Problem Set 1

Andrés Felipe Mejía Rodríguez

Problem 1

Dual and graphical solution

The original problem:

$$\max\ 2x_1 + 3x_2 - 4x_3$$

subject to:

$$3x_1 + 5x_2 + 2x_3 = 15$$
$$x_1 + 3x_2 - 4x_3 = 8$$
$$x_1, x2, x3 \geq 0$$

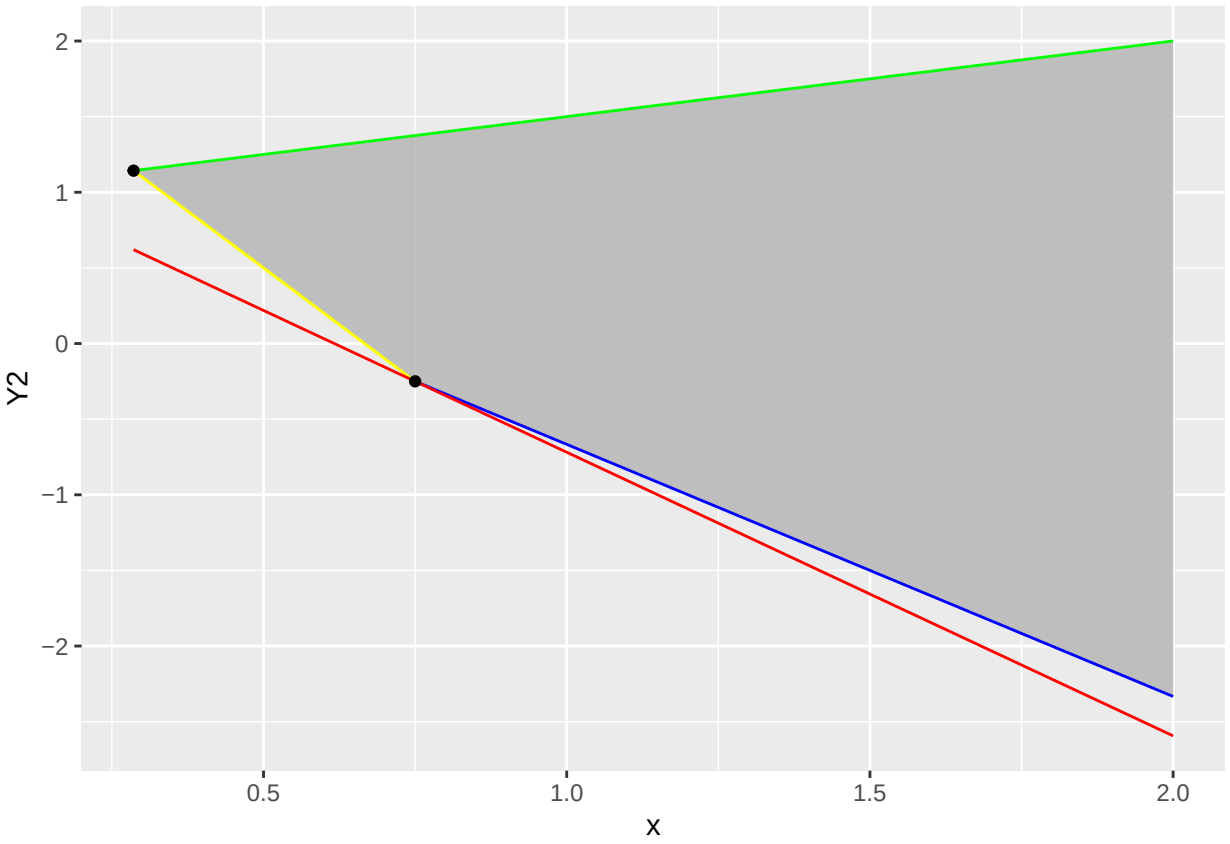
for this kind of problem: Maximize $\vec{c}^t\vec{x}$ subject to $A\vec{x} = \vec{b}, \vec{x} \geq 0$ the dual is given by: minimize $\vec{b}^t\vec{y}$ subject to $A^t y \geq \vec{c}$. So this gives us

$$\min\ 15y_1 + 8y_2$$

subject to:

$$3y_1 + y_2 \geq 2$$
$$5y_1 + 3y_2 \geq 3$$
$$2y_1 - 4y_2 \geq -4$$

Plotting this poligon we have:



So it seems that the optimal solution is on the vertex $(3/4, -1/4)$ that gives us 9.25 as the optimal value, its level curve is plotted in red.

