Daniel Alonso, UC3M UID: 100444499

Problem set # 1

Numerical Methods for Data Science 2020/21

UC3M — *Master on Statistics for Data Science*

Due date: October 19. Value: 50% of the final grade.

Note: This is an individual assignment. Evidence of plagiarism will be penalized. Hand in the assignment as a pdf file, with Gurobi-Python code in the pdf (no python files).

Problem 1 (35 points). Consider the linear optimization problem

$$\begin{aligned}
&\text{maximize } x_1 + 2x_2 - 8x_3 \\
&\text{subject to:} \\
&4x_1 + x_2 + 3x_3 = 12 \\
&x_1 + 2x_2 - 2x_3 = 8 \\
&x_1 \geqslant 0, x_2 \geqslant 0, x_3 \geqslant 0.
\end{aligned}$$

(a, 7 points) Formulate the dual problem, and find all its solutions using the graphical method.

Dual problem presented in problem 1 :

minimize $12\pi_1 + 8\pi_2$

subject to :

$x_1 : 4\pi_1 + \pi_2 \geq 1$
$x_2 : \pi_1 + 2\pi_2 \geq 2$
$x_3 : 3\pi_1 - 2\pi_2 \geq -8$

Feasable region plot

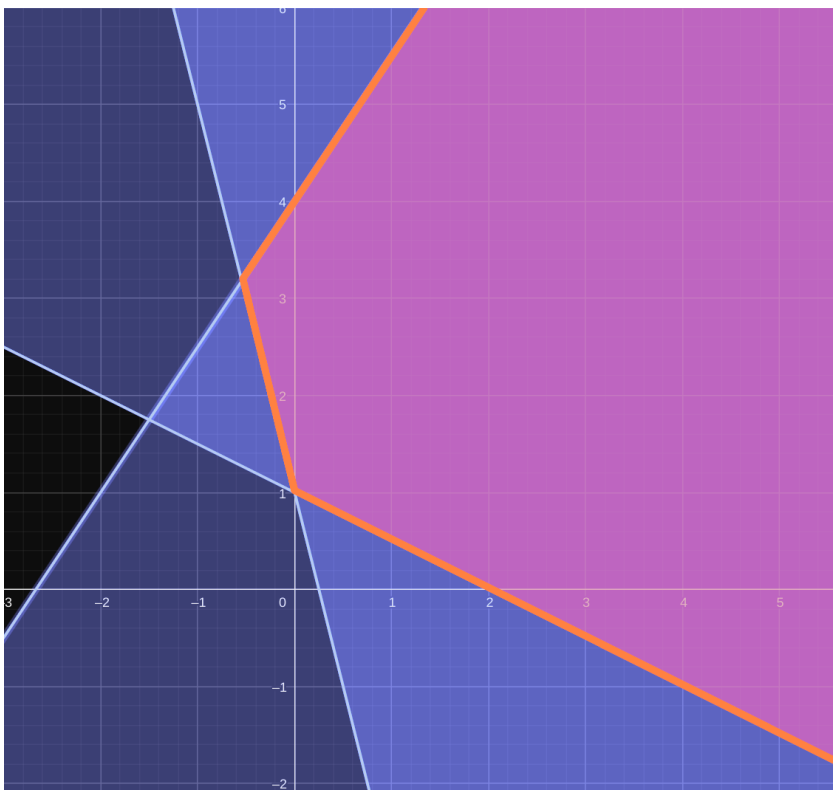

Figure 1

Feasable solutions

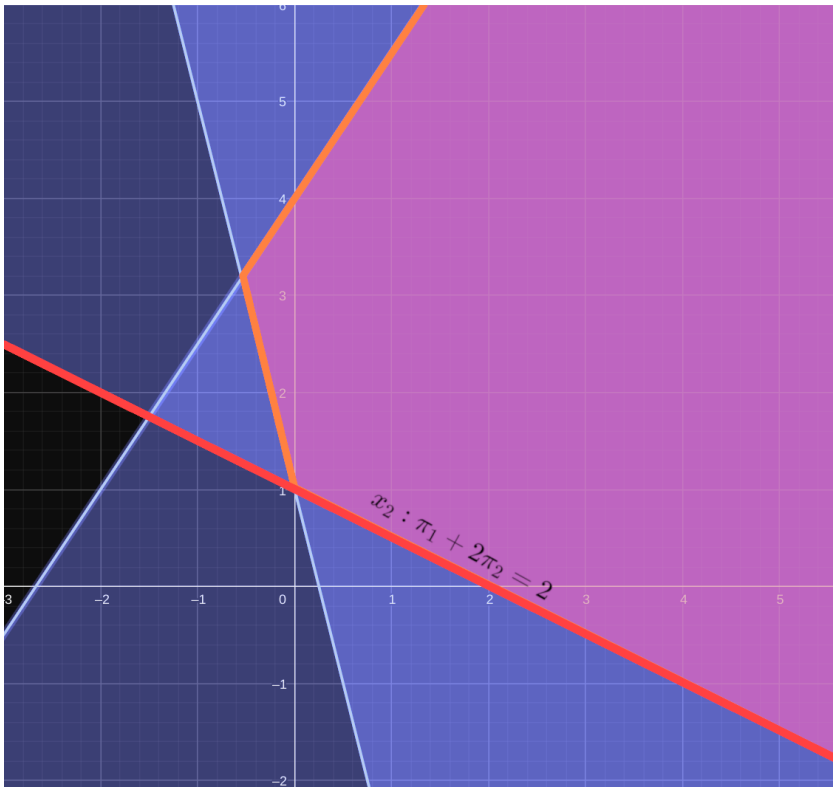

Figure 2

As we can see in Figure 2, all feasable solutions lie in the feasable region's edge $x_2 : \pi_1 + 2\pi_2 = 2$. Such line segment has vertex $(0, 1)$, which is optimal and the unique optimal solution for the dual problem:

$(\pi_1^*, \pi_2^*) = (0, 1)$

(b, 7 points) Formulate the optimality conditions that must be satisfied by any optimal primal solution in relation with a dual optimal solution $\boldsymbol{\pi}^*$. Apply them, along with part (a), to find all solutions of the primal problem.

Optimality condition 1:

$\mathbf{x}^*$ is primal feasable as it satisfies the inequalities in the primal problem

Optimality condition 2:

π^* is dual feasable as it satisfies the inequalities in the dual problem

Optimality condition 3:

Complementary slackness:

$x_3 : 3\pi_1 - 2\pi_2 \geq -8$ always holds

- therefore: $x_3 = 0$

Then we have:

$4x_1 + x_2 = 12$
$x_1 + 2x_2 = 8$
$x_1 \geq 0, x_2 \geq 0$

Solving for x_1 and x_2:

$x_2 = 12 - 4x_1$
$x_1 - 8x_1 + 24 = 8$
$-7x_1 = -16$

Which results in:

$x_1 = \frac{16}{7}$
$x_2 = \frac{20}{7}$

- therefore $\mathbf{x}^* = (\frac{16}{7}, \frac{20}{7}, 0)$

Solving the problem in Gurobi-Python

In [101]:
```
# importing libraries
import gurobipy as gp
from gurobipy import GRB

# setting up the optimization problem
m = 2  # number of resources
n = 3  # number of products

resources = range(m)  # list [1, ..., m]
products = range(n)  # list [1, ..., n]

# primal objective coefficients
r_coeff = [1, 2, -8]

# left-hand side (LHS) coefficients (matrix A)
A_coeff = [[4, 1, 3],
           [1, 2, -2]]

# right-hand side (RHS) coefficients
b_coeff = [12, 8]

r = {j: r_coeff[j] for j in products}
A = {i: {j: A_coeff[i][j] for j in products} for i in resources}
b = {i: b_coeff[i] for i in resources}

# naming the model
model = gp.Model('1d')
x = model.addVars(products, name="x")  # quantity produced

# Constraints
model.addConstrs((gp.quicksum(A[i][j] * x[j] for j in products)
                  == b[i]
                  for i in resources), name="pi")

# Objective
obj = gp.quicksum(r[j] * x[j] for j in products)
model.setObjective(obj, GRB.MAXIMIZE)
model.optimize()
```

```
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (linux64)
Optimize a model with 2 rows, 3 columns and 6 nonzeros
Model fingerprint: 0xd85cf6b9
Coefficient statistics:
  Matrix range     [1e+00, 4e+00]
  Objective range  [1e+00, 8e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+00, 1e+01]
Presolve time: 0.00s
Presolved: 2 rows, 3 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    3.0000000e+30   2.750000e+30   3.000000e+00      0s
       3    8.0000000e+00   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.00 seconds
Optimal objective  8.000000000e+00
```

In [102]:
```
# Printing solution
# Variable information including sensitivity information
varnames = [f"x[{n+1}]" for n,x in enumerate(model.getVars())]
for var, v in zip(varnames,model.getVars()):
    print(f"{var} = {v.X}")
```

```
x[1] = 2.2857142857142856
x[2] = 2.857142857142857
x[3] = 0.0
```

(c, 7 points) Obtain all possible values for the reduced cost of each primal variable. Are reduced costs unique? Interpret the reduced costs obtained.

