Linear Programming Summary

The University of Queensland

MATH3202

Matthew Rea
44304533
May 4, 2020

Summary

Notes

• constraints can never be $<, >$, they must always be $\leq, \geq$!

• slack variables can be added to change an inequality to an equality: i.e.
$ax_1 + bx_2 \leq c$ becomes $ax_1 + bx_2 + x_3 = c$, where x_3 is a slack variable

Farmer Jones

Problem

Farmer Jones bakes two types of cake (chocolate and plain) to supplement his income. Each chocolate cake can be sold for $4 and each plain cake can be sold for $2. Each chocolate cake requires 20 minutes of baking time, 250 mL of milk and 4 eggs, while each plain cake needs 50 minutes baking, 200 mL of milk and only 1 egg. In each day there are eight hours of baking time available. Farmer Jones' hens lay 30 eggs each day and his cows produce 5 L of milk. How many of each type of cake should Farmer Jones bake each day to maximise his revenue?

Solution

Sets

C, cakes {choc, plain}
I, ingredients {time, milk, eggs}

Data

r_c, revenue for each cake $c \in C$
a_i, available ingredient $i \in I$
u_{ic}, amount of $i \in I$ needed to make cake $c \in C$

Variables

x_c, number of cakes $c \in C$ to make

Objective

$$\max Z = 4x_{choc} + 2x_{plain}$$

OR

$$\max Z = \sum_{c \in C} r_c x_c$$

Constraints

Time: $20x_{choc} + 50x_{plain} \leq 480$
Eggs: $4x_{choc} + x_{plain} \leq 30$
Milk: $0.25x_{choc} + 0.2x_{plain} \leq 5$
Non-Negative Cakes: $x_{choc}, x_{plain} \geq 0$

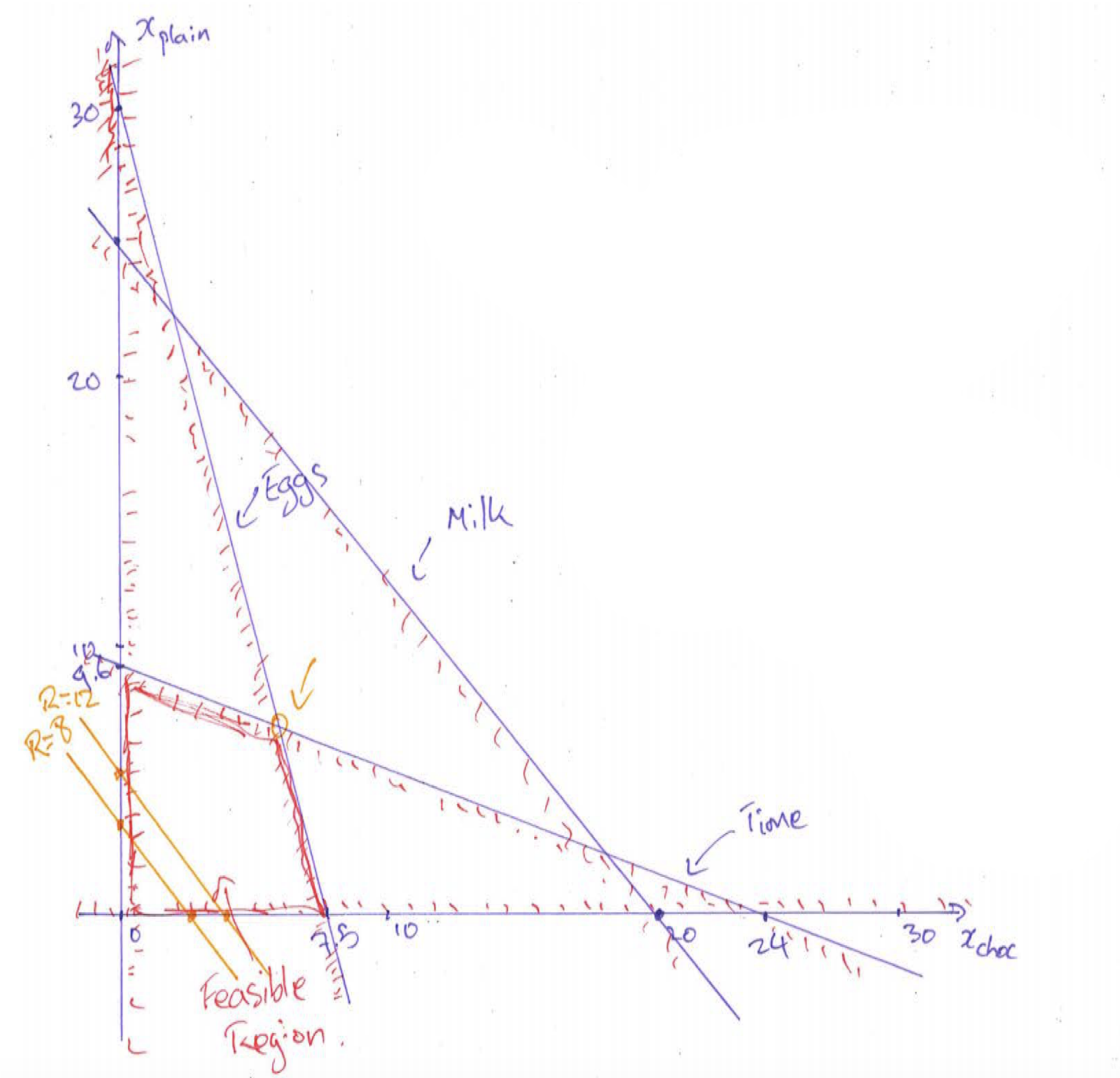
OR
Available Ingredients:

$$\sum_{c \in C} u_{ic} x_c \leq a_i, \ \forall i \in I$$

Non-Negative Cakes:

$$x_c \geq 0, \ \forall c \in C$$

This two-dimensional problem can be shown graphically, as seen below.



Code

```
from gurobipy import *

#Sets
Cakes = ["Chocolate","Plain"]
Ingredients = ["Time","Eggs","Milk"]

    #Create numeric vectors of sets
C = range(len(Cakes))
I = range(len(Ingredients))

#Data
r = [4, 2]
a = [8*60, 30, 5]
u = [[20,50],
     [4,1],
     [0.25,0.2]]

#Make model
m = Model("Farmer Jones")

#Variables
X = {}
for c in C:
    X[c] = m.addVar(vtype=GRB.INTEGER)

#Objective
m.setObjective(quicksum(r[c]*X[c] for c in C), GRB.MAXIMIZE)

#Constraints
for i in I:
    m.addConstr(quicksum(u[i][c]*X[c] for c in C) <= a[i])

#Optimise model
m.optimize()

#Print results
for c in C:
    print("Bake", X[c].x, Cakes[c])
print("Revenue is", m.objVal)
```

This gives the optimum value of \$36.00.

Wordlock

Problem

A padlock is to be made so that four-letter words are used for the password, as opposed to number combinations. There are 4 dials, with 10 letters on each dial (not necessarily the same letters on each dial). The aim is to determine which letters should be placed on which dial so to maximise the number of four-letter word combinations possible.

Solution

Sets

L, letters {A, B, ...}
D, dials {0, 1, 2, 3}
W, words {ABET, ABLE, ...}

Data

$$\delta_{wdl} = \begin{cases} 1, & \text{if word } w \in W \text{ had letter } l \in L \text{ on dial } d \in D \\ 0, & \text{otherwise} \end{cases}$$

Variables

$$x_{dl} = \begin{cases} 1, & \text{if letter } l \in L \text{ is on dial } d \in D \\ 0, & \text{otherwise} \end{cases}$$

$$y_w = \begin{cases} 1, & \text{if we can spell word } w \in W \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\max Z = \sum_{w \in W} y_w = y_{ABET} + y_{ABLE} + \dots$$

Constraints

10 letters on each dial:

$$\sum_{l \in L} x_{dl} = 10, \ \forall d \in D$$

Only spell word if letter on dial:

$$y_w \le x_{dl}, \ \forall w \in W, \ d \in D, \ l \in L, \text{ such that } \delta_{wdl} = 1$$

$$(\text{i.e. } y_{ABET} \le x_{0A}, \ y_{ABET} \le x_{1B}, \ y_{ABET} \le x_{2E}, \ y_{ABET} \le x_{3T})$$

Disclude profanity:

$$x_{0F} + x_{1U} + x_{2C} + x_{3K} \le 3$$

The Cost of Subsistence

Problem

George Stigler's 1945 paper "The Cost of Subsistence" (Journal of Farm Economics, 27, 303-314) presents one of the earliest applications of linear programming, that of finding minimum-cost diets:

"Elaborate investigations have been made of the adequacy of diets at various income levels, and a considerable number of 'low-cost,' 'moderate,' and 'expensive' diets have been recommended to consumers. Yet, so far as I know, no one has determined the minimum cost of obtaining the amounts of calories, protein, minerals, and vitamins which these studies accept as adequate or optimum."

Solution

Sets

F, foods
N, nutrients

Data

v_{nf}, value of nutrient $n \in N$ in food $f \in F$
c_f, cost per unit weight of food $f \in F$
$nmin_n$, minimum requirement of nutrient $n \in N$
$nmax_n$, maximum requirement of nutrient $n \in N$

Variables

x_f, quantity of food $f \in F$

Objective

$$\min Z = \sum_{f \in F} x_f c_f$$

Constraints

Min Nutrients:

$$\sum_{f \in F} v_{nf} x_f \geq nmin_n, \ \forall n \in N$$

Max Nutrients:

$$\sum_{f \in F} v_{nf} x_f \leq nmax_n, \ \forall n \in N$$

Non-Negative Food:

$$x_f \geq 0, \ \forall f \in F$$

Code

```
from gurobipy import *

#Sets
Foods = [’almonds’, ’apples’, ’apricots’, ’banana’, ’brie’,
         ’broccoli’, ’brown rice’, ’camembert’, ’carrots’,
         ’chicken’, ’chocolate’, ’couscous’, ’cream cheese’,
         ’croissants’, ’cucumber’, ’currants’, ’custard’,
         ’dark chocolate’, ’fried eggs’, ’green beans’, ’ham’,
         ’hamburgers’, ’hazelnuts’, ’herrings’, ’mushrooms’,
         ’potato chips’, ’roast pork’, ’tomato soup’,
         ’white bread’, ’white rice’]

Nutrients = [’energy’, ’protein’, ’fibre’, ’iron’, ’calcium’,
             ’vitc’, ’thiamin’, ’riboflavin’, ’vita’, ’zinc’,
             ’folate’, ’niacin’, ’sodium’]

    #Create numeric vectors of sets
F = range(len(Foods))
N = range(len(Nutrients))

#Data
    #Costs ($/100g) from coles.com.au [2020-02-26]
C = [1.76, 0.59, 0.89, 0.45, 4.72, 0.69, 0.32, 3.00, 0.22, 0.95,
     2.78, 0.56, 1.88, 1.32, 0.69, 1.63, 0.42, 2.78, 0.56, 1.09,
     2.20, 0.90, 2.67, 1.20, 1.10, 2.06, 1.00, 0.48, 0.49, 0.14]

    #Nutritional data from Australian Food Composition Database
    #https://www.foodstandards.gov.au/science/monitoringnutrients/
    #afcd/Pages/default.aspx
v = [[2503, 19.5, 8.8, 3.9, 250, 0, 0.19, 1.4, 2, 3.69, 29, 7.74,
      5],
     [206, 0.3, 2.5, 0.17, 3, 5, 0.02, 0.01, 3, 0.07, 0, 0.14, 2],
     [171, 0.8, 2.5, 0.3, 15, 12, 0.025, 0.035, 60, 0.15, 6, 1.38,
      2],
     [385, 1.4, 2.4, 0.29, 5, 4, 0.02, 0.047, 6, 0.16, 33, 0.6,
      0],
     [1465, 18.6, 0, 0.21, 464, 0, 0.013, 0.483, 371, 2.71, 49,
      4.73, 593],
     [129, 4.6, 3.8, 0.85, 33, 57, 0.063, 0.193, 46, 0.6, 31, 1.2,
      22],
     [639, 2.9, 1.5, 0.5, 5, 0, 0.14, 0.02, 0, 0.9, 16, 2.35, 3],
     [1286, 19.5, 0, 0.15, 484, 0, 0, 0, 0, 0, 0, 0, 0],
     [132, 0.8, 3.9, 0.28, 30, 6, 0.079, 0.04, 1316, 0.2, 18, 0.9,
      40],
     [637, 29, 0, 0.5, 9, 0, 0.05, 0.11, 7, 0.83, 3, 17.7, 46],
     [2206, 7.6, 2.3, 1.42, 252, 5, 0.05, 0.325, 79, 1.24, 36,
      1.59,
      68],
     [663, 5.2, 2.2, 0.48, 12, 0, 0.064, 0.034, 0, 0.37, 9, 2.41,
      6],
     [1384, 8.2, 0, 0.14, 82, 0, 0.05, 0.239, 350, 0.58, 0, 1.65,
      336],
```

```
      [1500, 10, 2.8, 0.95, 52, 0, 0.11, 0.09, 0, 0.75, 0, 1.67,
       457],
      [51, 0.4, 1, 0.27, 57, 13, 0.018, 0.018, 15, 0.18, 0, 0.34,
       19],
      [1167, 2.8, 6, 2.3, 87, 0, 0.11, 0, 2, 0.5, 0, 1.47, 46],
      [407, 3.5, 0, 0.05, 120, 0, 0.052, 0.218, 8, 0.41, 0, 0.57,
       61],
      [2142, 3.9, 1.2, 4.4, 52, 0, 0.05, 0.13, 21, 2, 13, 1.95,
       55],
      [1039, 16.2, 0, 2, 69, 0, 0.1, 0.38, 200, 1.3, 58, 4.74,
       146],
      [89, 1.5, 2.5, 1.1, 30, 13, 0.03, 0.07, 77, 0.8, 33, 0.66,
       3],
      [467, 17, 1.8, 0.66, 10, 0, 0.386, 0.065, 0, 1.91, 23, 7.17,
       1167],
      [974, 12.5, 1.7, 3.1, 74, 1, 0.26, 0.18, 0, 0, 0, 3.89, 477],
      [2689, 14.8, 10.4, 3.2, 86, 0, 0.39, 0.17, 3, 2.2, 113, 4.67,
       3],
      [1031, 14.2, 0, 1.2, 77, 0, 0.036, 0.139, 17, 0.5, 2, 5.67,
       870],
      [194, 6.2, 2.9, 0.5, 5, 2, 0.042, 0.661, 4, 1.06, 27, 7.02,
       15],
      [2160, 6, 3.5, 1.13, 20, 23, 0.16, 0.01, 0, 1.4, 67, 2.68,
       618],
      [699, 34.5, 0, 1.35, 6, 0, 0.857, 0.309, 0, 3.67, 2, 10.23,
       54],
      [193, 1.4, 0.2, 0.21, 37, 1, 0.127, 0.074, 30, 0.21, 2, 0.87,
       360],
      [1027, 9.7, 2.8, 1.48, 62, 0, 0.398, 0.049, 0, 0.83, 254,
       6.52, 456],
      [671, 2.7, 1, 0.61, 79, 0, 0.015, 0.02, 0, 0.46, 7, 0.56, 3]]

    #Recommended nutritional requirements for Tindra Lund
    #(a current student at Hofn University on the Islands)
    #from Nutrient Reference Values for Australia and New Zealand
    #A maximum value of -1 indicates there is no upper limit
nmin = [8491, 45.4, 23.1, 16.1, 1186, 41.9, 1.1, 1.1, 700, 7.38,
        400, 14, 690]
nmax = [10190, 68.1, -1, 45, 2500, -1, -1, -1, 2875, 36.9, 876.2,
        -1, 2300]

#Make model
m = Model("Diet")

#Variables
X = {}
for f in F:
    X[f] = m.addVar()

#Objective
m.setObjective(quicksum(C[f]*X[f] for f in F), GRB.MINIMIZE)

#Constraints
for n in N:
```

```
    m.addConstr(quicksum(v[f][n]*X[f] for f in F) >= nmin[n])
    if nmax[n] > 0:
        m.addConstr(quicksum(v[f][n]*X[f] for f in F) <= nmax[n])

#Optimise model
m.optimize()

#Print results
for f in F:
    if X[f].x > 0:
        print(Foods[f],round(100*X[f].x))
print("Cost:", m.objVal)
```