Returning to the primal

By complementary slackness we have $(A^t\vec{y} - \vec{c})^t\vec{x} = 0$ so in this case

$$\left(\begin{pmatrix}3 & 1\\ 5 & 3\\ 2 & -4\end{pmatrix}\begin{pmatrix}3/4\\ -1/4\end{pmatrix} - \begin{pmatrix}2\\ 3\\ -4\end{pmatrix}\right)^t \begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}0\\ 0\\ 6.5\end{pmatrix}^t \begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}$$

so $6.5x_3 = 0$ that means $x_3 = 0$ and we have

$$3x_1 + 5x_2 = 15$$
$$x_1 + 3x_2 = 8$$

Which has solutions $x = 5/4$, $y = 9/4$

Reduced costs

The reduced costs are calculated by $\vec{c} - A\vec{y}$ where y is the dual solution,the negative of this quantity is calculated as an intermediate step to calculate the primal solution from the dual solution. This is:

$$\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}0\\ 0\\ -6.5\end{pmatrix}$$

as we can see are the reduced costs are 0 for variables one and two, this means that these variables are not zero in the solution. The fact that x_3 is not zero means that x_3 will be zero in the solution, it also implies that an increase (given that we are maximizing) of the coefficient of x_3 by this value will make it be an active variable, thus having a non zero value.

Sensibility analysis - constraints

Analytic solution

We will slightly alter the constraint and see how changes in the contraints change the problem:

$$\max 2x_1 + 3x_2 - 4x_3$$

subject to

$$3x_1 + 5x_2 + 2x_3 = 15 + \Delta\pi_1$$
$$x_1 + 3x_2 - 4x_3 = 8 + \Delta\pi_2$$
$$x_1, x2, x3 \geq 0$$

In ordert to have the same solution in the dual we must have the same active variables that in the solution of the original problem, so $x_3 = 0$. In that context

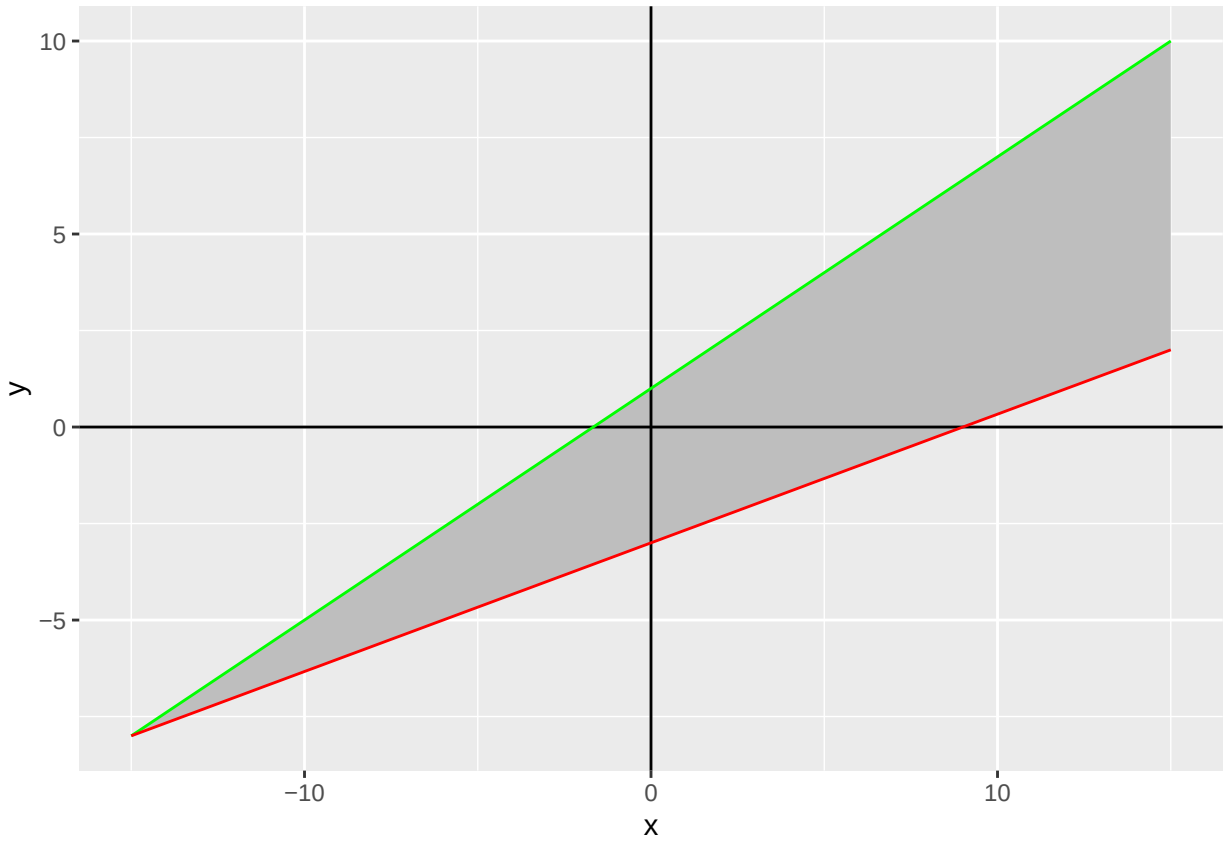
$$3x_1 + 5x_2 = 15 + \Delta\pi_1$$
$$x_1 + 3x_2 = 8 + \Delta\pi_2$$

