


# Sensitivity Analysis with Gurobi

Feb 11, 2018 · 10 min read ·  prescriptive analytics (/categories/prescriptive-analytics)



([https://twitter.com/intent/tweet?](https://twitter.com/intent/tweet?text=Sensitivity%20Analysis%20with%20Gurobi&url=%2fpost%2fsensitivity-analysis-with-gurobi%2f)

[text=Sensitivity%20Analysis%20with%20Gurobi&url=%2fpost%2fsensitivity-analysis-with-gurobi%2f](https://twitter.com/intent/tweet?text=Sensitivity%20Analysis%20with%20Gurobi&url=%2fpost%2fsensitivity-analysis-with-gurobi%2f))



(<https://www.facebook.com/sharer.php?u=%2fpost%2fsensitivity-analysis-with-gurobi%2f>)



(<https://www.linkedin.com/shareArticle?mini=true&url=%2fpost%2fsensitivity-analysis-with-gurobi%2f&title=Sensitivity%20Analysis%20with%20Gurobi>)



(<http://service.weibo.com/share/share.php?url=%2fpost%2fsensitivity-analysis-with-gurobi%2f&title=Sensitivity%20Analysis%20with%20Gurobi>)



(<mailto:?subject=Sensitivity%20Analysis%20with%20Gurobi&body=%2fpost%2fsensitivity-analysis-with-gurobi%2f>)

The problem is once again taken from Rardin's book *Optimization in Operations Research (2nd Ed)*. It's a simple problem that illustrates how to perform Sensitivity Analysis with Gurobi and Python.

## Problem 6-25

*The NCAA is making plans for distributing tickets to the upcoming regional basketball championships. The up to 10,000 available seats will be divided between the media, the competing universities, and the general public. Media people are admitted free, but the NCAA receives \$45 per ticket from universities and \$100 per ticket from the general*

public. At least 500 tickets must be reserved for the media, and at least half as many tickets should go to the competing universities as to the general public. Within these restrictions, the NCAA wishes to find the allocation that raises the most money.

The optimization problem can be set up as:

$$\begin{aligned} \max \quad & 45x_2 + 100x_3 \\ x_1 + x_2 + x_3 \leq \quad & 10,000 \\ x_2 - 0.5x_3 \geq \quad & 0 \\ x_1 \geq \quad & 500 \\ x_1, x_2, x_3 \geq \quad & 0 \end{aligned}$$

(a) What is the marginal cost to the NCAA of each seat guaranteed the media? (b) Suppose that there is an alternative arrangement of the dome where the games will be played that can provide 15,000 seats. How much additional revenue would be gained from the expanded seating? How much would it be for 20,000 seats?

(c) Since television revenue provides most of the income for NCAA events, another proposal would reduce the price of general public tickets to \$50 . How much revenue would be lost from this change? What if the price were \$30 ?

(d) Media-hating coach Sobby Day wants the NCAA to restrict media seats to 20% of those allocated for universities. Could this policy change the optimal solution? How about 10%?

(e) To accommodate high demand from student supporters of participating universities, the NCAA is considering marketing a new “scrunch seat” that consumes only 80% of a regular bleacher seat but counts fully against the “university Ú half public” rule. Could an optimal solution allocate any such seats at a ticket price of \$35 ? At a price of \$25 ?

Solutions:

```

from gurobipy import *
import pandas as pd
from collections import OrderedDict
# create model
m = Model()
# Add decision variables
# Let x1 : number of media people
# Let x2 : number of students from university
# Let x3 : number of general public
x1 = m.addVar(lb = 0, name = 'media')
x2 = m.addVar(lb = 0, name = 'university')
x3 = m.addVar(lb = 0, name = 'public')
m.update()
# set objective
obj = 45*x2 + 100*x3
m.setObjective(obj, GRB.MAXIMIZE)
# add constraints
con1 = m.addConstr(x1 + x2 + x3 <= 10000, name = 'total seat') # total seats
are less than 10,000
con2 = m.addConstr(x2 - 0.5*x3 >= 0, name = 'seating constraint') #half the
number of seats should go to competing univs
con3 = m.addConstr(x1 >= 500, name = 'reserved for media') #at least 500 sea
ts for media
m.optimize()
#print solution and objective