This gives the optimum value of \$6.40.

Giapetto's Wood-Carving

Problem

Giapetto's Woodcarving manufactures toy trains and toy soldiers. A soldier uses \$10 worth of raw materials and \$14 worth of labour and sells for \$27. A train uses \$9 worth of raw materials and \$10 worth of labour and sells for \$21. A soldier needs 2 hours finishing and 1 hour of carpentry. A train needs 1 hour of finishing and 1 hour of carpentry. Giapetto can obtain all the raw materials it needs but only has 100 hours of finishing and 80 hours of carpentry per week. They can sell any number of trains each week, but at most 40 toy soldiers.
How many of each kind of toy should be produced per week to maximise revenue?

Solution

Sets
T, toys {soldiers = 1, trains = 2}
P, parts {finishing, carpentry}

Data
r_t, revenue for toy $t \in T$
d_t, demand for toy $t \in T$

Variables
x_t, number of toys $t \in T$

Objective

$$\max Z = \sum_{t \in T} x_t r_t$$

Constraints
Finishing:

$$2x_1 + x_2 \leq 100$$

Carpentry:

$$x_1 + x_2 \leq 80$$

Demand:

$$x_1 \leq 40$$

Non-Negative Toys:

$$x_1, x_2 \geq 0$$

Oil-Blending

Problem

A food manufacturer refines raw oils and blends them together. The raw oils come in two categories – vegetable oils, of which there are two types; and non-vegetable oils, of which there are three types.

The oils are refined on different production lines. In any month it is not possible to refine more than 200 tonnes of vegetable oil and more than 250 tonnes of non-vegetable oils. There is no loss of mass in the refining process and the cost of refining may be ignored.

There is a technological restriction on the "hardness" of the final product. In the units in which hardness is measured it must lie between 3 and 6. The hardness of a blended product is the weighted average of its components. The cost per tonne and hardness of the raw oils are:

Oil	**Cost**	**Hardness**
Veg 1	$110	8.8
Veg 2	$120	6.1
Oil 1	$130	2.0
Oil 2	$110	4.2
Oil 3	$115	5.0

The final product sells for $150 per tonne. How should the food manufacturer make this product in order to maximise net profit?

Solution

Sets

F, final product {final blended oil product}
I, oils {Veg 1, Veg 2, Oil 1, Oil 2, Oil 3}

Data

$cost_i$, cost of oil $i \in I$ (\$ / tonne)
$hard_i$, hardness of oil $i \in I$ (units / tonne)
$isveg_i$, 1 if oil $i \in I$ is vegetable oil; 0 otherwise
$sell$, selling price of final product F (\$ / tonne)
$vegmax$, maximum vegetable oil that can be processed (tonne)
$nonvegmax$, maximum non-vegetable oil that can be processed (tonne)
$hmin$, minimum hardness of final product F (units)
$hmax$, maximum hardness of final product F (units)

Variables

x_i, amount of oil $i \in I$ to process (tonne)

Objective

$$\max Z = \sum_{i \in I} (sell - cost_i) x_i$$

Constraints

Maximum Vegetable Oil:

$$\sum_{i \in I} x_i \leq vegmax, \text{ for } isveg_i = 1$$

Maximum Non-Vegetable Oil:

$$\sum_{i \in I} x_i \leq nonvegmax, \text{ for } isveg_i = 0$$

Hardness Bounds:

$$hmin \leq \frac{\sum_{i \in I} x_i hard_i}{\sum_{i \in I} x_i} \leq hmax \text{ (this is non-linear!)}$$

$$hmin \sum_{i \in I} x_i \leq \sum_{i \in I} x_i hard_i \leq hmax \sum_{i \in I} x_i \text{ (which becomes)}$$

