0]+A[1][1]\*x[1]+A[1][2]\*x[2] >= 50)

m.Equation(A[2][0]\*x[0]+A[2][1]\*x[1]+A[2][2]\*x[2] >= 10)

*#objective*

m.Minimize(c[0]\*x[0]+c[1]\*x[1]+c[2]\*x[2])

m.options.SOLVER=1

m.solve(disp=True)

print(f'{ x = }')

print(f'Vitamin A: {(A[0][0]\*x[0][0]+A[0][1]\*x[1][0]+A[0][2]\*x[2][0])}')

print(f'Vitamin B: {(A[1][0]\*x[0][0]+A[1][1]\*x[1][0]+A[1][2]\*x[2][0])}')

print(f'Vitamin C: {A[2][0]\*x[0][0]+A[2][1]\*x[1][0]+A[2][2]\*x[2][0]}')

## Console print

Graphical user interface, text

Description automatically generated

\*Slight rounding errors

## Python solution, scenario b)

import numpy as np

from gekko import GEKKO

m = GEKKO(remote = False)

integer\_solutions = True

A = [[1,1,10],[100,10,10],[10,100,10]] *#vitamin mg/ration*

b = [1,50,10] *#minimum daily vitamin*

c = [1.0,2.0,0.5] *#cost per ration*

*#decision variables*

rations\_milk = m.Var(lb=0,ub=None, integer=integer\_solutions)

rations\_beef = m.Var(lb=0,ub=None, integer=integer\_solutions)

rations\_egg = m.Var(lb=0,ub=None, integer=integer\_solutions)

x = [rations\_milk,rations\_beef,rations\_egg]

*#constraints*

m.Equation(A[0][0]\*x[0]+A[0][1]\*x[1]+A[0][2]\*x[2] >= 1)

m.Equation(A[1][0]\*x[0]+A[1][1]\*x[1]+A[1][2]\*x[2] >= 50)

m.Equation(A[2][0]\*x[0]+A[2][1]\*x[1]+A[2][2]\*x[2] >= 10)

*#objective*

m.Minimize(c[0]\*x[0]+c[1]\*x[1]+c[2]\*x[2])

m.options.SOLVER=1

m.solve(disp=True)

print(f'{ x = }')

print(f'Vitamin A: {(A[0][0]\*x[0][0]+A[0][1]\*x[1][0]+A[0][2]\*x[2][0])}')

print(f'Vitamin B: {(A[1][0]\*x[0][0]+A[1][1]\*x[1][0]+A[1][2]\*x[2][0])}')

print(f'Vitamin C: {A[2][0]\*x[0][0]+A[2][1]\*x[1][0]+A[2][2]\*x[2][0]}')

## Console print

Text

Description automatically generated

# Problem 3

Consider a product mix problem within the context of a simplified oil refinery situation. Suppose that the refinery wishes to blend four petroleum constituents into three grades of gasoline: A, B, and C. The problem is to determine the mix of the four constituents that will maximize profit.

The availability and costs of the four constituents are given on the table below:

|  |  |  |
| --- | --- | --- |
| Constituent | Maximum quantity available in barrels per day | Cost per barrel |
| 1 | 3000 | $3.0 |
| 2 | 2000 | $6.0 |
| 3 | 4000 | $4.0 |
| 4 | 1000 | $5.0 |

To maintain the required quality for each grade of gasoline, it is necessary certain maximum or minimum percentages of the constituents in each blend. These are given in the table below, along with the selling price for each grade:

|  |  |  |
| --- | --- | --- |
| Grade | Specifications | Selling price per barrel |
| A | Not more than 30% of 1.  Not less than 40% of 2.  Not more than 50% of 3. | $5.50 |
| B | Not more than 50% of 1.  Not less than 10% of 2. | $4.50 |
| C | Not more than 70% of 1. | $3.50 |

Assume that all other cash flows are fixed, so that the «profit» to be maximized is total sales minus the total cost of constituents. Determine the optimal amount and blend of each grade of gasoline. Fractions of barrels are allowed on the solution.