Which has as solutions $x_1 = 1/4(3\Delta\pi_1 - 5\Delta\pi_2 + 5)$ and $x_2 = 1/4(-\Delta\pi_1 + 3\Delta\pi_2 + 9)$ then using the condition that $x_i \geq 0$ we have:

$$3\Delta\pi_1 - 5\Delta\pi_2 + 5 \geq 0$$

$$-\Delta\pi_1 + 3\Delta\pi_2 + 9 \geq 0$$

These inequalities generate the area given in the next plot. Also note the intersection between that area and both axis these give us the values when the other value is zero. For sensibility we get $(13.33, 24)$ and $(5, 9)$ after adding back the right hand side



Sensibility analysis using Gurobi

```
import gurobipy as gbp


# %% Define el modelo vacio

Modelo1=gbp.Model("ModeloTarea")


# %% Define Variables
```

```
## Restricted license - for non-production use only - expires 2022-01-13
```

```
x1=Modelo1.addVar(vtype=gbp.GRB.CONTINUOUS,name="x1")
x2=Modelo1.addVar(vtype=gbp.GRB.CONTINUOUS,name="x2")
x3=Modelo1.addVar(vtype=gbp.GRB.CONTINUOUS,name="x3")

# %% Funcion objetivo
Modelo1.setObjective(2*x1+3*x2-4*x3,gbp.GRB.MAXIMIZE)

# %% Restricciones

Modelo1.addConstr(3*x1+5*x2+2*x3==15)
```

```
## <gurobi.Constr *Awaiting Model Update*>
```

```
Modelo1.addConstr(x1+3*x2-4*x3==8)


# %% Ver el modelo
```

```
## <gurobi.Constr *Awaiting Model Update*>
```

```
Modelo1.update()
Modelo1.display()

# %%
```

```
## Maximize
##    <gurobi.LinExpr: 2.0 x1 + 3.0 x2 + -4.0 x3>
## Subject To
##    R0 : <gurobi.LinExpr: 3.0 x1 + 5.0 x2 + 2.0 x3> = 15.0
##    R1 : <gurobi.LinExpr: x1 + 3.0 x2 + -4.0 x3> = 8.0
```

```
Modelo1.optimize()
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 2 physical cores, 2 logical processors, using up to 2 threads
## Optimize a model with 2 rows, 3 columns and 6 nonzeros
## Model fingerprint: 0xb05650d2
## Coefficient statistics:
##   Matrix range     [1e+00, 5e+00]
##   Objective range  [2e+00, 4e+00]
##   Bounds range     [0e+00, 0e+00]
##   RHS range        [8e+00, 2e+01]
## Presolve time: 0.00s
## Presolved: 2 rows, 3 columns, 6 nonzeros
##
## Iteration    Objective       Primal Inf.    Dual Inf.      Time
##        0    7.0000000e+30   2.625000e+30   7.000000e+00      0s
##        2    9.2500000e+00   0.000000e+00   0.000000e+00      0s
##
## Solved in 2 iterations and 0.00 seconds
## Optimal objective  9.250000000e+00
```

```
print('\nOptimal shadow prices:\n')
```

```
##
## Optimal shadow prices:
```

```
for c in Modelo1.getConstrs():
        print("%s %s %8.2f %s %8.2f %s %8.2f" % (c.ConstrName, ": shadow price = ", c.Pi, ", from RHS =
```

```
## R0 : shadow price =      0.75 , from RHS =     13.33 to RHS =     24.00
## R1 : shadow price =     -0.25 , from RHS =      5.00 to RHS =      9.00
```

```
for v in Modelo1.getVars():
    print("%s %s %8.2f %s %8.2f %s %8.2f %s %8.2f" %
              (v.Varname, "=", v.X, ", reduced cost = ", abs(v.RC), ", from coeff = ", v.SAObjLow, "to
    print(" ")
```

```
## x1 =     1.25 , reduced cost =      0.00 , from coeff =      1.00 to coeff =       inf
##
## x2 =     2.25 , reduced cost =      0.00 , from coeff =      -inf to coeff =      4.86
```

```
##
## x3 =     0.00 , reduced cost =      6.50 , from coeff =      -inf to coeff =      2.50
##
```