1) $0 \leq \sum_{i \in I} x_i hard_i - hmin \sum_{i \in I} x_i \implies \sum_{i \in I} (hard_i - hmin) x_i \geq 0$

2) $\sum_{i \in I} x_i hard_i - hmax \sum_{i \in I} x_i \leq 0 \implies \sum_{i \in I} (hard_i - hmax) x_i \leq 0$

Non-Negative Oil:

$$x_i \geq 0, \ \forall i \in I$$

Code

```
from gurobipy import *

#Sets
Oils = ["Veg1","Veg2","Oil1","Oil2","Oil3"]

    #Create numeric vectors of sets
O = range(len(Oils))

#Data
cost = [110,120,130,110,115]
hard = [8.8,6.1,2.0,4.2,5.0]
isveg = [True,True,False,False,False]
vegmax = 200
nonvegmax = 250
sell = 150
hmin = 3
hmax = 6

#Make model
m = Model("Oil Blending")

#Variables
X = {}
for i in O:
    X[i] = m.addVar()

#Objective
m.setObjective(quicksum((sell-cost[i])*X[i] for i in O),
               GRB.MAXIMIZE)

#Constraints
m.addConstr(quicksum(X[i] for i in O if isveg[i]) <= vegmax)
m.addConstr(quicksum(X[i] for i in O if not isveg[i]) <= nonvegmax)
m.addConstr(quicksum((hard[i] - hmax)*X[i] for i in O) <= 0)
m.addConstr(quicksum((hard[i] - hmin)*X[i] for i in O) >= 0)

#Optimise model
m.optimize()

#Print results
for i in O:
    print(Oils[i], "process", round(X[i].x, 2), "tonnes")
print("Profit:", m.objVal)
#Hard: sum([hard[i]*X[i].x for i in O])/sum([X[i].x for i in O]))
```

This gave the optimum value of $17,592.59.

Oil-Blending Advanced

Problem

A food manufacturer refines raw oils and blends them together. The raw oils come in two categories – vegetable oils, of which there are two types; and non-vegetable oils, of which there are three types.

The oils are refined on different production lines. In any month it is not possible to refine more than 200 tonnes of vegetable oil and more than 250 tonnes of non-vegetable oils. There is no loss of mass in the refining process and the cost of refining may be ignored.

There is a technological restriction on the "hardness" of the final product. In the units in which hardness is measured it must lie between 3 and 6. The hardness of a blended product is the weighted average of its components. The hardness of the raw oils are:

Oil	Veg 1	Veg 2	Oil 1	Oil 2	Oil 3
Hardness	8.8	6.1	2.0	4.2	5.0

The raw oils may be purchased for immediate delivery (January) or bought on the futures market for delivery in subsequent months. Prices now and in future months are given by the following table:

Oil	January	February	March	April	May	June
Veg 1	$110	$130	$110	$120	$100	$90
Veg 2	$120	$130	$140	$110	$120	$100
Oil 1	$130	$110	$130	$120	$150	$140
Oil 2	$110	$90	$100	$120	$110	$80
Oil 3	$115	$115	$95	$125	$105	$135

It is possible to store up to 1000 tonnes of each raw oil for use later. The cost of storage is $5 per tonne per month.

The final product sells for $150 per tonne.

There are currently 500 tonnes of each raw oil in storage. What buying and manufacturing policy should the company pursue in order to maximise profit?

Solution

Sets

I, oils {Veg 1, Veg 2, Oil 1, Oil 2, Oil 3}
T, months {Jan, ..., June}

Data

$cost_{it}$, cost of oil $i \in I$ in month $t \in T$ ($ / tonne)
$hard_i$, hardness of oil $i \in I$ (units / tonne)
$isveg_i$, 1 if oil $i \in I$ is vegetable oil; 0 otherwise
$storecost$, cost of storage ($ / tonne / month)
$storemax$, maximum storage (tonne) for each oil
$initial$, initial amount in storage of each oil
$sell$, selling price of final product F ($ / tonne)
$vegmax$, maximum vegetable oil that can be processed (tonne)
$nonvegmax$, maximum non-vegetable oil that can be processed (tonne)
$hmin$, minimum hardness of final product F (units)
$hmax$, maximum hardness of final product F (units)

Variables

x_i, amount of oil $i \in I$ to process (tonne)
s_{it}, amount of oil $i \in I$ (tonne) in storage at the end of month $t \in T$
y_{it}, amount of oil $i \in I$ (tonne) purchased in month $t \in T$

Objective

Maximise Profit = Revenue − Costs

$$\max Z = \sum_{i \in I} \sum_{t \in T} sell \times x_{it} - \left(\sum_{i \in I} \sum_{t \in T} storecost \times s_{it} + \sum_{i \in I} \sum_{t \in T} cost_{it} \times y_{it} \right)$$

