Assignment 1 – Prescriptive Modeling with Gurobi

Problem 1: Deterministic optimization

a. The mathematical formulation

$$\min_{x_{sm}, s_m} \ 4.3x_{11} + 4.9x_{21} + 5.2x_{12} + 4.4x_{22} + 4.2x_{13} + 4.9x_{23} + 0.5(s_1 + s_2 + s_3)$$

$$\begin{aligned}
\text{s.t.}\quad & \\
& x_{sm} \leq 350 \\
& x_{11} + x_{12} + x_{13} \leq 800 \\
& x_{21} + x_{22} + x_{23} \leq 800 \\
& x_{11} + x_{21} - s_1 = 500 \\
& x_{12} + x_{22} + s_1 - s_2 = 600 \\
& x_{13} + x_{23} + s_2 - s_3 = 400 \\
& x_{sm}, s_m \geq 0, s \in \{1, 2\}, m \in \{1, 2, 3\}
\end{aligned}$$

The detail explanation is given in the coding file A1.1

b. The problem was solved in Gurobi for Python found in the coding file A1.1

c. The solution

- Optimal purchasing amounts:
    - $(x_{11}, x_{12}, x_{13}) = (350,\ 100,\ 350)$
    - $(x_{21}, x_{22}, x_{23}) = (300,\ 350,\ 50)$
- Optimal inventory amounts in each month: $(s_1, s_2, s_3) = (150,\ 0,\ 0)$
- Optimal cost: 6825.0 €

Interpretation

	Nov	**Dec**	**Jan**
Inventory amounts (litres)	150	0	0
Purchasing amounts (litres)			
From Liquor Oy	350	100	350
From Booze Oy	300	350	50

Problem 2: Stochastic optimization

d. The problem was solved in Gurobi in Python found in the coding file A1.2. The interpretation of the result are presented as following:

The investment decision

- The initial investment decision for Wind power project (W) zW0 = 1
- The initial investment decision for Geothermal power project (G) zG0 = 1
- The investment decision to continue project W in scenario 1 zW1 = 1
- The investment decision to continue project W in scenario 2 zW2 = 1
- The investment decision to continue project G in scenario 1 zG1 = 1
- The investment decision to continue project G in scenario 2 zG2 = 1

*** 0 : The project is terminated , 1 the project is continued

The evaluation outcome

- The outcome after the evaluation stage $i0S_0 = 4B$

The implementation outcome

- The outcome in scenario 1: $i0S_1 = 0.16B$
- The outcome in scenario 2: $i0S_2 = 0.16B$

The implementation result outcomes

- The outcome in scenario 11: $i0S_{11} = 20.1664B$
- The outcome in scenario 12: $i0S_{12} = 17.1664B$
- The outcome in scenario 13: $i0S_{13} = 13.1664B$
- The outcome in scenario 21: $i0S_{21} = 23.1664B$
- The outcome in scenario 22: $i0S_{22} = 13.1664B$

The expected cash flow from the project: $16.6414B$

e. The new solution for the changes in some initial investment variables:

zW0 = 1

zG0 = 1

zW1 = 1

zW2 = 0

zG1 = 1

zG2 = 1

i0SO = 5.5

i0S1 = 0.22

i0S2 = 3.72

i0S11 = 20.2288

i0S12 = 17.2288

i0S13 = 13.2288

i0S21 = 22.8688

i0S22 = 15.8688

The expected cash flow from the project: 17.3988 billion

Comparison:

- Both the solutions result in terminating the Geothermal project in scenario 1 and continuing the rest.
- The second solution delivered a higher number of expected cash flow with 17.3988 billion, compared to 16.6414 billions in the first solution.

A1.1-draft-for-students

November 2, 2022

1 Assignment 1 - parts a, b, and c

The café is finding the cost-minimising purchase plan for red wine to make glögi for the winter seaon Nov/2022 to Jan/2023. For simplicity, let's assume that for 1 litre of glögi we need 1 litre of red wine, so it does not evaporate. The supplier capacity, estimate demand, purchasing cost and inventory cost are shown in the below table.

Month	Supplier capacity (litres)	Estimated demand (litres)	Purchasing cost (EUR/litre)	Inventory cost (EUR/litre)
Nov	350	500	4.3 - 4.9	0.5
Dec	350	600	5.2 - 4.4	0.5
Jan	350	400	4.2 - 4.9	0.5

Red wine can be supplied by two companies: Liquor Oy and Booze Oy. Each company can provide a maximum of 350 litres of red wine per month and a maximum of 800 litres in total over 3 months. The red wine provided by the suppliers is interchangeable.