## Python solution

import numpy as np

from gekko import GEKKO

m = GEKKO(remote = False)

*# Used to tell the library I don't want integer solutions later*

integer\_solutions = False

*#maximum barrels of constituents 1,2,3,4*

const\_max = [3000,2000,4000,1000]

*#price per barrel of costituent 1,2,3,4*

const\_cost = [3.0,6.0,4.0,5.0]

*#price per barrel of gasoline product A,B,C*

prod\_price = [5.50,4.50,3.50]

*#Mix percentages: A\_1 == ratio of constituent 1 in gasoline mix A*

A\_1 = m.Var(lb=0,ub=1, integer=integer\_solutions)

A\_2 = m.Var(lb=0,ub=1, integer=integer\_solutions)

A\_3 = m.Var(lb=0,ub=1, integer=integer\_solutions)

A\_4 = m.Var(lb=0,ub=1, integer=integer\_solutions)

B\_1 = m.Var(lb=0,ub=1, integer=integer\_solutions)

B\_2 = m.Var(lb=0,ub=1, integer=integer\_solutions)

B\_3 = m.Var(lb=0,ub=1, integer=integer\_solutions)

B\_4 = m.Var(lb=0,ub=1, integer=integer\_solutions)

C\_1 = m.Var(lb=0,ub=1, integer=integer\_solutions)

C\_2 = m.Var(lb=0,ub=1, integer=integer\_solutions)

C\_3 = m.Var(lb=0,ub=1, integer=integer\_solutions)

C\_4 = m.Var(lb=0,ub=1, integer=integer\_solutions)

A = [[A\_1, B\_1, C\_1],[A\_2,B\_2,C\_2],[A\_3,B\_3,C\_3],[A\_4,B\_4,C\_4]]

*#TOTAL amount of each product*

x\_A = m.Var(lb=0,ub=None)

x\_B = m.Var(lb=0,ub=None)

x\_C = m.Var(lb=0,ub=None)

x = [x\_A,x\_B,x\_C]

*#Sum ratio equality constraints*

m.Equation(A\_1+A\_2+A\_3+A\_4 == 1)

m.Equation(B\_1+B\_2+B\_3+B\_4 == 1)

m.Equation(C\_1+C\_2+C\_3+C\_4 == 1)

*#Min/Max mix inequality constraints*

m.Equation(A\_1 <= 0.3)

m.Equation(A\_2 >= 0.4)

m.Equation(A\_3 <= 0.5)

m.Equation(B\_1 <= 0.5)

m.Equation(B\_2 >= 0.1)

m.Equation(C\_1 <= 0.7)

*#TOTAL amount of each constituents needed*

const\_1 = A\_1\*x\_A+B\_1\*x\_B+C\_1\*x\_C

const\_2 = A\_2\*x\_A+B\_2\*x\_B+C\_2\*x\_C

const\_3 = A\_3\*x\_A+B\_3\*x\_B+C\_3\*x\_C

const\_4 = A\_4\*x\_A+B\_4\*x\_B+C\_4\*x\_C

const = [const\_1, const\_2, const\_3, const\_4]

test = []

*#Utilization of mix percentage*

for i in range(4):

    test.append(m.Equation(const[i]<=const\_max[i]))

total\_cost = 0

total\_sales = 0

for i in range(4):

*#for each sum of constituents, multiply their cost and add it to the total*

    total\_cost += const[i]\*const\_cost[i]

for i in range(3):

*#for each TOTAL amount of each product, multiply their price and add it to the total*

    total\_sales += x[i]\*prod\_price[i]

*#\*The objective function is the profit*

profit = total\_sales-total\_cost

m.Maximize(profit)

m.solve(disp=True)

*#Barrels of each product produced*

print(f'Amount of each product: { x = }')

*#! Note: the gekko library makes all variables into arrays, so they have to be indexed to get their value.*