Note that the solution of the problem correspond to the one given in part 2, the shadow prices correspond to the solution of the dual problem given in part 3 and the ranges for the sensibility analysis for the shadow prices side correspond to those found above. The range corresponding for the sensibility analysis for the objective function is given in the next section.

Sensivity analysis - objective function We will consider changes in the objective function that do not change the solution of the dual problem that is:

$$\max (2 + \Delta x_1)x_1 + (3 + \Delta x_2)x_2 + (-4 + \Delta x_3)x_3$$

subject to:

$$3x_1 + 5x_2 + 2x_3 = 15$$
$$x_1 + 3x_2 - 4x_3 = 8$$
$$x_1, x2, x3 \geq 0$$

We again consider the dual of the problem

$$\min 15y_1 + 8y_2$$

subject to:

$$3y_1 + y_2 \geq 2 + +\Delta x_1$$
$$5y_1 + 3y_2 \geq 3 + \Delta x_2$$
$$2y_1 - 4y_2 \geq -4 + \Delta x_3$$

To keep the same solution the same equations should have the same complementary slackness i.e the first two inequalites should be equalities:

$$3y_1 + y_2 = 2 + +\Delta x_1$$

$$5y_1 + 3y_2 = 3 + \Delta x_2$$

Solving for y_1 and y_2 we get

$$y_1 = 3/4 + 3/4\Delta x_1 - 1/4\Delta x_2$$
$$y_2 = -1/4 - 5/4\Delta x_1 + 3/4\Delta x_2$$

Using the third inequality we have

$$2(3/4 + 3/4\Delta x_1 - 1/4\Delta x_2) - 4(-1/4 - 5/4\Delta x_1 + 3/4\Delta x_2) \geq -4 + \Delta x_3$$

$$3/2 + 3/2\Delta x_1 - 1/2\Delta x_2 + 1 + 5\Delta x_1 - 3\Delta x_2 \geq -4 + \Delta x_3$$

$$13/2 + 13/2\Delta x_1 - 7/2\Delta x_2 \geq \Delta x_3$$

If $\Delta x_2 = \Delta x_3 = 0$

$$\Delta x_1 \geq -1$$

So the range of the coefficient of x_1 is $(2, \infty)$

If $\Delta x_1 = \Delta x_3 = 0$

$$13/2 - 7/2\Delta x_2 \geq 0$$

$$13/7 \geq \Delta x_2$$

If $\Delta x_1 = \Delta x_2 = 0$

$$13/2 \geq \Delta x_3$$

So the range of the coefficient of x_3 is $(-\infty, 2.5)$

In all three cases the values are the same obtained ussing Gurobi above.