The reduced costs of the primal variables are computed as following:

$$\bar{c_1} = 4\pi_1^* + \pi_2^* - 1 = 4\lambda + 1 - \frac{\lambda}{2} - 1 = \frac{7\lambda}{2}$$
$$\bar{c_2} = \pi_1^* + 2\pi_2^* \quad 2 = \lambda + 2 \quad \lambda \quad 2 = 0$$
$$\bar{c_3} = 3\pi_1^* \quad 2\pi_2^* + 8 = 3\lambda \quad 2 \quad \lambda + 8 = 2\lambda + 6$$

Where π^* is a dual optimal solution

- therefore, the reduced costs are not unique

(d, 7 points) Carry out a sensitivity analysis with respect to simultaneous changes of contraint right-hand sides for the primal problem. Contrast the results with those obtained through Gurobi-Python.

primal problem:

maximize $x_1 + 2x_2 - 8x_3$

subject to :

$$4x_1 + x_2 + 3x_3 = 12$$
$$x_1 + 2x_2 - 2x_3 = 8$$
$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

Let's consider $\pi^* = (\mathbf{0}, \mathbf{1})$ we have $x_3 = 0$ and then:

$$4x_1 + x_2 = 12 + \Delta b_1$$
$$x_1 + 2x_2 = 8 + \Delta b_2$$

Solving we get:

$$\hat{x}^* = (\frac{1}{7}(2\Delta b_1 - \Delta b_2 + 16), \frac{1}{7}(-\Delta b_1 + 4\Delta b_2 + 20), 0)$$

Imposing primal feasability returns:

$$2\Delta b_1 - \Delta b_2 \geq -16$$
$$-\Delta b_1 + 4\Delta b_2 \geq -20$$

As a result: if $\Delta b_1 = 0$ then $\Delta b_2 \in [-5, 16]$ and if $\Delta b_2 = 0$ then $\Delta b_1 \in [-8, 20]$

Comparing with Gurobi-Python

In [103]:
```
[x.X for x in model.getVars()]
```

Out[103]:
```
[2.2857142857142856, 2.857142857142857, 0.0]
```

In [104]:
```
# Changes in constraint RHS
constnames = [(f"pi[{n+1}]",diff) for n,(x,diff) in enumerate(zip(model.getConstrs(),[
12,8]))]
for (cname,diff),c in zip(constnames,model.getConstrs()):
    print(f"{cname} : from RHS = {c.SARHSLow-diff}, to RHS = {c.SARHSUp-diff}")
```

```
pi[1] : from RHS = -8.0, to RHS = 20.0
pi[2] : from RHS = -5.0, to RHS = 16.0
```

As we can see, te Gurobi-Python solution matches both the intervals resulting from the analytical process.

(e, 7 points) Carry out a sensitivity analysis with respect to simultaneous changes of objective coefficients for the primal problem. Contrast the results with those obtained through Gurobi-Python.

The primal problem only has one feasable solution, so the optimal solution $\mathbf{x^*}$ remains optimal for the following modified problem:

maximize $(1+\Delta r_1)x_1 + (2+\Delta r_2)x_2 - (8+\Delta r_3)x_3$

subject to :

$4x_1 + x_2 + 3x_3 = 12$
$x_1 + 2x_2 - 2x_3 = 8$
$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$

Following a standard analysis, we consider $\mathbf{\hat{x}^*} = (\frac{16}{7}, \frac{20}{7}, 0)$

By complementary slackness we formulate the following 2 systems of linear equations:

system 1:

$4\pi_1 + \pi_2 = 1 + \Delta r_1$
$\pi_1 + 2\pi_2 = 2 + \Delta r_2$

resulting in:

$\hat{\pi}^* = (\frac{1}{7}(2\Delta r_1 - \Delta r_2), \frac{1}{7}(7 - \Delta r_1 + 4\Delta b_2))$

Dual feasability means that:

$3\hat{\pi}_1^* + 2\hat{\pi}_2^* + 8 - \Delta r_3 = \frac{1}{7}(42 + 8\Delta r_1 - 11\Delta r_2 - 7\Delta r_3)$

and:

$8\Delta r_1 - 11\Delta r_2 - 7\Delta r_3 \geq -42$

So, if:

$\Delta r_1 = 0, \Delta r_2 = 0$, then $\Delta r_3 \leq 6$

$\Delta r_1 = 0, \Delta r_3 = 0$, then $\Delta r_2 \leq \frac{42}{11}$

$\Delta r_2 = 0, \Delta r_3 = 0$, then $\Delta r_1 \geq \frac{-21}{4}$

system 2:

$\pi_1 + 2\pi_2 = 2 + \Delta r_2$
$3\pi_1 - 2\pi_2 = -8 + \Delta r_3$

resulting in:

$\hat{\pi}^* = (\frac{1}{4}(-14 - 3\Delta r_2 + \Delta r_3), \frac{1}{4}(-20 - 2\frac{1}{2}\Delta r_2 + 2\frac{1}{2}\Delta r_3))$

Dual feasability means that:

$4\hat{\pi}_1^* + \hat{\pi}_2^* - 1 - \Delta r_1 = \frac{1}{4}(-98 - 4\Delta r_1 - 11\Delta r_2 + 9\Delta r_3)$

and:

$-4\Delta r_1 - 11\Delta r_2 + 9\Delta r_3 \geq 98$

So, if:

$\Delta r_1 = 0, \Delta r_2 = 0$, then $\Delta r_3 \geq \frac{98}{9}$

$\Delta r_1 = 0, \Delta r_3 = 0$, then $\Delta r_2 \leq \frac{98}{11}$

$\Delta r_2 = 0, \Delta r_3 = 0$, then $\Delta r_1 \leq \frac{-49}{2}$

Combining both cases we obtain that $\mathbf{x}^*$ is optimal for any $\Delta r_j \in \mathbb{R}$

Comparing with Gurobi-Python

In [100]:
```
# Changes of objective coefficients for the primal problem
varnames = [(f"pi[{n+1}]",diff) for n,(x,diff) in enumerate(zip(model.getVars(),[1,2,-
8]))]
for (var,diff), v in zip(varnames,model.getVars()):
    print(f"{var} : from coeff = {v.SAObjLow - diff}, to coeff = {v.SAObjUp - diff}")
```

```
pi[1] : from coeff = -5.25, to coeff = inf
pi[2] : from coeff = -inf, to coeff = 3.8181818181818183
pi[3] : from coeff = -inf, to coeff = 6.0
```

As we can see, the Gurobi-Python solution matches the first 3 differences

Problem 2 (35 points). In a CSI investigation, the crime suspect left his/her shoe imprints in the crime scene. From that evidence the investigators want to infer the suspect's height. For that purpose, they plan to obtain a prediction equation for height based on shoe size based on the following data:

Shoe size (cm)	Height (cm)
30.1	176.2
29.5	176.8
30.4	184.2
31.6	173.2
27.4	172.8
28.3	174.1
33.4	180.5

(a, 20 points) Formulate the Linear Optimization model seen in class for estimating the best prediction equation under the Mean Absolute Error (MAE) criterion, and implement it in Gurobi-Python.

The primal LO formulation for LR with MAE criterion as seen in class has the following parameters:

- Objective: minimize $\sum_{i=1}^{n}(e_i^+ + e_i\ )$
- Subject to the following constraints:
  - $(e_i^+ - e_i\ ) + b_0 + x_{i1}b_1 + \cdots + x_{im}b_m = y_i\ \ i = 1, \ldots, n$
  - $e_i^+, e_i^- \geq 0, i = 1, \ldots, n$

Gurobi implementation of the problem

In [127]:
```
# importing libraries
import gurobipy as gp
from gurobipy import GRB
```

In [128]:
```
# creating multidict with observations
observations, x, y = gp.multidict({
    ('1'): [30.1,176.2],
    ('2'): [29.5,176.8],
    ('3'): [30.4,184.2],
    ('4'): [31.6,173.2],
    ('5'): [27.4,172.8],
    ('6'): [28.3,174.1],
    ('7'): [33.4,180.5],
})
```

In [129]:
```
model = gp.Model('CurveFitting')

# Constant term of the function f(x). This is a free continuous variable that can take
positive and negative values.
a = model.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="a")

# Coefficient of the linear term of the function f(x). This is a free continuous varia
ble that can take positive
# and negative values.
b = model.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="b")

# Non-negative continuous variables that capture the positive deviations
e_plus = model.addVars(observations, vtype=GRB.CONTINUOUS, name="e_plus")

# Non-negative continuous variables that capture the negative deviations
e_neg = model.addVars(observations, vtype=GRB.CONTINUOUS, name="e_neg")

# Non-negative continuous variables that capture the value of the maximum deviation
z = model.addVar(vtype=GRB.CONTINUOUS, name="z")
```

In [130]:
```
# Deviation constraints

deviations = model.addConstrs( (b*x[i] + a + e_plus[i] - e_neg[i] == y[i] for i in obs
ervations), name='deviations')

# Objective function of problem 1

model.setObjective(e_plus.sum('*') + e_neg.sum('*'))

# Verify model formulation

model.write('CurveFitting.lp')

# Run optimization engine

model.optimize()
```

```
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (linux64)
Optimize a model with 7 rows, 17 columns and 28 nonzeros
Model fingerprint: 0xc72524f5
Coefficient statistics:
  Matrix range     [1e+00, 3e+01]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 2e+02]
Presolve removed 0 rows and 1 columns
Presolve time: 0.04s
Presolved: 7 rows, 16 columns, 28 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                         0s
       7    1.4029412e+01   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.04 seconds
Optimal objective  1.402941176e+01
```

In [131]:
```
# The best straight line that minimizes the absolute value of the deviations is:
print(f"y = {b.x:.4f}x + ({a.x:.4f})")
```

```
y = 1.2549x + (138.5863)
```