We model the problem by using both the purchasing quantities and inventory quantities as decision variables, and linking them through constraints. The decision variables x_{sm} describe purchasing quantities from the two suppliers and s_m as the inventory quantities each month in the season.

The problem can now be stated as

$$
\begin{aligned}
\min_{x_{sm}, s_m} \quad & 4.3x_{11} + 4.9x_{21} + 5.2x_{12} + 4.4x_{22} + 4.2x_{13} + 4.9x_{23} + 0.5(s_1 + s_2 + s_3) \\
\text{s.t.} \quad & \\
& x_{sm} \leq 350 \\
& x_{11} + x_{12} + x_{13} \leq 800 \\
& x_{21} + x_{22} + x_{23} \leq 800 \\
& x_{11} + x_{21} - s_1 = 500 \\
& x_{12} + x_{22} + s_1 - s_2 = 600 \\
& x_{13} + x_{23} + s_2 - s_3 = 400 \\
& x_{sm}, s_m \geq 0, s \in \{1, 2\}, m \in \{1, 2, 3\}
\end{aligned}
$$

[ ]:
```
# Install gurobipy package. These cell must be executed at every launch of␣
↪Google Colab.
# DO NOT DELETE OR MODIFY THIS CELL
```

```
!pip install gurobipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting gurobipy
  Downloading gurobipy-9.5.2-cp37-cp37m-manylinux2014_x86_64.whl (11.5 MB)
     |                  | 11.5 MB 7.4 MB/s
Installing collected packages: gurobipy
Successfully installed gurobipy-9.5.2
```

[ ]:
```
# Import dependencies
# DO NOT DELETE OR MODIFY THIS CELL.
import gurobipy as gp
from gurobipy import GRB
```

[ ]:
```
# Initiate the model
# model = ....

# YOUR CODE HERE
model = gp.Model("assignmentA1.1")
```

```
Restricted license - for non-production use only - expires 2023-10-25
```

[ ]:
```
# Add variables for purchase and surplus
# x = model.addVars(...)
# s = model.addVars(...)

# YOUR CODE HERE
x = model.addVars(range(1,3), range(1,4), vtype = GRB.INTEGER, name='x')
s = model.addVars(range(1,4), vtype = GRB.INTEGER, name='s')
```

[ ]:
```
# Set an objective function, don't forget to mention the type of problem
 ↪(minimization/maximization)
# model.setObjective(...)

# YOUR CODE HERE
model.setObjective(4.3*x[1,1] + 4.9*x[2,1] + 5.2*x[1,2]+ 4.4*x[2,2] + 4.
 ↪2*x[1,3] +4.9*x[2,3] + 0.5 * (s[1]+s[2]+s[3]), GRB.MINIMIZE)
```

[ ]:
```
# Add all constraints
# For each contraint use model.addConstr(...)

# YOUR CODE HERE
# The supplier capacity for each month is 350 litres
model.addConstr(x[1,1] <= 350)
model.addConstr( x[2,1] <= 350)
model.addConstr( x[1,2] <= 350)
```

```
model.addConstr( x[2,2] <= 350)
model.addConstr( x[1,3] <= 350)
model.addConstr( x[2,3] <= 350)

# The supplier capacity in the total over 3 months is 800 litres
model.addConstr( x[1,1] + x[1,2] + x[1,3] <= 800)
model.addConstr( x[2,1] + x[2,2] + x[2,3] <= 800)

# The estimate demand in each month
model.addConstr( x[1,1] + x[2,1] - s[1] == 500)
model.addConstr( x[1,2] + x[2,2] + s[1] - s[2] == 600)
model.addConstr( x[1,3] + x[2,3] + s[2] - s[3] == 400)
```