Problem 2

Formulation of the MAE problem

The problem is a regression using the absolute value as the measure to be minimized, that is:

$$\min_{\beta_0, \beta_1, \beta 2 \in \mathbb{R}} \sum_{1=i}^{n} |H_i - \beta_0 + \beta_1 G + \beta_2 S|$$

Where H is the height of an individual, G is their glove (hand) size and S is their shoe (feet) size. However we are faced with the fact that the problem is not linear due to the presence of the absolute values.

There are several ways turning this problem into a linear one [ref], we will use the method known as a variable spliting where we will take each expression whose absolute value must be calculated and split it into positive and negative parts, doing this we have:

$$a = a^+ - a^-$$
$$|a| = a^+ - a^-$$
$$a^+.a^- \geq 0$$

Our problem becomes:

$$\min \sum_{i=1}^{n} e_i^+ + e_i^-$$

subject to

$$e_i^+ - e_i^- + \beta_0 + \beta_1 G_i + \beta_2 S_i = H_i$$
$$e_i^+, e_i^- \geq 0$$
$$i \in \{1, 2, \ldots, n\}$$

The formulation of this problem using gurobi python using a modified form of the template seen in class is as follows:

```
from gurobipy import *

n = 7 # number of observations

oneton = range(1, n+1)  # list [1, ..., n]

zerotoone = range(3) # list [0, 1]

G_data = [17.9,18.2,18.5,16.9,17.3,17.9,18.1]
S_data = [30.1,29.5,30.4,31.6,27.4,28.3,33.4]


H_data = [176.2,176.8,184.2,173.2,172.8,174.1,180.5]

G = {j : G_data[j-1] for j in oneton}

S = {j : S_data[j-1] for j in oneton}
H = {j : H_data[j-1] for j in oneton}

model = Model('mae')

ePlus = model.addVars(oneton, name="ePlus")

eMinus = model.addVars(oneton, name="eMinus")

b = model.addVars(zerotoone, name="b", lb=-GRB.INFINITY)

model.addConstrs((ePlus[i] - eMinus[i] + b[0] + G[i] * b[1]+S[i]*b[2] == H[i] for i in oneton), name =

# Objective
```

```
## {1: <gurobi.Constr *Awaiting Model Update*>, 2: <gurobi.Constr *Awaiting Model Update*>, 3: <gurobi.C
```

```
obj = quicksum(((ePlus[i] + eMinus[i]) for i in oneton))

model.setObjective(obj, GRB.MINIMIZE)

model.setParam(GRB.Param.PoolSolutions, 3)
```

```
## Changed value of parameter PoolSolutions to 3
##    Prev: 10  Min: 1  Max: 2000000000  Default: 10
```

```
model.Params.PoolSearchMode=2
```

```
## Changed value of parameter PoolSearchMode to 2
##    Prev: 0  Min: 0  Max: 2  Default: 0
```

```
model.update()
model.display()
```

```
## Minimize
##    <gurobi.LinExpr: ePlus[1] + ePlus[2] + ePlus[3] + ePlus[4] + ePlus[5] + ePlus[6] + ePlus[7] + eMi
## Subject To
##    pi[1] : <gurobi.LinExpr: ePlus[1] + -1.0 eMinus[1] + b[0] + 17.9 b[1] + 30.1 b[2]> = 176.2
##    pi[2] : <gurobi.LinExpr: ePlus[2] + -1.0 eMinus[2] + b[0] + 18.2 b[1] + 29.5 b[2]> = 176.8
##    pi[3] : <gurobi.LinExpr: ePlus[3] + -1.0 eMinus[3] + b[0] + 18.5 b[1] + 30.4 b[2]> = 184.2
```

```
##    pi[4] : <gurobi.LinExpr: ePlus[4] + -1.0 eMinus[4] + b[0] + 16.9 b[1] + 31.6 b[2]> = 173.2
##    pi[5] : <gurobi.LinExpr: ePlus[5] + -1.0 eMinus[5] + b[0] + 17.3 b[1] + 27.4 b[2]> = 172.8
##    pi[6] : <gurobi.LinExpr: ePlus[6] + -1.0 eMinus[6] + b[0] + 17.9 b[1] + 28.3 b[2]> = 174.1
##    pi[7] : <gurobi.LinExpr: ePlus[7] + -1.0 eMinus[7] + b[0] + 18.1 b[1] + 33.4 b[2]> = 180.5
## Bounds
##    b[0]  free
##    b[1]  free
##    b[2]  free
```

```
model.optimize()

# Display solution (print the name of each variable and the solution value)
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 2 physical cores, 2 logical processors, using up to 2 threads
## Optimize a model with 7 rows, 17 columns and 35 nonzeros
## Model fingerprint: 0x5f9389a7
## 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 time: 0.00s
## Presolved: 7 rows, 17 columns, 35 nonzeros
##
## Iteration    Objective       Primal Inf.    Dual Inf.      Time
##        0      handle free variables                         0s
##        8    7.4625000e+00   0.000000e+00   0.000000e+00      0s
##
## Solved in 8 iterations and 0.00 seconds
## Optimal objective  7.462500000e+00
```