In [132]:
```
for n,v in enumerate(model.getVars()):
    print(f"{v.Varname} = {v.X}, reduced cost = {abs(v.RC):.2f}, from coeff = {v.SAObj
Low:.2f}, to coeff = {v.SAObjUp:.2f}")
```

```
a = 138.58627450980384, reduced cost = 0.00, from coeff = -0.15, to coeff = 0.03
b = 1.2549019607843164, reduced cost = 0.00, from coeff = -0.90, to coeff = 4.20
e_plus[1] = 0.0, reduced cost = 2.00, from coeff = -1.00, to coeff = inf
e_plus[2] = 1.1941176470588317, reduced cost = 0.00, from coeff = 0.77, to coeff = 3.38
e_plus[3] = 7.464705882352931, reduced cost = 0.00, from coeff = 0.70, to coeff = 3.86
e_plus[4] = 0.0, reduced cost = 2.00, from coeff = -1.00, to coeff = inf
e_plus[5] = 0.0, reduced cost = 2.00, from coeff = -1.00, to coeff = inf
e_plus[6] = 0.0, reduced cost = 0.18, from coeff = 0.82, to coeff = inf
e_plus[7] = 0.0, reduced cost = 0.82, from coeff = 0.18, to coeff = inf
e_neg[1] = 0.15882352941177658, reduced cost = 0.00, from coeff = -1.00, to coeff = 1.2
7
e_neg[2] = 0.0, reduced cost = 2.00, from coeff = -1.00, to coeff = inf
e_neg[3] = 0.0, reduced cost = 2.00, from coeff = -1.00, to coeff = inf
e_neg[4] = 5.041176470588255, reduced cost = 0.00, from coeff = -0.82, to coeff = 1.50
e_neg[5] = 0.1705882352940975, reduced cost = 0.00, from coeff = -0.55, to coeff = 1.15
e_neg[6] = 0.0, reduced cost = 1.82, from coeff = -0.82, to coeff = inf
e_neg[7] = 0.0, reduced cost = 1.18, from coeff = -0.18, to coeff = inf
z = 0.0, reduced cost = 0.00, from coeff = 0.00, to coeff = inf
```

In [136]:
```
# Optimal shadow prices
for n,c in enumerate(model.getConstrs()):
    print(f"{c.ConstrName} : shadow price = {c.Pi}, from RHS = {c.SARHSLow}, to RHS =
{c.SARHSUp:.2f}")
```

```
deviations[1] : shadow price = -1.0, from RHS = -inf, to RHS = 176.36
deviations[2] : shadow price = 1.0, from RHS = 175.6058823529412, to RHS = inf
deviations[3] : shadow price = 1.0, from RHS = 176.73529411764707, to RHS = inf
deviations[4] : shadow price = -1.0, from RHS = -inf, to RHS = 178.24
deviations[5] : shadow price = -1.0, from RHS = -inf, to RHS = 172.97
deviations[6] : shadow price = 0.8235294117647058, from RHS = 173.955, to RHS = 175.66
deviations[7] : shadow price = 0.17647058823529418, from RHS = 180.04999999999995, to R
HS = 181.47
```

(b, 7 points) Solve the model and give the optimal solution (prediction equation). Is it unique?

The prediction equation (representing the optimal solution) resulting from the implementation is $\hat{y} = 1.2549x + 138.5863$.

In this case, the variables that have to take the value zero in any optimal solution because they have positive reduced costs are:

$$e_1^+ = e_4^+ = e_5^+ = e_6^+ = e_7^+ = e_2^- = e_3^- = e_6^- = e_7^- = 0$$

We thus have the following system of equations that characterizes the optimal solutions:

$$\begin{aligned}
-e_1^- + b_0 + x_1 b_1 &= y_1 \\
e_2^+ + b_0 + x_2 b_1 &= y_2 \\
e_3^+ + b_0 + x_3 b_1 &= y_3 \\
-e_4^- + b_0 + x_4 b_1 &= y_4 \\
-e_5^- + b_0 + x_5 b_1 &= y_5 \\
b_0 + x_6 b_1 &= y_6 \\
b_0 + x_7 b_1 &= y_7
\end{aligned}$$

Given that this system has a unique solution, then there is a unique optimal solution.

(c, 8 points) Obtain the optimal dual solution and discuss its possible interpretation.

The dual problem is

(D) maximize $d = \sum_{i=1}^{n} y_i \pi_i$

subject to:

$b_0 : \sum_{i=1}^{n} \pi_i = 0$
$b_1 : \sum_{i=1}^{n} x_i \pi_i = 0$
$e_i^+ : \pi_i \leq 1, i = 1, \ldots, n$
$e_i^- : \quad \pi_i \leq 1, i = 1, \ldots, n$

by strong duality, we write the MAE as $z^*(\mathbf{y})$, so then we have

$\pi_1^* = \frac{\partial}{\partial y_i} z^*(\mathbf{y}),$

whenever the derivative exists.

therefore we have the following optimal dual solution:

$\pi_1^* = -1$
$\pi_2^* = 1$
$\pi_3^* = 1$
$\pi_4^* = -1$
$\pi_5^* = -1$
$\pi_6^* = 0.8235$
$\pi_7^* = 0.1764$

We can see that every time the prediction equation underestimates the response (meaning $e_i^+ > 0$), our solution for such π_i^* will be 1. For the opposite case, when it overestimates the response (meaning $e_i \ > 0$), our solution for such π_i^* will be -1.

In [138]:
```
# Optimal shadow prices
for n,c in enumerate(model.getConstrs()):
    print(f"{c.ConstrName} : shadow price = {c.Pi}, from RHS = {c.SARHSLow}, to RHS =
{c.SARHSUp:.2f}")
    if c.Pi == float(-1):
        print("Prediction equation overestimates the response\n")
    elif c.Pi == float(1):
        print("Prediction equation underestimates the response\n")
    else:
        print("Response is within range\n")
```

```
deviations[1] : shadow price = -1.0, from RHS = -inf, to RHS = 176.36
Prediction equation overestimates the response

deviations[2] : shadow price = 1.0, from RHS = 175.6058823529412, to RHS = inf
Prediction equation underestimates the response

deviations[3] : shadow price = 1.0, from RHS = 176.73529411764707, to RHS = inf
Prediction equation underestimates the response

deviations[4] : shadow price = -1.0, from RHS = -inf, to RHS = 178.24
Prediction equation overestimates the response

deviations[5] : shadow price = -1.0, from RHS = -inf, to RHS = 172.97
Prediction equation overestimates the response

deviations[6] : shadow price = 0.8235294117647058, from RHS = 173.955, to RHS = 175.66
Response is within range

deviations[7] : shadow price = 0.17647058823529418, from RHS = 180.04999999999995, to R
HS = 181.47
Response is within range
```

Problem 3 (30 points). Consider the food manufacture I Jupiter notebook modeling example seen in class and available in Aula Global.

(a, 10 puntos) Modify the code so that it generates sensitivity analysis information (reduced costs, shadow prices and corresponding ranges of validity).

In [41]:
```
# importing libraries
import numpy as np
import pandas as pd
import gurobipy as gp
from gurobipy import GRB
```

In [42]:
```
# Parameters

months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]

oils = ["VEG1", "VEG2", "OIL1", "OIL2", "OIL3"]

cost = {
    ('Jan', 'VEG1'): 110,
    ('Jan', 'VEG2'): 120,
    ('Jan', 'OIL1'): 130,
    ('Jan', 'OIL2'): 110,
    ('Jan', 'OIL3'): 115,
    ('Feb', 'VEG1'): 130,
    ('Feb', 'VEG2'): 130,
    ('Feb', 'OIL1'): 110,
    ('Feb', 'OIL2'): 90,
    ('Feb', 'OIL3'): 115,
    ('Mar', 'VEG1'): 110,
    ('Mar', 'VEG2'): 140,
    ('Mar', 'OIL1'): 130,
    ('Mar', 'OIL2'): 100,
    ('Mar', 'OIL3'): 95,
    ('Apr', 'VEG1'): 120,
    ('Apr', 'VEG2'): 110,
    ('Apr', 'OIL1'): 120,
    ('Apr', 'OIL2'): 120,
    ('Apr', 'OIL3'): 125,
    ('May', 'VEG1'): 100,
    ('May', 'VEG2'): 120,
    ('May', 'OIL1'): 150,
    ('May', 'OIL2'): 110,
    ('May', 'OIL3'): 105,
    ('Jun', 'VEG1'): 90,
    ('Jun', 'VEG2'): 100,
    ('Jun', 'OIL1'): 140,
    ('Jun', 'OIL2'): 80,
    ('Jun', 'OIL3'): 135
}


hardness = {"VEG1": 8.8, "VEG2": 6.1, "OIL1": 2.0, "OIL2": 4.2, "OIL3": 5.0}

price = 150
init_store = 500
target_store = 500
veg_cap = 200
oil_cap = 250

min_hardness = 3
max_hardness = 6
holding_cost = 5
```

In [43]:
```
# Model deployment

food = gp.Model('Food Manufacture I')
# Quantity of food produced in each period
produce = food.addVars(months, name="Produce")
# Quantity bought of each product in each period
buy = food.addVars(months, oils, name = "Buy")
# Quantity used of each product  in each period
consume = food.addVars(months, oils, name = "Use")
# Quantity stored of each product  in each period
store = food.addVars(months, oils, name = "Store")
```