[ ]: <gurobi.Constr *Awaiting Model Update*>

[ ]:
```
# Optimize the model

# YOUR CODE HERE
model.optimize()
```

```
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 11 rows, 9 columns and 23 nonzeros
Model fingerprint: 0xd4e47164
Variable types: 0 continuous, 9 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [5e-01, 5e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+02, 8e+02]
Presolve removed 6 rows and 1 columns
Presolve time: 0.00s
Presolved: 5 rows, 8 columns, 16 nonzeros
Variable types: 0 continuous, 8 integer (0 binary)
Found heuristic solution: objective 6969.3000000
Found heuristic solution: objective 6874.2000000

Root relaxation: objective 6.825000e+03, 7 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0    6825.0000000 6825.00000  0.00%     -    0s

Explored 1 nodes (7 simplex iterations) in 0.05 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)
```

```
Solution count 3: 6825 6874.2 6969.3

Optimal solution found (tolerance 1.00e-04)
Best objective 6.825000000000e+03, best bound 6.825000000000e+03, gap 0.0000%
```

[ ]:
```python
# Get the objective value

# YOUR CODE HERE
model.ObjVal
```

[ ]:
```
6825.0
```

[ ]:
```python
# Show the values of variables related to purchase

# YOUR CODE HERE
model.getAttr('x',(x))
```

[ ]:
```
{(1, 1): 350.0,
 (1, 2): 100.0,
 (1, 3): 350.0,
 (2, 1): 300.0,
 (2, 2): 350.0,
 (2, 3): 50.0}
```

[ ]:
```python
# Show the values of variables related to surplus

# YOUR CODE HERE
model.getAttr('x', (s))
```

[ ]:
```
{1: 150.0, 2: -0.0, 3: 0.0}
```

A1_2_draft_for_students

November 2, 2022

0.1 Assignment 1, parts d) and e)

The text of the task can be found in the .pdf file with assignment description. The mathematical forumaltion of the problem will be presented here. Your task will be to implement this forumaltion in Python. At first, the whole forumlation will be given. Then, necessary parts will be shown at the respective parts of the code.

Let's denote z_0^p: initial investment in project p (1 variable per project) and z_i^p: continue project p in scenario i (2 variables per project).

If in implementation stage scenario i, $s_i, i \in 1, 2$ realizes, then in implemenation result the scenarios can be either s_1, s_2 or s_3.

Then, let's denote ioS_0 is the cash flow (remaining cash) after the first decision for each project, $ioS_k, k \in \{1, 2, 3\}$ is the total cash flow in scenario k in implementaion stage immediately after making the decision to implement or to terminate for each project and ioS_{ij} is the outcome in scenario ij after the implementation result scenario outcome is known.

The optimisation problem can now be stated as

$$
\begin{aligned}
&\max_{z_i^p} \sum_{i\in\{1,2\},j\in\{1,2,3\}} P(s_{ij}) iOS_{ij} \\
&\text{s.t.} \\
&\quad ioS_0 = 9 - 2z_0^W - 3z_0^G \\
&\quad ioS_1 = 1.04 \times ioS_0 - 2z_1^W - 2z_1^G \\
&\quad ioS_2 = 1.04 \times ioS_0 - 2z_2^W - 2z_2^G \\
&\quad ioS_{11} = 1.04 \times ioS_1 + 18z_1^W + 2z_1^G \\
&\quad ioS_{12} = 1.04 \times ioS_1 + 14z_1^W + 3z_1^G \\
&\quad ioS_{13} = 1.04 \times ioS_1 + 7z_1^W + 6z_1^G \\
&\quad ioS_{21} = 1.04 \times ioS_2 + 4z_2^W + 19z_2^G \\
&\quad ioS_{22} = 1.04 \times ioS_2 + 1.5z_2^W + 12z_2^G \\
&\quad z_0^W \geq z_1^W \\
&\quad z_0^W \geq z_2^W \\
&\quad z_0^G \geq z_1^G \\
&\quad z_0^G \geq z_2^G \\
&\quad z_i^p \in \{0,1\} \, \forall i,p \\
&\quad ioS_i \geq 0 \, \forall i
\end{aligned}
$$

[ ]:

```
# Install gurobipy package. These cell must be executed at every launch of
↪Google Colab.
# DO NOT DELETE OR MODIFY THIS CELL
!pip install gurobipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: gurobipy in /usr/local/lib/python3.7/dist-
packages (9.5.2)
```

[ ]:

```
# Import dependencies
# DO NOT DELETE OR MODIFY THIS CELL.
import gurobipy as gp
from gurobipy import GRB
```

[ ]:

```
# Initiate the model
# model = ....

# YOUR CODE HERE
model = gp.Model("assignmentA1.2")
```