```
print('---------------------------------')
```

```
## ---------------------------------
```

```
print('\nOptimal solution:\n')
```

```
##
## Optimal solution:
```

```
print('Variable Information Including Sensitivity Information:')

# tVars = PrettyTable(['Variable Name', ' Value', 'ReducedCost',
#                      ' SensLow', ' SensUp'])  #column headers
```

```
## Variable Information Including Sensitivity Information:
```

```
for v in model.getVars():
    print("%s %s %8.2f %s %8.2f %s %8.2f %s %8.2f" %
              (v.Varname, "=", v.X, ", reduced cost = ", abs(v.RC), ", from coeff = ", v.SAObjLow, "to
    print(" ")
```

```
## ePlus[1] =     0.00 , reduced cost =      0.85 , from coeff =      0.15 to coeff =       inf
##
## ePlus[2] =     0.00 , reduced cost =      2.00 , from coeff =     -1.00 to coeff =       inf
```

```
##
## ePlus[3] =     4.97 , reduced cost =      0.00 , from coeff =      0.29 to coeff =      1.96
##
## ePlus[4] =     0.00 , reduced cost =      1.27 , from coeff =     -0.27 to coeff =       inf
##
## ePlus[5] =     2.10 , reduced cost =      0.00 , from coeff =      0.47 to coeff =      1.72
##
## ePlus[6] =     0.00 , reduced cost =      2.00 , from coeff =     -1.00 to coeff =       inf
##
## ePlus[7] =     0.00 , reduced cost =      0.88 , from coeff =      0.12 to coeff =       inf
##
## eMinus[1] =     0.00 , reduced cost =      1.15 , from coeff =     -0.15 to coeff =       inf
##
## eMinus[2] =     0.15 , reduced cost =      0.00 , from coeff =      0.15 to coeff =      1.63
##
## eMinus[3] =     0.00 , reduced cost =      2.00 , from coeff =     -1.00 to coeff =       inf
##
## eMinus[4] =     0.00 , reduced cost =      0.73 , from coeff =      0.27 to coeff =       inf
##
## eMinus[5] =     0.00 , reduced cost =      2.00 , from coeff =     -1.00 to coeff =       inf
##
## eMinus[6] =     0.25 , reduced cost =      0.00 , from coeff =      0.28 to coeff =      1.53
##
## eMinus[7] =     0.00 , reduced cost =      1.12 , from coeff =     -0.12 to coeff =       inf
##
## b[0] =    63.97 , reduced cost =      0.00 , from coeff =     -0.05 to coeff =      0.07
##
## b[1] =     4.54 , reduced cost =      0.00 , from coeff =     -1.39 to coeff =      0.79
##
## b[2] =     1.03 , reduced cost =      0.00 , from coeff =     -2.55 to coeff =      3.15
##
```

```
print('\nOptimal objective value: %g' % model.objVal)
```

```
##
## Optimal objective value: 7.4625
```

```
print('\nOptimal shadow prices:\n')
```

```
##
## Optimal shadow prices:
```

```
for c in model.getConstrs():
        print("%s %s %8.2f %s %8.2f %s %8.2f" % (c.ConstrName, ": shadow price = ", c.Pi, ", from RHS =
        print(" ")
```

```
## pi[1] : shadow price =      0.15 , from RHS =    176.09 to RHS =    177.51
##
## pi[2] : shadow price =     -1.00 , from RHS =      -inf to RHS =    176.95
##
## pi[3] : shadow price =      1.00 , from RHS =    179.23 to RHS =       inf
##
## pi[4] : shadow price =     -0.27 , from RHS =    163.89 to RHS =    173.67
##
## pi[5] : shadow price =      1.00 , from RHS =    170.70 to RHS =       inf
##
```

```
## pi[6] : shadow price =     -1.00 , from RHS =      -inf to RHS =    174.35
##
## pi[7] : shadow price =      0.12 , from RHS =    178.40 to RHS =    181.00
##
```

```
print('Model Status: '+str(model.status))
```

```
## Model Status: 2
```

```
print('Number of basic solutions: '+str(model.SolCount))
```

```
## Number of basic solutions: 1
```