In [44]:
```
#1. Initial Balance
Balance0 = food.addConstrs((init_store + buy[months[0], oil]
                == consume[months[0], oil] + store[months[0], oil]
                for oil in oils), "Initial_Balance")

#2. Balance
Balance = food.addConstrs((store[months[months.index(month)-1], oil] + buy[month, oil]
                == consume[month, oil] + store[month, oil]
                for oil in oils for month in months if month != month[0]), "Balance")

#3. Inventory Target
TargetInv = food.addConstrs((store[months[-1], oil] == target_store for oil in oils),
"End_Balance")

#4.1 Vegetable Oil Capacity
VegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for oil in oils if "VE
G" in oil)
                <= veg_cap for month in months), "Capacity_Veg")

#4.2 Non-vegetable Oil Capacity
NonVegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for oil in oils if
"OIL" in oil)
                <= oil_cap for month in months), "Capacity_Oil")

#5. Hardness
HardnessMin = food.addConstrs((gp.quicksum(hardness[oil]*consume[month, oil] for oil i
n oils)
                >= min_hardness*produce[month] for month in months), "Hardness_lower"
)
HardnessMax = food.addConstrs((gp.quicksum(hardness[oil]*consume[month, oil] for oil i
n oils)
                <= max_hardness*produce[month] for month in months), "Hardness_upper"
)

#6. Mass Conservation
MassConservation = food.addConstrs((consume.sum(month) == produce[month] for month in
months), "Mass_conservation")

#0. Objective Function
obj = price*produce.sum() - buy.prod(cost) - holding_cost*store.sum()
food.setObjective(obj, GRB.MAXIMIZE) # maximize profit
```

In [45]:
```
food.optimize()
```

```
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (linux64)
Optimize a model with 70 rows, 96 columns and 278 nonzeros
Model fingerprint: 0xd588eb19
Coefficient statistics:
  Matrix range     [1e+00, 9e+00]
  Objective range  [5e+00, 2e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 5e+02]
Presolve removed 33 rows and 45 columns
Presolve time: 0.00s
Presolved: 37 rows, 51 columns, 149 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    3.7375000e+05   1.703125e+03   0.000000e+00      0s
      32    1.0784259e+05   0.000000e+00   0.000000e+00      0s

Solved in 32 iterations and 0.00 seconds
Optimal objective  1.078425926e+05
```

In [23]:
```
# Variable information including sensitivity information
for n,v in enumerate(food.getVars()):
    if n>1 and food.getVars()[n-1].Varname[0:3] != v.Varname[0:3]:
        print('-----------------------------------------------')
    print(f"{v.Varname} = {v.X}, reduced cost = {abs(v.RC):.2f}, from coeff = {v.SAObj
Low:.2f}, to coeff = {v.SAObjUp:.2f}")
```

```
Produce[Jan] = 450.0, reduced cost = 0.00, from coeff = 65.00, to coeff = inf
Produce[Feb] = 450.0, reduced cost = 0.00, from coeff = 75.00, to coeff = inf
Produce[Mar] = 450.0, reduced cost = 0.00, from coeff = 85.00, to coeff = inf
Produce[Apr] = 450.0, reduced cost = 0.00, from coeff = 95.00, to coeff = inf
Produce[May] = 450.0, reduced cost = 0.00, from coeff = 105.00, to coeff = inf
Produce[Jun] = 450.0, reduced cost = 0.00, from coeff = 100.37, to coeff = inf
-----------------------------------------------
Buy[Jan,VEG1] = 0.0, reduced cost = 60.00, from coeff = -inf, to coeff = -50.00
Buy[Jan,VEG2] = 0.0, reduced cost = 70.00, from coeff = -inf, to coeff = -50.00
Buy[Jan,OIL1] = 0.0, reduced cost = 65.00, from coeff = -inf, to coeff = -65.00
Buy[Jan,OIL2] = 0.0, reduced cost = 45.00, from coeff = -inf, to coeff = -65.00
Buy[Jan,OIL3] = 0.0, reduced cost = 50.00, from coeff = -inf, to coeff = -65.00
Buy[Feb,VEG1] = 0.0, reduced cost = 70.00, from coeff = -inf, to coeff = -60.00
Buy[Feb,VEG2] = 0.0, reduced cost = 70.00, from coeff = -inf, to coeff = -60.00
Buy[Feb,OIL1] = 0.0, reduced cost = 35.00, from coeff = -inf, to coeff = -75.00
Buy[Feb,OIL2] = 0.0, reduced cost = 15.00, from coeff = -inf, to coeff = -75.00
Buy[Feb,OIL3] = 0.0, reduced cost = 40.00, from coeff = -inf, to coeff = -75.00
Buy[Mar,VEG1] = 0.0, reduced cost = 40.00, from coeff = -inf, to coeff = -70.00
Buy[Mar,VEG2] = 0.0, reduced cost = 70.00, from coeff = -inf, to coeff = -70.00
Buy[Mar,OIL1] = 0.0, reduced cost = 45.00, from coeff = -inf, to coeff = -85.00
Buy[Mar,OIL2] = 0.0, reduced cost = 15.00, from coeff = -inf, to coeff = -85.00
Buy[Mar,OIL3] = 0.0, reduced cost = 10.00, from coeff = -inf, to coeff = -85.00
Buy[Apr,VEG1] = 0.0, reduced cost = 40.00, from coeff = -inf, to coeff = -80.00
Buy[Apr,VEG2] = 0.0, reduced cost = 30.00, from coeff = -inf, to coeff = -80.00
Buy[Apr,OIL1] = 0.0, reduced cost = 25.00, from coeff = -inf, to coeff = -95.00
Buy[Apr,OIL2] = 0.0, reduced cost = 25.00, from coeff = -inf, to coeff = -95.00
Buy[Apr,OIL3] = 0.0, reduced cost = 30.00, from coeff = -inf, to coeff = -95.00
Buy[May,VEG1] = 0.0, reduced cost = 10.00, from coeff = -inf, to coeff = -90.00
Buy[May,VEG2] = 0.0, reduced cost = 30.00, from coeff = -inf, to coeff = -90.00
Buy[May,OIL1] = 0.0, reduced cost = 45.00, from coeff = -inf, to coeff = -105.00
Buy[May,OIL2] = 0.0, reduced cost = 5.00, from coeff = -inf, to coeff = -105.00
Buy[May,OIL3] = 750.0, reduced cost = 0.00, from coeff = -110.00, to coeff = -78.15
Buy[Jun,VEG1] = 659.2592592592592, reduced cost = 0.00, from coeff = -100.00, to coeff
= -57.05
Buy[Jun,VEG2] = 540.7407407407408, reduced cost = 0.00, from coeff = -110.00, to coeff
= -90.00
Buy[Jun,OIL1] = 0.0, reduced cost = 25.00, from coeff = -inf, to coeff = -115.00
Buy[Jun,OIL2] = 750.0, reduced cost = 0.00, from coeff = -106.85, to coeff = inf
Buy[Jun,OIL3] = 0.0, reduced cost = 20.00, from coeff = -inf, to coeff = -115.00
-----------------------------------------------
Use[Jan,VEG1] = 159.25925925925924, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Jan,VEG2] = 40.74074074074077, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Jan,OIL1] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Jan,OIL2] = 250.0, reduced cost = 0.00, from coeff = -0.00, to coeff = inf
Use[Jan,OIL3] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Feb,VEG1] = 85.18518518518519, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Feb,VEG2] = 114.81481481481481, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Feb,OIL1] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Feb,OIL2] = 0.0, reduced cost = 0.00, from coeff = -0.00, to coeff = 0.00
Use[Feb,OIL3] = 250.0, reduced cost = 0.00, from coeff = -0.00, to coeff = 0.00
Use[Mar,VEG1] = 85.18518518518519, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Mar,VEG2] = 114.81481481481481, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Mar,OIL1] = 0.0, reduced cost = 0.00, from coeff = -0.00, to coeff = 25.00
Use[Mar,OIL2] = 250.0, reduced cost = 0.00, from coeff = -0.00, to coeff = 0.00
Use[Mar,OIL3] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Apr,VEG1] = 85.18518518518519, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
```

```
Use[Apr,VEG2] = 114.81481481481481, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[Apr,OIL1] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Apr,OIL2] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[Apr,OIL3] = 250.0, reduced cost = 0.00, from coeff = -0.00, to coeff = inf
Use[May,VEG1] = 85.18518518518519, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[May,VEG2] = 114.81481481481481, reduced cost = 0.00, from coeff = -0.00, to coeff =
0.00
Use[May,OIL1] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = 0.00
Use[May,OIL2] = 0.0, reduced cost = 0.00, from coeff = -0.00, to coeff = 5.00
Use[May,OIL3] = 250.0, reduced cost = 0.00, from coeff = -5.00, to coeff = 0.00
Use[Jun,VEG1] = 159.25925925925924, reduced cost = 0.00, from coeff = -10.00, to coeff
= 32.95
Use[Jun,VEG2] = 40.74074074074077, reduced cost = 0.00, from coeff = -32.95, to coeff =
10.00
Use[Jun,OIL1] = 0.0, reduced cost = 26.85, from coeff = -inf, to coeff = 26.85
Use[Jun,OIL2] = 250.0, reduced cost = 0.00, from coeff = -26.85, to coeff = inf
Use[Jun,OIL3] = 0.0, reduced cost = 37.96, from coeff = -inf, to coeff = 37.96
---------------------------------------------
Store[Jan,VEG1] = 340.74074074074076, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Jan,VEG2] = 459.25925925925924, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Jan,OIL1] = 500.0, reduced cost = 0.00, from coeff = -10.00, to coeff = 55.00
Store[Jan,OIL2] = 250.0, reduced cost = 0.00, from coeff = -inf, to coeff = -10.00
Store[Jan,OIL3] = 500.0, reduced cost = 0.00, from coeff = -10.00, to coeff = 40.00
Store[Feb,VEG1] = 255.55555555555557, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Feb,VEG2] = 344.44444444444446, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Feb,OIL1] = 500.0, reduced cost = 0.00, from coeff = -10.00, to coeff = 25.00
Store[Feb,OIL2] = 250.0, reduced cost = 0.00, from coeff = -10.00, to coeff = -10.00
Store[Feb,OIL3] = 250.0, reduced cost = 0.00, from coeff = -10.00, to coeff = -10.00
Store[Mar,VEG1] = 170.37037037037038, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Mar,VEG2] = 229.62962962962965, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Mar,OIL1] = 500.0, reduced cost = 0.00, from coeff = -35.00, to coeff = -10.00
Store[Mar,OIL2] = 0.0, reduced cost = 0.00, from coeff = -10.00, to coeff = -10.00
Store[Mar,OIL3] = 250.0, reduced cost = 0.00, from coeff = -10.00, to coeff = -10.00
Store[Apr,VEG1] = 85.18518518518519, reduced cost = 0.00, from coeff = -10.00, to coeff
= -10.00
Store[Apr,VEG2] = 114.81481481481484, reduced cost = 0.00, from coeff = -10.00, to coef
f = -10.00
Store[Apr,OIL1] = 500.0, reduced cost = 0.00, from coeff = -35.00, to coeff = -10.00
Store[Apr,OIL2] = 0.0, reduced cost = 0.00, from coeff = -10.00, to coeff = -0.00
Store[Apr,OIL3] = 0.0, reduced cost = 0.00, from coeff = -inf, to coeff = -10.00
Store[May,VEG1] = 0.0, reduced cost = 10.00, from coeff = -inf, to coeff = -0.00
Store[May,VEG2] = 0.0, reduced cost = 0.00, from coeff = -20.00, to coeff = -0.00
Store[May,OIL1] = 500.0, reduced cost = 0.00, from coeff = -35.00, to coeff = 16.85
Store[May,OIL2] = 0.0, reduced cost = 35.00, from coeff = -inf, to coeff = 25.00
Store[May,OIL3] = 500.0, reduced cost = 0.00, from coeff = -30.00, to coeff = 27.96
Store[Jun,VEG1] = 500.0, reduced cost = 0.00, from coeff = -inf, to coeff = inf
Store[Jun,VEG2] = 500.0, reduced cost = 0.00, from coeff = -inf, to coeff = inf
Store[Jun,OIL1] = 500.0, reduced cost = 0.00, from coeff = -inf, to coeff = inf
Store[Jun,OIL2] = 500.0, reduced cost = 0.00, from coeff = -inf, to coeff = inf
Store[Jun,OIL3] = 500.0, reduced cost = 0.00, from coeff = -inf, to coeff = inf
```