```

```

## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
##   Matrix range      [5e-01, 1e+00]
##   Objective range   [5e+01, 1e+02]
##   Bounds range      [0e+00, 0e+00]
##   RHS range         [5e+02, 1e+04]
## Presolve removed 3 rows and 3 columns
## Presolve time: 0.00s
## Presolve: All rows and columns removed
## Iteration    Objective          Primal Inf.    Dual Inf.      Time
##           0      7.7583333e+05    0.000000e+00    0.000000e+00    0s
##
## Solved in 0 iterations and 0.00 seconds
## Optimal objective  7.75833333e+05

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```
## media 500.0
## university 3166.666666666665
## public 6333.333333333334
```

```
print('Obj:', m.objVal)
#### Part (a) ####
```

```
## Obj: 775833.3333333334
```

```
target_cell = {'Name':['Cost'],'Cost':[m.ObjVal]}
# decision variable table
decision_var = OrderedDict([
    ('Name',['media','university','public']),
    ('Final Value',[x1.x,x2.x,x3.x]),
    ('Reduced Cost',[x1.RC,x2.RC,x3.RC]),
    ('Objective Coefficient',[0,45,100]),
    ('Allowable Coeff Increase',[x1.SAObjUp,x2.SAObjUp,x3.SAObjUp]),
    ('Allowable Coeff Decrease',[x1.SAObjLow,x2.SAObjLow,x3.SAObjLow]),
    ('Lower Bound',[x1.LB,x2.LB,x3.LB]),
    ('Upper Bound',[x1.UB,x2.UB,x3.UB])
])
# constraint table
constraint = OrderedDict([
    ('Name',['total seat','seat constraint','reserved for media']),
    ('Shadow Price',[con1.Pi,con2.Pi,con3.Pi]),
    ('RHS Coeff',[1e4,0,500]),
    ('Slack',[con1.Slack,con2.Slack,con3.Slack]),
    ('Lower Range',[con1.SARHSLow,con2.SARHSLow,con3.SARHSLow]),
    ('Upper Range',[con1.SARHSUp,con2.SARHSUp,con3.SARHSUp])
])
print(pd.DataFrame.from_dict(target_cell))
```

```
##          Cost Name
## 0  775833.333333 Cost
```

```
print(pd.DataFrame.from_dict(decision_var))
```

```
##          Name Final Value Reduced Cost Objective Coefficient |
## 0      media  500.000000          0.0              0
## 1 university 3166.666667          0.0             45
## 2    public 6333.333333          0.0            100
##
## Allowable Coeff Increase Allowable Coeff Decrease Lower Bound |
## 0          8.166667e+01      -1.000000e+100          0.0
## 1          1.000000e+02      -2.000000e+02          0.0
## 2          1.000000e+100       4.500000e+01          0.0
##
##      Upper Bound
## 0  1.000000e+100
## 1  1.000000e+100
## 2  1.000000e+100
```

```
print(pd.DataFrame.from_dict(constraint))
#### Part (a) ####
## Changing the seating arrangement to 15,000
```

```
##          Name Shadow Price RHS Coeff Slack Lower Range |
## 0      total seat  81.666667  10000.0  0.0      500.0
## 1    seat constraint -36.666667    0.0  0.0     -4750.0
## 2 reserved for media -81.666667   500.0  0.0       -0.0
##
##      Upper Range
## 0  1.000000e+100
## 1  9.500000e+03
## 2  1.000000e+04
```

```
con1.RHS = 15000
# update the new RHS of constraint 1
m.update()
m.optimize()
#print solution and objective
```

```

## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
## Matrix range [5e-01, 1e+00]
## Objective range [5e+01, 1e+02]
## Bounds range [0e+00, 0e+00]
## RHS range [5e+02, 2e+04]
## Iteration Objective Primal Inf. Dual Inf. Time
## 0 1.1841667e+06 0.000000e+00 0.000000e+00 0s
##
## Solved in 0 iterations and 0.00 seconds
## Optimal objective 1.184166667e+06

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```

## media 500.0
## university 4833.333333333333
## public 9666.666666666668