ref: http://yetanothermathprogrammingconsultant.blogspot.com/2017/11/lp-and-lad-regression.html

Solution

From the output above we can infer that the equation for the regression is:

$$H = 63.97 + 4.54G + 1.03S$$

Note that we expect at least one pair of each e_i^+, e_i^- to be cero as it is inneficient to have both positives at the same time, also as we have 3 more degrees of freedom (from the three coefficients of the regression) we can guess that 3 e_i will be zero (meaning that the regression will exactly pass though 3 points), this is indeed the case for points 1,4 and 7.

We can also ask gurobi to find multiple solutions in case one or more of the restricctions is compatible with the gradient of the objective function. To do this gurobi was asked to Pool up to 3 solutions, in the end the number of optimal solutions kept is one.

Dual solution

The dual of the problem is

$$\max \sum_{i=1}^{n} y_i \pi_i$$

$$\sum_{i=1}^{n} \pi_i = 0$$

$$\sum_{i=1}^{n} x_{ij} \pi_i = 0 \; j \in \{1, \ldots, m\}$$

$$-1 \leq \pi_i \leq 1 \; i \in \{1, \ldots, n\}$$

Solving this in Gurobi

```
from gurobipy import *

n = 7 # number of observations

oneton = range(1, n+1)  # list [1, ..., n]

zerotoone = range(3) # list [0, 1]

G_data = [17.9,18.2,18.5,16.9,17.3,17.9,18.1]
S_data = [30.1,29.5,30.4,31.6,27.4,28.3,33.4]
```

```
H_data = [176.2,176.8,184.2,173.2,172.8,174.1,180.5]

G = {j : G_data[j-1] for j in oneton}

S = {j : S_data[j-1] for j in oneton}
H = {j : H_data[j-1] for j in oneton}

model = Model('mae-dual')

pi = model.addVars(oneton, name="pi",lb=-1,ub=1)


model.addConstr(quicksum(pi[i] for i in oneton)==0, name = "b0")
```

```
## <gurobi.Constr *Awaiting Model Update*>
```

```
model.addConstr(quicksum(pi[i]*G[i] for i in oneton)==0, name = "b1")
```

```
## <gurobi.Constr *Awaiting Model Update*>
```

```
model.addConstr(quicksum(pi[i]*S[i] for i in oneton)==0, name = "b2")

# Objective
```

```
## <gurobi.Constr *Awaiting Model Update*>
```

```
obj = quicksum((pi[i]*H[i] for i in oneton))

model.setObjective(obj, GRB.MAXIMIZE)

model.setParam(GRB.Param.PoolSolutions, 3)
```

```
## Changed value of parameter PoolSolutions to 3
##    Prev: 10  Min: 1  Max: 2000000000  Default: 10
```

```
model.Params.PoolSearchMode=2
```

```
## Changed value of parameter PoolSearchMode to 2
##    Prev: 0  Min: 0  Max: 2  Default: 0
```

```
model.update()
model.display()
```

```
## Maximize
##    <gurobi.LinExpr: 176.2 pi[1] + 176.8 pi[2] + 184.2 pi[3] + 173.2 pi[4] + 172.8 pi[5] + 174.1 pi[6]
## Subject To
##    b0 : <gurobi.LinExpr: pi[1] + pi[2] + pi[3] + pi[4] + pi[5] + pi[6] + pi[7]> = 0.0
##    b1 : <gurobi.LinExpr: 17.9 pi[1] + 18.2 pi[2] + 18.5 pi[3] + 16.9 pi[4] + 17.3 pi[5] + 17.9 pi[6]
##    b2 : <gurobi.LinExpr: 30.1 pi[1] + 29.5 pi[2] + 30.4 pi[3] + 31.6 pi[4] + 27.4 pi[5] + 28.3 pi[6]
## Bounds
##    -1.0 <= pi[1] <= 1.0
##    -1.0 <= pi[2] <= 1.0
##    -1.0 <= pi[3] <= 1.0
##    -1.0 <= pi[4] <= 1.0
##    -1.0 <= pi[5] <= 1.0
##    -1.0 <= pi[6] <= 1.0
```

```
##    -1.0 <= pi[7] <= 1.0
```

```
model.optimize()

# Display solution (print the name of each variable and the solution value)
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 2 physical cores, 2 logical processors, using up to 2 threads