In [7]:
```
# Optimal objective value
print(f"{food.objVal:.3f}")
```

```
107842.593
```

In [8]:
```
# Optimal shadow prices
for n,c in enumerate(food.getConstrs()):
    if n>1 and food.getConstrs()[n-1].ConstrName[0:3] != c.ConstrName[0:3]:
        print('-----------------------------------------------')
    print(f"{c.ConstrName} : shadow price = {c.Pi}, from RHS = {c.SARHSLow}, to RHS =
{c.SARHSUp:.2f}")
```

```
Initial_Balance[VEG1] : shadow price = 0.0, from RHS = -500.0, to RHS = -500.00
Initial_Balance[VEG2] : shadow price = 0.0, from RHS = -500.0, to RHS = -500.00
Initial_Balance[OIL1] : shadow price = 0.0, from RHS = -500.0, to RHS = -500.00
Initial_Balance[OIL2] : shadow price = 0.0, from RHS = -500.0, to RHS = -500.00
Initial_Balance[OIL3] : shadow price = 0.0, from RHS = -500.0, to RHS = -500.00
-----------------------------------------------
Balance[VEG1,Jan] : shadow price = -75.0, from RHS = 0.0, to RHS = 0.00
Balance[VEG1,Feb] : shadow price = -80.0, from RHS = -62.96296296296294, to RHS = 0.00
Balance[VEG1,Mar] : shadow price = -85.0, from RHS = -62.96296296296294, to RHS = 0.00
Balance[VEG1,Apr] : shadow price = -90.0, from RHS = -62.96296296296294, to RHS = 0.00
Balance[VEG1,May] : shadow price = -95.0, from RHS = -62.96296296296294, to RHS = 0.00
Balance[VEG1,Jun] : shadow price = -90.0, from RHS = -659.2592592592592, to RHS = inf
Balance[VEG2,Jan] : shadow price = -75.0, from RHS = 0.0, to RHS = 0.00
Balance[VEG2,Feb] : shadow price = -80.0, from RHS = -540.7407407407408, to RHS = 0.00
Balance[VEG2,Mar] : shadow price = -85.0, from RHS = -540.7407407407408, to RHS = 0.00
Balance[VEG2,Apr] : shadow price = -90.0, from RHS = -540.7407407407408, to RHS = 0.00
Balance[VEG2,May] : shadow price = -95.0, from RHS = -540.7407407407408, to RHS = 0.00
Balance[VEG2,Jun] : shadow price = -100.0, from RHS = -540.7407407407408, to RHS = inf
Balance[OIL1,Jan] : shadow price = -85.0, from RHS = 0.0, to RHS = 0.00
Balance[OIL1,Feb] : shadow price = -90.0, from RHS = -50.00000000000005, to RHS = 0.00
Balance[OIL1,Mar] : shadow price = -95.0, from RHS = -50.00000000000005, to RHS = 0.00
Balance[OIL1,Apr] : shadow price = -100.0, from RHS = -50.00000000000005, to RHS = 0.00
Balance[OIL1,May] : shadow price = -105.0, from RHS = -50.00000000000005, to RHS = 0.00
Balance[OIL1,Jun] : shadow price = -110.0, from RHS = -50.00000000000005, to RHS = 0.00
Balance[OIL2,Jan] : shadow price = -85.0, from RHS = 0.0, to RHS = 0.00
Balance[OIL2,Feb] : shadow price = -90.0, from RHS = -250.0, to RHS = inf
Balance[OIL2,Mar] : shadow price = -95.0, from RHS = -250.0, to RHS = inf
Balance[OIL2,Apr] : shadow price = -100.0, from RHS = -250.0, to RHS = inf
Balance[OIL2,May] : shadow price = -105.0, from RHS = -250.0, to RHS = inf
Balance[OIL2,Jun] : shadow price = -80.0, from RHS = -750.0, to RHS = inf
Balance[OIL3,Jan] : shadow price = -85.0, from RHS = 0.0, to RHS = 0.00
Balance[OIL3,Feb] : shadow price = -90.0, from RHS = -500.0, to RHS = 0.00
Balance[OIL3,Mar] : shadow price = -95.0, from RHS = -500.0, to RHS = 0.00
Balance[OIL3,Apr] : shadow price = -100.0, from RHS = -500.0, to RHS = 0.00
Balance[OIL3,May] : shadow price = -105.0, from RHS = -500.0, to RHS = inf
Balance[OIL3,Jun] : shadow price = -110.0, from RHS = -500.0, to RHS = inf
-----------------------------------------------
End_Balance[VEG1] : shadow price = -20.0, from RHS = 500.0, to RHS = 500.00
End_Balance[VEG2] : shadow price = -30.0, from RHS = 500.0, to RHS = 500.00
End_Balance[OIL1] : shadow price = -30.0, from RHS = 500.0, to RHS = 500.00
End_Balance[OIL2] : shadow price = 0.0, from RHS = 500.0, to RHS = 500.00
End_Balance[OIL3] : shadow price = -30.0, from RHS = 500.0, to RHS = 500.00
-----------------------------------------------
Capacity_Veg[Jan] : shadow price = 75.0, from RHS = 0.0, to RHS = 200.00
Capacity_Veg[Feb] : shadow price = 70.0, from RHS = 89.28571428571428, to RHS = 200.00
Capacity_Veg[Mar] : shadow price = 65.0, from RHS = 96.2962962962963, to RHS = 200.00
Capacity_Veg[Apr] : shadow price = 60.0, from RHS = 160.7142857142857, to RHS = 200.00
Capacity_Veg[May] : shadow price = 55.0, from RHS = 160.7142857142857, to RHS = 200.00
Capacity_Veg[Jun] : shadow price = 49.62962962962963, from RHS = 160.7142857142857, to
RHS = 4500.00
Capacity_Oil[Jan] : shadow price = 65.0, from RHS = 20.0, to RHS = 250.00
Capacity_Oil[Feb] : shadow price = 60.0, from RHS = 80.0, to RHS = 250.00
Capacity_Oil[Mar] : shadow price = 55.0, from RHS = 155.55555555555554, to RHS = inf
Capacity_Oil[Apr] : shadow price = 50.0, from RHS = 155.55555555555554, to RHS = 311.11
Capacity_Oil[May] : shadow price = 45.0, from RHS = 155.55555555555554, to RHS = 311.11
Capacity_Oil[Jun] : shadow price = 76.66666666666669, from RHS = 11.111111111111, to RH
S = 311.11
-----------------------------------------------
Hardness_lower[Jan] : shadow price = 0.0, from RHS = -inf, to RHS = 1120.00
Hardness_lower[Feb] : shadow price = 0.0, from RHS = -inf, to RHS = 1350.00
Hardness_lower[Mar] : shadow price = 0.0, from RHS = -inf, to RHS = 1180.00
Hardness_lower[Apr] : shadow price = 0.0, from RHS = -inf, to RHS = 1350.00
Hardness_lower[May] : shadow price = 0.0, from RHS = -inf, to RHS = 1350.00
Hardness_lower[Jun] : shadow price = 0.0, from RHS = -inf, to RHS = 1350.00
```

```
Hardness_upper[Jan] : shadow price = 0.0, from RHS = -230.0, to RHS = inf
Hardness_upper[Feb] : shadow price = 0.0, from RHS = -170.0, to RHS = 260.00
Hardness_upper[Mar] : shadow price = 0.0, from RHS = -170.0, to RHS = inf
Hardness_upper[Apr] : shadow price = 0.0, from RHS = -170.0, to RHS = 110.00
Hardness_upper[May] : shadow price = 0.0, from RHS = -170.0, to RHS = 110.00
Hardness_upper[Jun] : shadow price = 3.7037037037037024, from RHS = -430.0000000000001,
to RHS = 110.00
-----------------------------------------------
Mass_conservation[Jan] : shadow price = -150.0, from RHS = -373.3333333333333, to RHS =
38.33
Mass_conservation[Feb] : shadow price = -150.0, from RHS = -43.33333333333335, to RHS =
28.33
Mass_conservation[Mar] : shadow price = -150.0, from RHS = -393.3333333333333, to RHS =
28.33
Mass_conservation[Apr] : shadow price = -150.0, from RHS = -18.333333333333353, to RHS
= 28.33
Mass_conservation[May] : shadow price = -150.0, from RHS = -18.333333333333332, to RHS
= 28.33
Mass_conservation[Jun] : shadow price = -172.22222222222223, from RHS = -18.33333333333
3353, to RHS = 71.67
```