Constraints

Initial Storage:

$$s_{i0} = initial - x_{i0} + y_{i0}, \ \forall i \in I$$

Storage at end of each month:

$$s_{it} = s_{i(t-1)} - x_{it} + y_{it}, \ \forall i \in I, \ t \in T, \text{ such that } t > 0$$

Minimum Hardness:

$$\sum_{i \in I} (hard_i - hmin) x_{it} \geq 0, \ \forall t \in T$$

Maximum Hardness:

$$\sum_{i \in I} (hard_i - hmax) x_{it} \leq 0, \ \forall t \in T$$

Maximum Vegetable Oil:

$$\sum_{i \in I, \ isveg_i = 1} x_{it} \leq vegmax, \ \forall t \in T$$

Maximum Non-Vegetable Oil:

$$\sum_{i \in I,\ isveg_i = 0} x_{it} \leq nonvegmax,\ \forall t \in T$$

Maximum Storage:

$$s_{it} \leq storemax,\ \forall i \in I,\ t \in T$$

Non-Negative Variables:

$$x_{it}, y_{it}, s_{it} \geq 0,\ \forall i \in I,\ t \in T$$

Code

```
from gurobipy import *

#Sets
Oils = ["Veg1","Veg2","Oil1","Oil2","Oil3"]
Months = ["Jan","Feb","Mar","Apr","May","Jun"]

    #Create numeric vectors of sets
I = range(len(Oils))
T = range(len(Months))

#Data
cost = [[110,120,130,110,115],
         [130,130,110,90,115],
         [110,140,130,100,95],
         [120,110,120,120,125],
         [100,120,150,110,105],
         [90,100,140,80,135]]
hard = [8.8,6.1,2.0,4.2,5.0]
isveg = [True,True,False,False,False]
vegmax = 200
nonvegmax = 250
sell = 150
hmin = 3
hmax = 6
storemax = 1000
storecost = 5
initial = 500

#Make model
m = Model("Oil Blending 2")

#Variables
X = {(i,t): m.addVar() for i in I for t in T}
S = {(i,t): m.addVar() for i in I for t in T}
Y = {(i,t): m.addVar() for i in I for t in T}

#Objective
m.setObjective(quicksum(sell*X[i,t] for i in I for t in T) -
        quicksum(storecost*S[i,t] for i in I for t in T) -
        quicksum(cost[t][i]*Y[i,t] for i in I for t in T),
```

```
        GRB.MAXIMIZE)

for t in T:
    m.addConstr(quicksum(X[i,t] for i in I if isveg[i]) <= vegmax)
    m.addConstr(quicksum(X[i,t] for i in I if not isveg[i]) <=
                nonvegmax)
    m.addConstr(quicksum((hard[i] - hmax)*X[i,t] for i in I) <= 0)
    m.addConstr(quicksum((hard[i] - hmin)*X[i,t] for i in I) >= 0)
    for i in I:
        m.addConstr(S[i,t] <= storemax)
        if t == 0:
            m.addConstr(S[i,t] == initial - X[i,t] + Y[i,t])
        else:
            m.addConstr(S[i,t] == S[i,t-1] - X[i,t] + Y[i,t])
for i in I:
    m.addConstr(S[i,T[-1]] == initial)

#Optimise model
m.optimize()

#Print results
print("Profit is",m.objVal)
print("Processing")
for i in I:
    print(Oils[i], [round(X[i,t].x,1) for t in T])
print("Storage")
for i in I:
    print(Oils[i], [round(S[i,t].x,1) for t in T])
print("Purchasing")
for i in I:
    print(Oils[i], [round(Y[i,t].x,1) for t in T])

print("Hardness")
print([sum([hard[i]*X[i,t].x for i in I])/sum([X[i,t].x
       for i in I]) for t in T])
```

This gave the optimum value of $107,842.59.

Portfolio Optimisation

Problem - Part 1

A manager must decide how to invest $100,000 for one year in different financial products. Her goal is to maximise earnings while avoiding high-risk exposure. The financial products available are given in the following table:

Financial Product	Market	Return (%)
1	Cars (Germany)	10.3
2	Cars (Japan)	10.1
3	Computers (USA)	11.8
4	Computers (Singapore)	11.4
5	Appliances (Europe)	12.7
6	Appliances (Asia)	12.2
7	Insurance (Germany)	9.5
8	Insurance (USA)	9.9
9	Short-term bonds	3.6
10	Medium-term bonds	4.2

The investment requirements are:

1. No more than $30,000 in the car options
2. No more than $30,000 in the computer options
3. No more than $20,000 in the insurance options
4. At least $20,000 in the insurance options
5. At least $25,000 in the bonds
6. At least 40% of the amount invested in medium-term bonds must be invested in short-term bonds
7. No more than $50,000 in Germany options
8. No more than $40,000 in USA options

The manager would like to know:

1. What investment portfolio will maximise earnings?
2. How much would the optimal earnings improve if we could put more than $20,000 in the appliance options?
3. How much higher would the return on Insurance (Germany) have to be before it became part of the optimal portfolio?