```

```

print('Obj:', m.objVal)
# constraint table

```

```

## Obj: 1184166.6666666667

```

```

constraint = OrderedDict([
    ('Name', ['total seat', 'seat constraint', 'reserved for media']),
    ('Shadow Price', [con1.Pi, con2.Pi, con3.Pi]),
    ('RHS Coeff', [con1.RHS, con2.RHS, con3.RHS]),
    ('Slack', [con1.Slack, con2.Slack, con3.Slack]),
    ('Lower Range', [con1.SARHSLow, con2.SARHSLow, con3.SARHSLow]),
    ('Upper Range', [con1.SARHSUp, con2.SARHSUp, con3.SARHSUp])
])
print(pd.DataFrame.from_dict(constraint))
## The shadow price of constraint 1 is once again 81.6. So for 5000 more uni
ts, there would be 81.6*5000 dollar increase or 40,800 dollar increase. Ther
efore the new objective should be 1184166 + 40,800 = 1592166 approximately
# revert back

```

##	Name	Shadow Price	RHS Coeff	Slack	Lower Range	Upper Range
## 0	total seat	81.666667	15000.0	0.0	500.0	
## 1	seat constraint	-36.666667	0.0	0.0	-7250.0	
## 2	reserved for media	-81.666667	500.0	0.0	-0.0	
##						
##	Upper Range					
## 0		1.000000e+100				
## 1		1.450000e+04				
## 2		1.500000e+04				

```

con1.RHS = 10000
m.update()
## Changing the seating arrangement to 20,000
#change again
con1.RHS = 20000
m.optimize()
#print solution and objective

```

```

## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
## Matrix range [5e-01, 1e+00]
## Objective range [5e+01, 1e+02]
## Bounds range [0e+00, 0e+00]
## RHS range [5e+02, 2e+04]

```

##	Iteration	Objective	Primal Inf.	Dual Inf.	Time
##	0	1.5925000e+06	0.000000e+00	0.000000e+00	0s

```

##
## Solved in 0 iterations and 0.00 seconds
## Optimal objective 1.592500000e+06

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```

## media 500.0
## university 6500.0
## public 13000.0

```

```
print('Obj:', m.objVal)
# Close enough!
#revert back
```

```
## Obj: 1592500.0
```

```
con1.RHS = 10000
m.update()
#### Part (c) ####
# set new objective
obj = 45*x2 + 50*x3
m.setObjective(obj, GRB.MAXIMIZE)
m.optimize()
#print solution and objective
```

```
## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
##   Matrix range      [5e-01, 1e+00]
##   Objective range   [5e+01, 5e+01]
##   Bounds range      [0e+00, 0e+00]
##   RHS range         [5e+02, 1e+04]
## Iteration    Objective    Primal Inf.    Dual Inf.    Time
##           0      4.5916667e+05    0.000000e+00    0.000000e+00    0s
##
## Solved in 0 iterations and 0.00 seconds
## Optimal objective 4.591666667e+05
```

```
for i in m.getVars():
    print(i.varName, i.x)
```

```
## media 500.0
## university 3166.6666666666665
## public 6333.3333333333334
```



```

print('Obj:', m.objVal)
## Reducing the cost for public by 50 would reduce the objectice by 50*633.33 dollar or by 316667 dollars. The new objcetive value should be 775833.33333334 - 316667 = 459166.33 approximately. This matches the result above.
# set new objective

```

```

## Obj: 459166.6666666667

```

```

obj = 45*x2 + 30*x3
m.setObjective(obj, GRB.MAXIMIZE)
m.optimize()
#print solution and objective

```

```

## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
##   Matrix range      [5e-01, 1e+00]
##   Objective range   [3e+01, 5e+01]
##   Bounds range      [0e+00, 0e+00]
##   RHS range         [5e+02, 1e+04]
## Iteration   Objective      Primal Inf.    Dual Inf.      Time
##           0    1.0000000e+31  6.666667e+29  1.000000e+01    0s
##           1    4.2750000e+05  0.000000e+00  0.000000e+00    0s
##
## Solved in 1 iterations and 0.00 seconds
## Optimal objective  4.275000000e+05

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```

## media 500.0
## university 9500.0
## public 0.0