(b, 10 puntos) Interpret the sensitivity analysis results with respect to changes in the objective coefficients.

In [9]:
```
# importing libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [112]:
```
objectives = {'holding_cost':[],'price':[],'obj':[]}

# Model deployment
food = gp.Model('Food Manufacture I')
# Quantity of food produced in each period
produce = food.addVars(months, name="Produce")
# Quantity bought of each product in each period
buy = food.addVars(months, oils, name = "Buy")
# Quantity used of each product  in each period
consume = food.addVars(months, oils, name = "Use")
# Quantity stored of each product  in each period
store = food.addVars(months, oils, name = "Store")
#1. Initial Balance
Balance0 = food.addConstrs((init_store + buy[months[0], oil]
                == consume[months[0], oil] + store[months[0], oil]
                for oil in oils), "Initial_Balance")
#2. Balance
Balance = food.addConstrs((store[months[months.index(month)-1], oil] + buy[month, oil]
                == consume[month, oil] + store[month, oil]
                for oil in oils for month in months if month != month[0]), "Balance")
#3. Inventory Target
TargetInv = food.addConstrs((store[months[-1], oil] == target_store for oil in oils),
"End_Balance")
#4.1 Vegetable Oil Capacity
VegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for oil in oils if "VE
G" in oil)
                <= veg_cap for month in months), "Capacity_Veg")
#4.2 Non-vegetable Oil Capacity
NonVegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for oil in oils if
"OIL" in oil)
                <= oil_cap for month in months), "Capacity_Oil")
#5. Hardness
HardnessMin = food.addConstrs((gp.quicksum(hardness[oil]*consume[month, oil] for oil i
n oils)
                >= min_hardness*produce[month] for month in months), "Hardness_lower")
HardnessMax = food.addConstrs((gp.quicksum(hardness[oil]*consume[month, oil] for oil i
n oils)
                <= max_hardness*produce[month] for month in months), "Hardness_upper")
#6. Mass Conservation
MassConservation = food.addConstrs((consume.sum(month) == produce[month] for month in
months), "Mass_conservation")

for i in range(3,24):
    for j in range(135,180):
        holding_cost = i
        price = j

        #0. Objective Function
        obj = price*produce.sum() - buy.prod(cost) - holding_cost*store.sum()
        food.setObjective(obj, GRB.MAXIMIZE) # maximize profit

        food.optimize()

        # appending
        objectives['holding_cost'].append(i)
        objectives['price'].append(j)
        objectives['obj'].append(food.ObjVal)
```

In [114]:
```
# plotting heatmap for objective coefficient changes
fig = plt.figure(figsize=(48,30))
hm = sns.heatmap(pd.DataFrame(objectives).pivot(columns='holding_cost', index='price',
values='obj'), annot=True)
for text in hm.texts:
    if text.get_text() == '1.1e+05':
        text.set_size(15)
        text.set_weight('bold')
        text.set_style('italic')
```

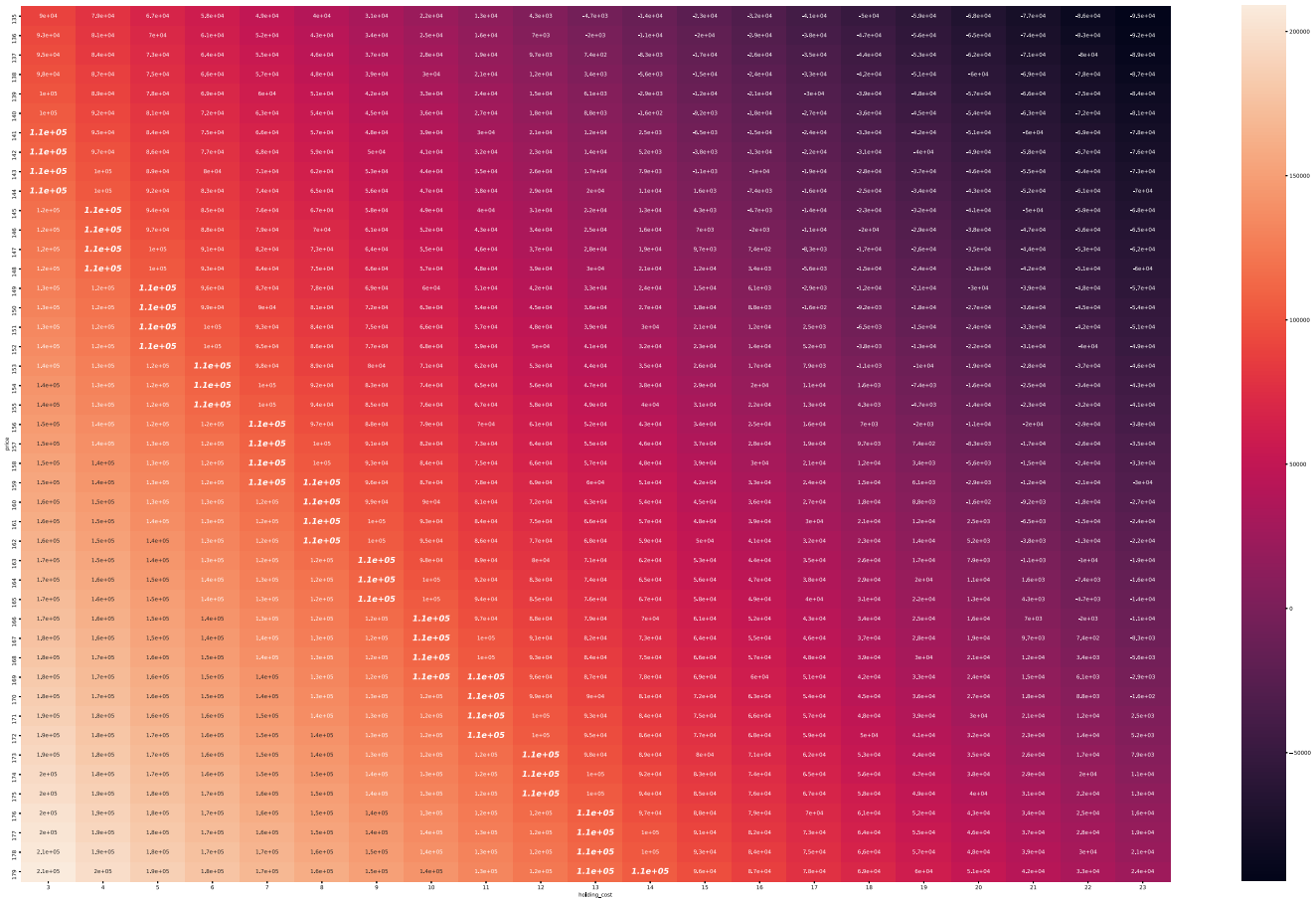

Figure 3

Modifying the values of the objective function coefficients

As we can see in Figure 3, proportionally changing the price and holding cost can yield the optimal solution in a specific range of pairs of price and holding cost.

Which in this case must be proportional to each other to an extent for the result to be optimal.

Obviously, if our costs were signfiicantly lower and our price higher, our optimal value would be higher, however, that wouldn't accurately portray reality.

If costs would remain fixed at 5, we would increase profit probably only by increasing price if our changes were exclusively done in the objective function.

As we increase costs, we reduce profits and we can see the effects of holding costs are heavier than the effects of price, therefore an increase of 1 in our holding cost would force us to increase the price by at least 3 in order to obtain the same optimal value.

We could say that the effect of a unit increase or decrease of **holding cost** is 3-4x more significant than a unit increase of **price** in our optimal value.

(c, 10 puntos) Interpret the sensitivity analysis results with respect to changes in the constraint right-hand sides.

In [90]:

```
# parameters to test out
test_params = {
    'init_store': range(400, 600, 5),
    'target_store': range(400, 600, 5),
    'veg_cap': range(100, 300, 5),
    'oil_cap': range(150, 350, 5),
    'min_hardness': range(1, 41),
    'max_hardness': range(1, 41)
}

# objective function coefficients
holding_cost = 5
price = 150

# function to loop through test parameters to produce a heatmap
def loop_RHS(**params):
    default_params = {
        'init_store_b': (True, 500),
        'target_store_b': (True, 500),
        'veg_cap_b': (True, 200),
        'oil_cap_b': (True, 250),
        'min_hardness_b': (True, 3),
        'max_hardness_b': (True, 6)
    }