*#Barrels of each constituent needed*

print("total amount of constituents needed:")

print(f'constituent 1: {A\_1[0]\*x\_A[0]+B\_1[0]\*x\_B[0]+C\_1[0]\*x\_C[0]}')

print(f'constituent 2: {A\_2[0]\*x\_A[0]+B\_2[0]\*x\_B[0]+C\_2[0]\*x\_C[0]}')

print(f'constituent 3: {A\_3[0]\*x\_A[0]+B\_3[0]\*x\_B[0]+C\_3[0]\*x\_C[0]}')

print(f'constituent 4: {A\_4[0]\*x\_A[0]+B\_4[0]\*x\_B[0]+C\_4[0]\*x\_C[0]}')

*#Optimal blends*

print(f'product A mix: c1 = {A\_1[0]}, c2 = {A\_2[0]}, c3 = {A\_3[0]}, c4 = {A\_4[0]}')

print(f'product B mix: c1 = {B\_1[0]}, c2 = {B\_2[0]}, c3 = {B\_3[0]}, c4 = {B\_4[0]}')

print(f'product C mix: c1 = {C\_1[0]}, c2 = {C\_2[0]}, c3 = {C\_3[0]}, c4 = {C\_4[0]}')

*#Total profit*

print("total profit:")

print(f'{ -m.options.OBJFCNVAL = }')

*#print(f'profit: {profit = }')*

## Console print

Text

Description automatically generated

# Problem 4

A certain farming organization operates three farms of comparable productivity. The output of each farm is limited both by the usable area and by the amount of water available for irrigation. The data for the upcoming season is contained in the following table:

|  |  |  |
| --- | --- | --- |
| Farm | Usable area in Hectares | Water available in Hectare-meter \* (See note below) |
| 1 | 200 | 250 |
| 2 | 150 | 333 |
| 3 | 300 | 150 |

The organization is considering three crops for planting, which differ primarily in their expected profit per Hectare and in their consumption of water. Furthermore, the total area that can be devoted to each of the crops is limited by the amount of appropriate harvest equipment available. This information is specified in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Crop | Maximum area in Ha | Water consumption in Hectare-meter/Hectare | Expected profit per hectare |
| A | 350 | 1.6 | $800 |
| B | 400 | 1.3 | $600 |
| C | 150 | 1.0 | $100 |

In order to maintain a uniform workload among the farms, it is the policy of the organization that the percentage of the usable area planted must be the same for each farm. However, any combination of the crops may be grown at any of the farms. The organization wishes to know how much of each crop should be planted at the respective farms in order to maximize the expected profit.

\*Note: A hectare-meter is a unit of volume of water, equivalent to the volume of water contained in one hectare (10 000 square meter) area irrigated with a water level of 1 meter.

## Python solution

import numpy as np

from gekko import GEKKO

m = GEKKO(remote = False)

*#m.options.Linear = 1 #Tells the solver that the problem in linear*

*#Problem 4*

*#Farms*

farm\_A\_max\_area = 200

farm\_B\_max\_area = 150

farm\_C\_max\_area = 300

area = [200,150,300]

farm\_A\_max\_water = 250

farm\_B\_max\_water = 333

farm\_C\_max\_water = 150

water = [250,333,150]

*#Crops*

water\_per\_hectare\_1 = 1.6

water\_per\_hectare\_2 = 1.3

water\_per\_hectare\_3 = 1.0

water\_per\_hectare = [1.6,1.3,1.0]

profit\_per\_hectare\_1 = 800

profit\_per\_hectare\_2 = 600

profit\_per\_hectare\_3 = 100

profit\_per\_hectare = [800,600,100]

*# farm areas with bounds*

x1 = m.Var(lb=0, ub=farm\_A\_max\_area, integer=False)

x2 = m.Var(lb=0, ub=farm\_B\_max\_area, integer=False)

x3 = m.Var(lb=0, ub=farm\_C\_max\_area, integer=False)

x = [x1,x2,x3]

*# percentage of each crop type*

*#Mix percentages (farm A,B,C and crop 1,2,3)*

A\_1 = m.Var(lb=0,ub=1, integer=False)

A\_2 = m.Var(lb=0,ub=1, integer=False)

A\_3 = m.Var(lb=0,ub=1, integer=False)

B\_1 = m.Var(lb=0,ub=1, integer=False)

B\_2 = m.Var(lb=0,ub=1, integer=False)