## Optimize a model with 3 rows, 7 columns and 21 nonzeros
## Model fingerprint: 0x1def6cc7
## Coefficient statistics:
##   Matrix range     [1e+00, 3e+01]
##   Objective range  [2e+02, 2e+02]
##   Bounds range     [1e+00, 1e+00]
##   RHS range        [0e+00, 0e+00]
## Presolve time: 0.00s
## Presolved: 3 rows, 7 columns, 21 nonzeros
##
## Iteration    Objective       Primal Inf.    Dual Inf.      Time
##        0    5.1600000e+01   2.085000e+01   0.000000e+00      0s
##        5    7.4625000e+00   0.000000e+00   0.000000e+00      0s
##
## Solved in 5 iterations and 0.00 seconds
## Optimal objective  7.462500000e+00
```

```
print('---------------------------------')
```

```
## ---------------------------------
```

```
print('\nOptimal solution:\n')
```

```
##
## Optimal solution:
```

```
print('Variable Information Including Sensitivity Information:')

# tVars = PrettyTable(['Variable Name', ' Value', 'ReducedCost',
#                      ' SensLow', ' SensUp'])  #column headers
```

```
## Variable Information Including Sensitivity Information:
```

```
for v in model.getVars():
    print("%s %s %8.2f %s %8.2f %s %8.2f %s %8.2f" %
              (v.Varname, "=", v.X, ", reduced cost = ", abs(v.RC), ", from coeff = ", v.SAObjLow, "to
    print(" ")
```

```
## pi[1] =     0.15 , reduced cost =      0.00 , from coeff =    176.09 to coeff =    177.51
##
## pi[2] =    -1.00 , reduced cost =      0.15 , from coeff =      -inf to coeff =    176.95
##
## pi[3] =     1.00 , reduced cost =      4.97 , from coeff =    179.23 to coeff =       inf
##
## pi[4] =    -0.27 , reduced cost =      0.00 , from coeff =    163.89 to coeff =    173.67
##
## pi[5] =     1.00 , reduced cost =      2.10 , from coeff =    170.70 to coeff =       inf
##
```

```
## pi[6] =    -1.00 , reduced cost =      0.25 , from coeff =      -inf to coeff =    174.35
##
## pi[7] =     0.12 , reduced cost =      0.00 , from coeff =    178.40 to coeff =    181.00
##
```

```
print('\nOptimal objective value: %g' % model.objVal)
```

```
##
## Optimal objective value: 7.4625
```

```
print('\nOptimal shadow prices:\n')
```

```
##
## Optimal shadow prices:
```

```
for c in model.getConstrs():
        print("%s %s %8.2f %s %8.2f %s %8.2f" % (c.ConstrName, ": shadow price = ", c.Pi, ", from RHS =
        print(" ")
```

```
## b0 : shadow price =     63.97 , from RHS =     -0.05 to RHS =      0.07
##
## b1 : shadow price =      4.54 , from RHS =     -1.39 to RHS =      0.79
##
## b2 : shadow price =      1.03 , from RHS =     -2.55 to RHS =      3.15
##
```

```
print('Model Status: '+str(model.status))
```

```
## Model Status: 2
```

```
print('Number of basic solutions: '+str(model.SolCount))
```

```
## Number of basic solutions: 1
```

The formulation of the dual has us find a vector that is perpendicular to each of the variables (including an row of ones to include the constant term) while maximizing its projection to the target variable.

The solution of the original problem i.e the coefficients of the regression problem, can be recovered as they are shadow price of each of the constant as seen in the output, that is:

$$H = 63.97 + 4.54G + 1.03S$$