    # dict for obj values per RHS change
    obj_changes = {param: [] for param in params.keys()}

    for param, options in params.items():

        dp = {k: v for k, v in default_params.items() if k != f'{param}_b'}
        dp[f'{param}_b'] = (False, 0)

        for option in options:
            # Model deployment
            food = gp.Model('Food Manufacture I')
            # Quantity of food produced in each period
            produce = food.addVars(months, name="Produce")
            # Quantity bought of each product in each period
            buy = food.addVars(months, oils, name="Buy")
            # Quantity used of each product  in each period
            consume = food.addVars(months, oils, name="Use")
            # Quantity stored of each product  in each period
            store = food.addVars(months, oils, name="Store")

            if param == 'init_store':
                # 1. Initial Balance
                Balance0 = food.addConstrs((option + buy[months[0], oil]
                                            == consume[months[0], oil] + store[months[
0], oil]
                                            for oil in oils), "Initial_Balance")
            elif param == 'target_store':
                # 3. Inventory Target
                TargetInv = food.addConstrs(
                    (store[months[-1], oil] == option for oil in oils), "End_Balance")
            elif param == 'veg_cap':
                # 4.1 Vegetable Oil Capacity
                VegCapacity = food.addConstrs((gp.quicksum(
                    consume[month, oil] for oil in oils if "VEG" in oil) <= option for
month in months), "Capacity_Veg")
            elif param == 'oil_cap':
                # 4.2 Non-vegetable Oil Capacity
                NonVegCapacity = food.addConstrs((gp.quicksum(
                    consume[month, oil] for oil in oils if "OIL" in oil) <= option for
month in months), "Capacity_Oil")
            elif param == 'min_hardness':
                # 5.1 Min Hardness
                HardnessMin = food.addConstrs((gp.quicksum(
                    hardness[oil]*consume[month, oil] for oil in oils) >= option*produ
ce[month] for month in months), "Hardness_lower")
            else:
                # 5.2 Max Hardness
```

```
                HardnessMax = food.addConstrs((gp.quicksum(
                    hardness[oil]*consume[month, oil] for oil in oils) <= option*produ
ce[month] for month in months), "Hardness_upper")


            # constraints that remain the same, as the changes for the constraints are
done one by one
            if dp['init_store_b'][0] == True:
                Balance0 = food.addConstrs((dp['init_store_b'][1] + buy[months[0], oil
]
                                            == consume[months[0], oil] + store[months[
0], oil]
                                            for oil in oils), "Initial_Balance")
            Balance = food.addConstrs((store[months[months.index(month)-1], oil] + buy
[month, oil]
                                       == consume[month, oil] + store[month, oil]
                                       for oil in oils for month in months if month !=
month[0]), "Balance")
            if dp['target_store_b'][0] == True:
                TargetInv = food.addConstrs(
                    (store[months[-1], oil] == dp['target_store_b'][1] for oil in oils
), "End_Balance")
            if dp['veg_cap_b'][0] == True:
                VegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for oil
in oils if "VEG" in oil)
                                               <= dp['veg_cap_b'][1] for month in mont
hs), "Capacity_Veg")
            if dp['oil_cap_b'][0] == True:
                NonVegCapacity = food.addConstrs((gp.quicksum(consume[month, oil] for
oil in oils if "OIL" in oil)
                                                  <= dp['oil_cap_b'][1] for month in m
onths), "Capacity_Oil")
            if dp['min_hardness_b'][0] == True:
                HardnessMin = food.addConstrs((gp.quicksum(hardness[oil]*consume[month
, oil] for oil in oils)
                                               >= dp['min_hardness_b'][1]*produce[mont
h] for month in months), "Hardness_lower")
            if dp['max_hardness_b'][0] == True:
                HardnessMax = food.addConstrs((gp.quicksum(hardness[oil]*consume[month
, oil] for oil in oils)
                                               <= dp['max_hardness_b'][1]*produce[mont
h] for month in months), "Hardness_upper")
            MassConservation = food.addConstrs(
                (consume.sum(month) == produce[month] for month in months), "Mass_cons
ervation")
            obj = price*produce.sum() - buy.prod(cost) - holding_cost*store.sum()
            food.setObjective(obj, GRB.MAXIMIZE)  # maximize profit
            food.optimize()
            try:
                obj_changes[param].append(food.ObjVal)
            except:
                obj_changes[param].append(0)