[ ]:

```
# Create binary variables representing the investment decisions, six in total
# NOTE: those variables have binary type, so do not forget to specify the type
↪of the variable as GRB.BINARY
```

```
# For each variable use model.addVar(...)
# zW0 = model.addVar(...)

# YOUR CODE HERE
zW0 = model.addVar(vtype=GRB.BINARY, name="zW0") #initial investment decision␣
 ↪for project Wind(W)
zG0 = model.addVar(vtype=GRB.BINARY, name="zG0") #initical investment decision␣
 ↪for project Geothermal(G)
zW1 = model.addVar(vtype=GRB.BINARY, name="zW1") #continue project W in␣
 ↪scenario 1
zW2 = model.addVar(vtype=GRB.BINARY, name="zW2") #continue project W in␣
 ↪scenario 2
zG1 = model.addVar(vtype=GRB.BINARY, name="zG1") #continue project G in␣
 ↪scenario 1
zG2 = model.addVar(vtype=GRB.BINARY, name="zG2") #continue project G in␣
 ↪scenario 2
```

[ ]:
```
# Create real variables representing the investment outcomes, eight in total
# NOTE: those variables take real values, there is no need to specify the type␣
 ↪additionally
# For each variable use model.addVar(...)
# iOS0 = model.addVar(...)

# YOUR CODE HERE
iOS0 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS0") #cash flow (remaining␣
 ↪cash) after the first decision for each project
iOS1 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS1") #total cash flow in␣
 ↪scenario 1 in implementaion stage immediately after making the decision␣
 ↪(continue or not)
iOS2 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS2") #total cash flow in␣
 ↪scenario 2 in implementaion stage immediately after making the decision␣
 ↪(continue or not)
iOS11 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS11") #the outcome in␣
 ↪scenario 11 after the implementation result scenario outcome is known.
iOS12 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS12") #the outcome in␣
 ↪scenario 12 after the implementation result scenario outcome is known.
iOS13 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS13") #the outcome in␣
 ↪scenario 13 after the implementation result scenario outcome is known.
iOS21 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS21") #the outcome in␣
 ↪scenario 13 after the implementation result scenario outcome is known.
iOS22 = model.addVar(vtype=GRB.CONTINUOUS, name="iOS22") #the outcome in␣
 ↪scenario 13 after the implementation result scenario outcome is known.
```

[ ]:
```
# Define probabilities in the scenario tree
P1 = 0.5
P2 = 0.5
```

```
P11 = P1*0.4
P12 = P1*0.2
P13 = P1*0.4

P21 = P2*0.3
P22 = P2*0.7
```

[ ]:
```
# Define interest
r = 1.04
```

[ ]:
```
# Define cash flows for different outcomes
cfWs11 = 18
cfWs12 = 14
cfWs13 = 7

cfWs21 = 4
cfWs22 = 1.5

cfGs11 = 2
cfGs12 = 3
cfGs13 = 6

cfGs21 = 19
cfGs22 = 12
```

[ ]:
```
# Define initial investments at evaluation stage
#iEW = 2
iEW = 1.5 #for task E
#iEG = 3
iEG = 2 #for task E
# Define initial investments at imlementation stage
#iIW = 2
iIW = 3.5 #for task E
iIG = 2

# Define allocated for investments
totalInv = 9
```

Let's set the objective function

$$\max_{z_i^p} \sum_{i \in \{1,2\}, j \in \{1,2,3\}} P(s_{ij}) iOS_{ij} \tag{1}$$

$$\tag{2}$$

[ ]:
```
# Set an objective function, don't forget to mention the type of problem␣
↪(minimization/maximization)
```

```python
# model.setObjective(...)

# YOUR CODE HERE
model.setObjective(P11*iOS11 + P12*iOS12 + P13*iOS13 + P21*iOS21 + P22*iOS22,
    GRB.MAXIMIZE)
```

Let's add the constrain for the outcome in evaluation stage

$$ioS_0 = 9 - 2z_0^W - 3z_0^G$$

[ ]:
```python
# Calculate the outcome in the evaluation stage and add it as a constrain
# model.addConstr(...)

# YOUR CODE HERE
model.addConstr(iOS0 == totalInv-iEW*zW0-iEG*zG0)
```