Solution - Part 1

Sets

P, products $\{1 = \text{Cars (Germany)}, 2 = \text{Cars (Japan)}, \dots\}$

Data

r_p, return (%) on investment for product $p \in P$

Variables

x_p, amount ($) to invest in product $p \in P$

Objective

$$\max Z = \frac{1}{100} \sum_{p \in P} r_p x_p$$

Constraints

Total money to invest:

$$\sum_{p \in P} x_p \leq 100000$$

Maximum amount to be invested in cars:

$$\sum_{p \in \{1,2\}} x_p \leq 30000$$

Maximum amount to be invested in computers:

$$\sum_{p \in \{3,4\}} x_p \leq 30000$$

Maximum amount to be invested in appliances:

$$\sum_{p \in \{5,6\}} x_p \leq 20000$$

Maximum amount to be invested in insurance:

$$\sum_{p \in \{7,8\}} x_p \leq 20000$$

Maximum amount to be invested in bonds:

$$\sum_{p \in \{9,10\}} x_p \geq 25000$$

At least 40% of value of medium-term bonds must be invested in short-term bonds:

$$x_9 \geq 0.4 x_{10}$$

Maximum amount to be invested in German options:

$$\sum_{p \in \{1,7\}} x_p \geq 50000$$

Maximum amount to be invested in USA options:

$$\sum_{p \in \{3,8\}} x_p \geq 40000$$

Problem - Part 2

The manager is now considering a two-year investment strategy, where $100,000 is invested each year. The returns for the first year are as in the table above. The returns for each product in the second year will depend upon the economic environment for the second year, which will not be known until the end of the first year. The possible scenarios are:

Scenario	**Probability**
Business as usual	0.80
Downturn	0.15
Upturn	0.04
Crash	0.01

The returns for each product in each scenario are found in the table below.

Financial Product	**Market**	**Scenario 1 Return (%)**	**Scenario 2 Return (%)**	**Scenario 3 Return (%)**	**Scenario 4 Return (%)**
1	Cars (Germany)	10.3	5.1	11.8	-30.0
2	Cars (Japan)	10.1	4.4	12.0	-35.0
3	Computers (USA)	11.8	10.0	12.5	1.0
4	Computers (Singapore)	11.4	11.0	11.8	2.0
5	Appliances (Europe)	12.7	8.2	13.4	-10.0
6	Appliances (Asia)	12.2	8.0	13.0	-12.0
7	Insurance (Germany)	9.5	2.0	14.7	-5.4
8	Insurance (USA)	9.9	3.0	12.9	-4.6
9	Short-term bonds	3.6	4.2	3.1	5.9
10	Medium-term bonds	4.2	4.7	3.5	6.3

The investments for the second year are made at the end of the first year. The investment requirements for the first year do not have to be followed in the second year but the amount invested in any product must be within $10,000 of the amount invested in that product in the first year. What is the manager's optimal two-year investment strategy?

Solution - Part 2

Sets

P, products $\{1 = \text{Cars (Germany)}, 2 = \text{Cars (Japan)}, \dots\}$
S, scenario $\{1, \dots, 4\}$

Data

r_{sp}, return (%) on investment for product $p \in P$ in scenario $s \in S$
$prob_s$, probability of scenario $s \in S$

Variables

x_p, amount (\$) to invest in product $p \in P$
y_{sp}, amount to invest in product $p \in P$ in year 2 for scenario $s \in S$

Objective

$$\max Z = \frac{1}{100}\left(\sum_{p \in P} r_p x_p + \sum_{s \in S}\sum_{p \in P} prob_s \times r_{sp} \times y_{sp}\right)$$

Constraints

Total money to invest in year 1:

$$\sum_{p \in P} x_p \leq 100000$$

Maximum amount to be invested in cars:

$$\sum_{p \in \{1,2\}} x_p \leq 30000$$

Maximum amount to be invested in computers:

$$\sum_{p \in \{3,4\}} x_p \leq 30000$$

Maximum amount to be invested in appliances:

$$\sum_{p \in \{5,6\}} x_p \leq 20000$$

Maximum amount to be invested in insurance:

$$\sum_{p \in \{7,8\}} x_p \leq 20000$$

Maximum amount to be invested in bonds:

$$\sum_{p \in \{9,10\}} x_p \geq 25000$$

At least 40% of value of medium-term bonds must be invested in short-term bonds:

$$x_9 \geq 0.4 x_{10}$$

Maximum amount to be invested in German options:

$$\sum_{p \in \{1,7\}} x_p \geq 50000$$

Maximum amount to be invested in USA options:

$$\sum_{p \in \{3,8\}} x_p \geq 40000$$

Total money to invest in year 2:

$$\sum_{p \in P} x_p \leq 100000$$

Amounts invested in year 2 must be within \$10,000 of those invested in year 1:

$$y_{sp} \leq x_p + 10000, \ \forall p \in P, \ s \in S$$

$$y_{sp} \geq x_p - 10000, \ \forall p \in P, \ s \in S$$

Code

```
from gurobipy import *

#Part 1

#Sets & Data (using a dictionary to define a set and associated data)
Products = {
    'Cars (Germany)': 10.3,
    'Cars (Japan)': 10.1,
    'Computers (USA)': 11.8,
    'Computers (Singapore)': 11.4,
    'Appliances (Europe)': 12.7,
    'Appliances (Asia)': 12.2,
    'Insurance (Germany)': 9.5,
    'Insurance (USA)': 9.9,
    'Short-term bonds': 3.6,
    'Medium-term bonds': 4.2
}

#Model
m = Model("Portfolio Optimisation")

#Variables (X[p] is amount to invest in product p (in first year))
X = {p: m.addVar() for p in Products}

#Objective
m.setObjective(quicksum(Products[p] * X[p] / 100 for p in Products),
               GRB.MAXIMIZE)

#Constraints (dictionary of the constraints so we can access them later
#             for sensitivity analysis)
Constraints = {
    'Total':
        m.addConstr(quicksum(X[p] for p in Products)
                            <= 100000),
    'Cars':
        m.addConstr(X['Cars (Germany)'] + X['Cars (Japan)']
                    <= 30000),
    'Computers':
        m.addConstr(X['Computers (USA)'] + X['Computers (Singapore)']
                    <= 30000),
    'Appliances':
        m.addConstr(X['Appliances (Europe)'] + X['Appliances (Asia)']
                    <= 20000),
    'Insurance':
        m.addConstr(X['Insurance (Germany)'] + X['Insurance (USA)']
                    >= 20000),
    'Bonds':
        m.addConstr(X['Short-term bonds'] + X['Medium-term bonds']
                    >= 25000),
    'Balance':
        m.addConstr(X['Short-term bonds']
                    >= 0.4 * X['Medium-term bonds']),
```

```
    'Germany':
        m.addConstr(X['Cars (Germany)'] + X['Insurance (Germany)']
                        <= 50000),
    'USA':
        m.addConstr(X['Computers (USA)'] + X['Insurance (USA)']
                        <= 40000)
}

#Optimise
m.optimize()

#Print results
for p in Products:
    print(p, X[p].x)

print("Total return = $", m.objVal)

#Sensitivity Analysis (to answer (b) and (c) in Part 1)
    #Sensitivity Analysis - Constraints
print("Sensitivity Analysis - Constraints")

    #PI is dual variable
for c in Constraints:
    print(c, Constraints[c].RHS,
            Constraints[c].Slack, round(Constraints[c].PI,3),
            Constraints[c].SARHSLow, Constraints[c].SARHSUp)

print("Dual objective = $", sum(Constraints[c].RHS * Constraints[c].PI
                                for c in Constraints))

    #Sensitivity Analysis - Variables
print("Sensitivity Analysis - Variables")

    #RC is reduced cost
for p in Products:
    print(p, round(X[p].obj, 3), X[p].x, round(X[p].RC, 3),
            round(X[p].SAObjLow, 3), round(X[p].SAObjUp, 3))

#Part 2

#Sets & Data (scenarios: business as usual, downturn, upturn, crash)
ScenarioProb = [0.8, 0.15, 0.04, 0.01]
S = range(len(ScenarioProb))

Year2Return = {
    'Cars (Germany)': [10.3, 5.1, 11.8, -30.0],
    'Cars (Japan)': [10.1, 4.4, 12.0, -35.0],
    'Computers (USA)': [11.8, 10.0, 12.5, 1.0],
    'Computers (Singapore)': [11.4, 11.0, 11.8, 2.0],
    'Appliances (Europe)': [12.7, 8.2, 13.4, -10.0],
    'Appliances (Asia)': [12.2, 8.0, 13.0, -12.0],
    'Insurance (Germany)': [9.5, 2.0, 14.7, -5.4],
    'Insurance (USA)': [9.9, 3.0, 12.9, -4.6],
    'Short-term bonds': [3.6, 4.2, 3.1, 5.9],
```

```
    'Medium-term bonds': [4.2, 4.7, 3.5, 6.3]
}

#Variables (X[p] is unchanged, Y[p,s] is amount to invest in
#           product p under scenario s in Year 2)
Y = {(p,s): m.addVar() for p in Products for s in S}

#Objective
m.setObjective(quicksum(Products[p] * X[p] / 100 for p in Products) +
               quicksum(ScenarioProb[s] * Year2Return[p][s] * Y[p,s]
                        / 100 for (p,s) in Y),
               GRB.MAXIMIZE)

#Constraints
Year2Total = {s: m.addConstr(quicksum(Y[p,s] for p in Products)
                             <= 100000) for s in S}

    #Constrain amounts to be within $10,000 of year 1
Year2LinkUp = {(p,s): m.addConstr(Y[p,s] <= X[p] + 10000)
               for (p,s) in Y}
Year2LinkDown = {(p,s): m.addConstr(Y[p,s] >= X[p] - 10000)
                 for (p,s) in Y}

#Optimise (note that we can optimise a model again within the same file,
#          now with the additional Y variables, constraints & updated
#          objective)
m.optimize()

#Print results
for p in Products:
    print(p, X[p].x)

for s in S:
    print('************', ScenarioProb[s])
    for p in Products:
        print(p, Y[p,s].x)
```

This gave the optimum value in part 1 of $9542.14.
This gave the optimum value in part 2 of $20,682.35.