B\_3 = m.Var(lb=0,ub=1, integer=False)

C\_1 = m.Var(lb=0,ub=1, integer=False)

C\_2 = m.Var(lb=0,ub=1, integer=False)

C\_3 = m.Var(lb=0,ub=1, integer=False)

A = [[A\_1, B\_1, C\_1],[A\_2,B\_2,C\_2],[A\_3,B\_3,C\_3]]

*# Sum percentage equality constraints*

m.Equation(A[0][0]+A[1][0]+A[2][0] == 1)

m.Equation(A[0][1]+A[1][1]+A[2][1] == 1)

m.Equation(A[0][2]+A[1][2]+A[2][2] == 1)

*# Same percentage used of each farm*

m.Equation(x[0]/area[0] == x[1]/area[1])

m.Equation(x[1]/area[1] == x[2]/area[2])

*# water consumption inequality constraint (max water per farm)*

*# sum(percentage\_of\_crop\_n\*needed\_water\_per\_hectar\_crop\_n)\*utilized\_area\_farm\_m*

m.Equation((A[0][0]\*water\_per\_hectare[0]+A[1][0]\*water\_per\_hectare[1]+A[2][0]\*water\_per\_hectare[2])\*x[0] <= farm\_A\_max\_water)

m.Equation((A[0][1]\*water\_per\_hectare[0]+A[1][1]\*water\_per\_hectare[1]+A[2][1]\*water\_per\_hectare[2])\*x[1] <= farm\_B\_max\_water)

m.Equation((A[0][2]\*water\_per\_hectare[0]+A[1][2]\*water\_per\_hectare[1]+A[2][2]\*water\_per\_hectare[2])\*x[2] <= farm\_C\_max\_water)

*# Objective*

profit = 0

*#x[i] is total utilized area on farm i*

*#A[j,i]: j points to crop, i references the farm*

*#profit\_per\_hectare[j] is the profit per hectare for crop j*

for i in range(3):

*#for each farm*

    for j in range(3):

*#for each crop*

        profit += x[i]\*A[j][i]\*profit\_per\_hectare[j]

m.Maximize(profit)

*#solution*

m.options.SOLVER=1

m.solve(disp=True)

*#Utilized percentage of farms:*

print(f'Utilization of farm 1: {x[0][0]/area[0]}')

print(f'Utilization of farm 2: {x[1][0]/area[1]}')

print(f'Utilization of farm 3: {x[2][0]/area[2]}')

*#Crop percentage on each farm:*

print(f'Crop split of farm 1. Crop 1: {A[0][0][0]}, crop 2: {A[1][0][0]}, crop 3: {A[2][0][0]}')

print(f'Crop split of farm 2. Crop 1: {A[0][1][0]}, crop 2: {A[1][1][0]}, crop 3: {A[2][1][0]}')

print(f'Crop split of farm 3. Crop 1: {A[0][2][0]}, crop 2: {A[1][2][0]}, crop 3: {A[2][2][0]}')

*#Water utilization*

print(f'Water usage farm 1: {(A[0][0][0]\*water\_per\_hectare[0]+A[1][0][0]\*water\_per\_hectare[1]+A[2][0][0]\*water\_per\_hectare[2])\*x[0][0]} <= {farm\_A\_max\_water}')

print(f'Water usage farm 2: {(A[0][1][0]\*water\_per\_hectare[0]+A[1][1][0]\*water\_per\_hectare[1]+A[2][1][0]\*water\_per\_hectare[2])\*x[1][0]} <= {farm\_B\_max\_water}')

print(f'Water usage farm 3: {(A[0][2][0]\*water\_per\_hectare[0]+A[1][2][0]\*water\_per\_hectare[1]+A[2][2][0]\*water\_per\_hectare[2])\*x[2][0]} <= {farm\_C\_max\_water}')

profit\_maxed = 0

for i in range(3):

    for j in range(3):

        profit\_maxed += x[i][0]\*A[j][i][0]\*profit\_per\_hectare[j]

print(f'{profit\_maxed = }')

## Console print

Text

Description automatically generated