[ ]: <gurobi.Constr *Awaiting Model Update*>

Let's add the constrain for the outcomes in implementation stage

$$\begin{aligned} ioS_1 &= 1.04 \times ioS_0 - 2z_1^W - 2z_1^G \\ ioS_2 &= 1.04 \times ioS_0 - 2z_2^W - 2z_2^G \\ ioS_i &\geq 0\,\forall i \end{aligned}$$

[ ]:
```python
# Calculate outcomes in the implementation stage and add them as constrains
# Don't forget, that the goverment can't spend more money, that it has allocated
# model.addConstr(...)

# YOUR CODE HERE
model.addConstr(iOS1 == r*iOS0 - iIW*zW1 - iIG*zG1)
model.addConstr(iOS2 == r*iOS0 - iIW*zW2 - iIG*zG2)
#model.addConstr(iOS)
```

[ ]: <gurobi.Constr *Awaiting Model Update*>

Let's add the constrain for the outcomes after implementation stage

$$\begin{aligned} ioS_{11} &= 1.04 \times ioS_1 + 18z_1^W + 2z_1^G \\ ioS_{12} &= 1.04 \times ioS_1 + 14z_1^W + 3z_1^G \\ ioS_{13} &= 1.04 \times ioS_1 + 7z_1^W + 6z_1^G \\ ioS_{21} &= 1.04 \times ioS_2 + 4z_2^W + 19z_2^G \\ ioS_{22} &= 1.04 \times ioS_2 + 1.5z_2^W + 12z_2^G \end{aligned}$$

[ ]:
```
# Calculate outcomes after the implementation stage using cash flows for
↪different scenarios and add them as constrains
# model.addConstr(...)

# YOUR CODE HERE
# Constraints for the scenario s1
model.addConstr(iOS11 == r*iOS1 + cfWs11*zW1 + cfGs11*zG1)
model.addConstr(iOS12 == r*iOS1 + cfWs12*zW1 + cfGs12*zG1)
model.addConstr(iOS13 == r*iOS1 + cfWs13*zW1 + cfGs13*zG1)
# Constraints doe the scenario s2
model.addConstr(iOS21 == r*iOS2 + cfWs21*zW2 + cfGs21*zG2)
model.addConstr(iOS22 == r*iOS2 + cfWs22*zW2 + cfGs22*zG2)
```

[ ]: <gurobi.Constr *Awaiting Model Update*>

Let's add consistency constraints for decisions

$$z_0^W \geq z_1^W$$
$$z_0^W \geq z_2^W$$
$$z_0^G \geq z_1^G$$
$$z_0^G \geq z_2^G$$

[ ]:
```
# Add consistency constraints for decisions
# model.addConstr(...)

# YOUR CODE HERE
model.addConstr(zW0 >= zW1)
model.addConstr(zW0 >= zW2)
model.addConstr(zG0 >= zG1)
model.addConstr(zG0 >= zG2)
```

[ ]: <gurobi.Constr *Awaiting Model Update*>

[ ]:
```
# Optimize the model

# YOUR CODE HERE
model.optimize()
```

```
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 12 rows, 14 columns and 39 nonzeros
Model fingerprint: 0x3f325fdb
Variable types: 8 continuous, 6 integer (6 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
```

```
  Objective range  [1e-01, 3e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [9e+00, 9e+00]
Presolve removed 12 rows and 14 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 1 (of 2 available processors)

Solution count 1: 17.3988

Optimal solution found (tolerance 1.00e-04)
Best objective 1.739880000000e+01, best bound 1.739880000000e+01, gap 0.0000%
```

[ ]:

```
# Print out solutions, first 6 are decisions, next is evaluation outcome, next␣
↪2 are implementation outcomes and last 5 are for implementation result␣
↪outcomes

# YOUR CODE HERE
for v in model.getVars():
  print('%s = %g' % (v.varName, v.x))
```

```
zW0 = 1
zG0 = 1
zW1 = 1
zW2 = 0
zG1 = 1
zG2 = 1
iOS0 = 5.5
iOS1 = 0.22
iOS2 = 3.72
iOS11 = 20.2288
iOS12 = 17.2288
iOS13 = 13.2288
iOS21 = 22.8688
iOS22 = 15.8688
```

[ ]:

```
# Print out the expected cash flow from the project (the value of objective␣
↪function). We do not take into account the time value of money.

# YOUR CODE HERE
print('Obj: %g' % model.objVal)
```

```
Obj: 17.3988
```