    return obj_changes


obj_changes = loop_RHS(**test_params)
```

In [115]:
```
# plotting heatmap for objective coefficient changes
fig = plt.figure(figsize=(22,20))
ax = sns.heatmap(pd.DataFrame(obj_changes), annot=True)
ax.get_yaxis().set_visible(False)
for text in ax.texts:
    if text.get_text() == '1.1e+05':
        text.set_size(18)
        text.set_weight('bold')
        text.set_style('italic')
```

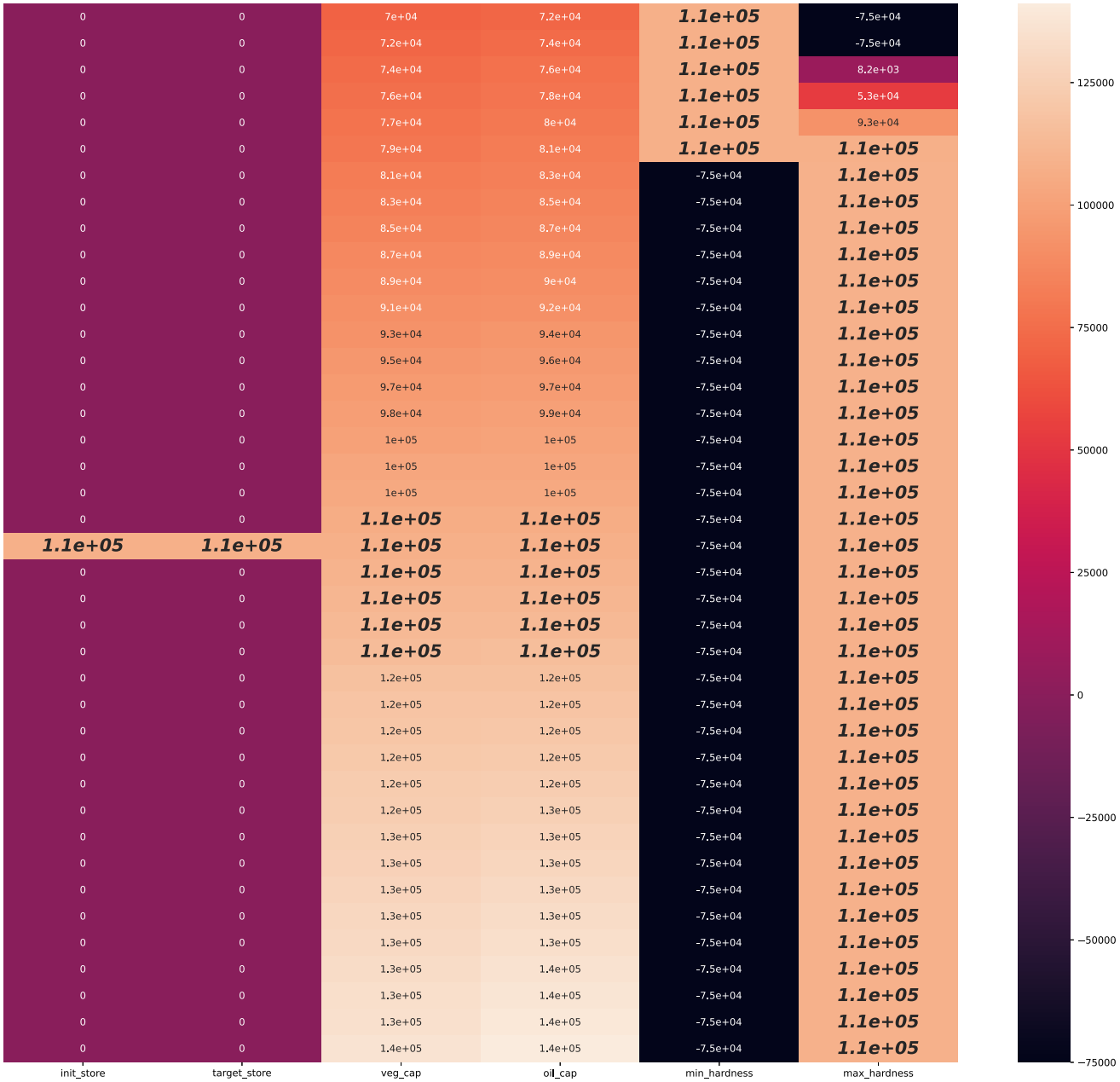

Figure 4

Modifying the values of constraint RHS

Methodology

The values for the constraint RHS were changed individually, therefore, each variable had its own loop. The variable in loop would change its values while the other variables would remain the same, effectively ignoring interactions between variables, as this approach would be too intensive computationally.

data returned by the testing

In [117]:
```
pd.DataFrame(obj_changes)
```

Out[117]:

	init_store	target_store	veg_cap	oil_cap	min_hardness	max_hardness
0	0.000000	0.000000	69750.000000	71766.835017	107842.592593	-75000.000000
1	0.000000	0.000000	71675.000000	73934.259259	107842.592593	-75000.000000
2	0.000000	0.000000	73600.000000	76101.683502	107842.592593	8207.478006
3	0.000000	0.000000	75525.000000	77950.925926	107842.592593	52734.848485
4	0.000000	0.000000	77450.000000	79709.259259	107842.592593	92613.636364
5	0.000000	0.000000	79375.000000	81467.592593	107472.222222	107842.592593
6	0.000000	0.000000	81300.000000	83225.925926	-75000.000000	108250.000000
7	0.000000	0.000000	83225.000000	84984.259259	-75000.000000	108250.000000
8	0.000000	0.000000	85150.000000	86742.592593	-75000.000000	108250.000000
9	0.000000	0.000000	87075.000000	88500.925926	-75000.000000	108250.000000
10	0.000000	0.000000	89000.000000	90259.259259	-75000.000000	108250.000000
11	0.000000	0.000000	90925.000000	92017.592593	-75000.000000	108250.000000
12	0.000000	0.000000	92850.000000	93775.925926	-75000.000000	108250.000000
13	0.000000	0.000000	94730.555556	95534.259259	-75000.000000	108250.000000
14	0.000000	0.000000	96603.703704	97292.592593	-75000.000000	108250.000000
15	0.000000	0.000000	98476.851852	99050.925926	-75000.000000	108250.000000
16	0.000000	0.000000	100350.000000	100809.259259	-75000.000000	108250.000000
17	0.000000	0.000000	102223.148148	102567.592593	-75000.000000	108250.000000
18	0.000000	0.000000	104096.296296	104325.925926	-75000.000000	108250.000000
19	0.000000	0.000000	105969.444444	106084.259259	-75000.000000	108250.000000
20	107842.592593	107842.592593	107842.592593	107842.592593	-75000.000000	108250.000000
21	0.000000	0.000000	109590.740741	109600.925926	-75000.000000	108250.000000
22	0.000000	0.000000	111338.888889	111359.259259	-75000.000000	108250.000000
23	0.000000	0.000000	113087.037037	113117.592593	-75000.000000	108250.000000
24	0.000000	0.000000	114835.185185	114875.925926	-75000.000000	108250.000000
25	0.000000	0.000000	116583.333333	116634.259259	-75000.000000	108250.000000
26	0.000000	0.000000	118331.481481	118392.592593	-75000.000000	108250.000000
27	0.000000	0.000000	119824.074074	120150.925926	-75000.000000	108250.000000
28	0.000000	0.000000	121194.444444	121909.259259	-75000.000000	108250.000000
29	0.000000	0.000000	122564.814815	123667.592593	-75000.000000	108250.000000
30	0.000000	0.000000	123935.185185	125425.925926	-75000.000000	108250.000000
31	0.000000	0.000000	125305.555556	127184.259259	-75000.000000	108250.000000
32	0.000000	0.000000	126675.925926	128942.592593	-75000.000000	108250.000000
33	0.000000	0.000000	128046.296296	130675.000000	-75000.000000	108250.000000
34	0.000000	0.000000	129416.666667	132400.000000	-75000.000000	108250.000000
35	0.000000	0.000000	130787.037037	134125.000000	-75000.000000	108250.000000
36	0.000000	0.000000	132157.407407	135850.000000	-75000.000000	108250.000000
37	0.000000	0.000000	133527.777778	137575.000000	-75000.000000	108250.000000
38	0.000000	0.000000	134898.148148	139300.000000	-75000.000000	108250.000000
39	0.000000	0.000000	136268.518519	141025.000000	-75000.000000	108250.000000

In [120]:
```
obj_changes_2 = {x:[] for x in obj_changes.keys()}
for (key,optval), change in zip(obj_changes.items(), test_params.values()):
    for opt, chng in zip(optval,change):
        obj_changes_2[key].append((chng,opt))
pd.DataFrame(obj_changes_2)
```

Out[120]:

	init_store	target_store	veg_cap	oil_cap	min_hardness	n
0	(400, 0)	(400, 0)	(100, 69750.0)	(150, 71766.83501683499)	(1, 107842.59259259261)	
1	(405, 0)	(405, 0)	(105, 71675.00000000003)	(155, 73934.25925925927)	(2, 107842.59259259261)	
2	(410, 0)	(410, 0)	(110, 73600.0)	(160, 76101.68350168355)	(3, 107842.59259259261)	8207.4
3	(415, 0)	(415, 0)	(115, 75525.0)	(165, 77950.9259259259)	(4, 107842.59259259261)	52734.8
4	(420, 0)	(420, 0)	(120, 77450.0)	(170, 79709.25925925927)	(5, 107842.59259259261)	92613.
5	(425, 0)	(425, 0)	(125, 79375.0)	(175, 81467.59259259267)	(6, 107472.22222222222)	107842.
6	(430, 0)	(430, 0)	(130, 81300.0)	(180, 83225.92592592593)	(7, -75000.0)	
7	(435, 0)	(435, 0)	(135, 83224.99999999999)	(185, 84984.25925925927)	(8, -75000.0)	
8	(440, 0)	(440, 0)	(140, 85150.00000000001)	(190, 86742.59259259261)	(9, -75000.0)	
9	(445, 0)	(445, 0)	(145, 87075.0)	(195, 88500.92592592593)	(10, -75000.0)	
10	(450, 0)	(450, 0)	(150, 89000.0)	(200, 90259.25925925927)	(11, -75000.0)	
11	(455, 0)	(455, 0)	(155, 90925.0)	(205, 92017.59259259261)	(12, -75000.0)	
12	(460, 0)	(460, 0)	(160, 92850.0)	(210, 93775.92592592593)	(13, -75000.0)	
13	(465, 0)	(465, 0)	(165, 94730.55555555556)	(215, 95534.25925925926)	(14, -75000.0)	
14	(470, 0)	(470, 0)	(170, 96603.70370370371)	(220, 97292.59259259261)	(15, -75000.0)	
15	(475, 0)	(475, 0)	(175, 98476.85185185182)	(225, 99050.92592592593)	(16, -75000.0)	
16	(480, 0)	(480, 0)	(180, 100350.0)	(230, 100809.25925925927)	(17, -75000.0)	
17	(485, 0)	(485, 0)	(185, 102223.14814814815)	(235, 102567.59259259261)	(18, -75000.0)	
18	(490, 0)	(490, 0)	(190, 104096.29629629629)	(240, 104325.92592592596)	(19, -75000.0)	
19	(495, 0)	(495, 0)	(195, 105969.44444444444)	(245, 106084.25925925927)	(20, -75000.0)	
20	(500, 107842.59259259261)	(500, 107842.59259259261)	(200, 107842.59259259261)	(250, 107842.59259259261)	(21, -75000.0)	
21	(505, 0)	(505, 0)	(205, 109590.74074074073)	(255, 109600.92592592596)	(22, -75000.0)	
22	(510, 0)	(510, 0)	(210, 111338.88888888888)	(260, 111359.25925925927)	(23, -75000.0)	
23	(515, 0)	(515, 0)	(215, 113087.03703703705)	(265, 113117.59259259261)	(24, -75000.0)	
24	(520, 0)	(520, 0)	(220, 114835.18518518517)	(270, 114875.92592592596)	(25, -75000.0)	
25	(525, 0)	(525, 0)	(225, 116583.33333333331)	(275, 116634.25925925927)	(26, -75000.0)	
26	(530, 0)	(530, 0)	(230, 118331.48148148152)	(280, 118392.59259259261)	(27, -75000.0)	

27	(535, 0)	(535, 0)	(235, 119824.07407407401)	(285, 120150.92592592596)	(28, -75000.0)
28	(540, 0)	(540, 0)	(240, 121194.44444444447)	(290, 121909.25925925927)	(29, -75000.0)
29	(545, 0)	(545, 0)	(245, 122564.81481481477)	(295, 123667.59259259261)	(30, -75000.0)
30	(550, 0)	(550, 0)	(250, 123935.1851851852)	(300, 125425.92592592596)	(31, -75000.0)
31	(555, 0)	(555, 0)	(255, 125305.55555555559)	(305, 127184.25925925927)	(32, -75000.0)
32	(560, 0)	(560, 0)	(260, 126675.92592592593)	(310, 128942.59259259261)	(33, -75000.0)
33	(565, 0)	(565, 0)	(265, 128046.29629629635)	(315, 130675.0)	(34, -75000.0)
34	(570, 0)	(570, 0)	(270, 129416.66666666663)	(320, 132400.0)	(35, -75000.0)
35	(575, 0)	(575, 0)	(275, 130787.03703703708)	(325, 134125.0)	(36, -75000.0)
36	(580, 0)	(580, 0)	(280, 132157.40740740742)	(330, 135850.0)	(37, -75000.0)
37	(585, 0)	(585, 0)	(285, 133527.7777777778)	(335, 137575.0)	(38, -75000.0)
38	(590, 0)	(590, 0)	(290, 134898.14814814818)	(340, 139300.0)	(39, -75000.0)
39	(595, 0)	(595, 0)	(295, 136268.51851851857)	(345, 141025.0)	(40, -75000.0)

Initial storage amount and target storage amount in tons

For the values of these two constraints, we get an infeasable model in every option we tried except the values provided by the model by default, therefore this value must remain as default.

Capacity in tons to refine vegetable oils and non-vegetable oils

Increasing the capacity to refine vegetable oils would yield a higher profit, as expected, but not by a huge margin, and if such decision increases the cost too much we might see diminishing returns. therefore we reach the optimal value around the value set up by default for each of them. With a higher room to increase the capacity while obtaining the same optimal value than room to decrease such capacity.

lowest hardness allowed for the final product

Anything below or equal to 5 will yield the optimal value. Anything above or equal to 7 would return a negative value.

highest hardness allowed for the final product.

As long as the maximum hardness allowed is 6 or more, the optimal value will remain. However, less than 6 maximum hardness would yield a result significantly lower than the optimal value and even negative below 3.