```

```
print('Obj:', m.objVal)
# We see above that the reduction is not equal to 20*633.33. This is because
30 is outside the range [45, \inf) for the coefficient of x3. The most reduction
will therefore be only by 5*633.33 or (50 - 45)*633.33 from the previous
objective value. So, the new value should be close to 427500 - 3166 = 424333
dollars approximately.
#revert back to original
```

```
## Obj: 427500.0
```

```
obj = 45*x2 + 100*x3
m.setObjective(obj, GRB.MAXIMIZE)
m.optimize()
#### Part (d) ####
#add new constraints
```

```
## Optimize a model with 3 rows, 3 columns and 6 nonzeros
## Coefficient statistics:
##   Matrix range      [5e-01, 1e+00]
##   Objective range   [5e+01, 1e+02]
##   Bounds range      [0e+00, 0e+00]
##   RHS range         [5e+02, 1e+04]
## Iteration    Objective    Primal Inf.    Dual Inf.    Time
##           0    5.5000000e+31  2.5000000e+30  5.5000000e+01  0s
##           1    7.7583333e+05  0.0000000e+00  0.0000000e+00  0s
##
## Solved in 1 iterations and 0.00 seconds
## Optimal objective  7.758333333e+05
```

```
con4 = m.addConstr(x1 <= 0.2*x2, name = 'Media hating coach')
m.update()
m.optimize()
#print solution and objective
```

```

## Optimize a model with 4 rows, 3 columns and 8 nonzeros
## Coefficient statistics:
## Matrix range [2e-01, 1e+00]
## Objective range [5e+01, 1e+02]
## Bounds range [0e+00, 0e+00]
## RHS range [5e+02, 1e+04]
## Iteration Objective Primal Inf. Dual Inf. Time
## 0 7.7583333e+05 0.000000e+00 0.000000e+00 0s
##
## Solved in 0 iterations and 0.00 seconds
## Optimal objective 7.75833333e+05

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```

## media 500.0
## university 3166.6666666666665
## public 6333.3333333333334

```

```

print('Obj:', m.objVal)
# It does not change the optimal solutions or the objective value. The solution is the same as before. Because this new restriction is satisfied by the old primal solution.
#remove constraint

```

```

## Obj: 775833.33333333334

```

```

m.remove(m.getConstrs()[3])
m.getConstrs()
#add new constraints
con4 = m.addConstr(x1 <= 0.1*x2, name = 'Media hating coach')
m.update()
m.optimize()
#print solution and objective

```

```

## Optimize a model with 4 rows, 3 columns and 8 nonzeros
## Coefficient statistics:
## Matrix range [1e-01, 1e+00]
## Objective range [5e+01, 1e+02]
## Bounds range [0e+00, 0e+00]
## RHS range [5e+02, 1e+04]

| <i>## Iteration</i> | <i>Objective</i>     | <i>Primal Inf.</i>   | <i>Dual Inf.</i>    | <i>Time</i> |
|---------------------|----------------------|----------------------|---------------------|-------------|
| <i>## 0</i>         | <i>7.7583333e+05</i> | <i>1.8333333e+02</i> | <i>0.000000e+00</i> | <i>0s</i>   |
| <i>## 1</i>         | <i>6.7500000e+05</i> | <i>0.000000e+00</i>  | <i>0.000000e+00</i> | <i>0s</i>   |

##
## Solved in 1 iterations and 0.00 seconds
## Optimal objective 6.750000000e+05

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```

## media 500.0
## university 5000.0
## public 4500.0

```

```

print('Obj:', m.objVal)
## In this case the solution changes. We see that the new constraint was not
satisfied by the old primal solution. We also get new solutions.
# remove the constraint again

```

```

## Obj: 675000.0

```

```

m.remove(m.getConstrs()[3])
m.getConstrs()
#### Part (d) ####
#lets add a new decision variable to the problem : x4
x4 = m.addVar(lb = 1, name = 'scrunch seat')
m.update()
#lets remove all constraints and add new ones to accomodate the new decision
var
m.remove(m.getConstrs()[0:3])
m.getConstrs()
# set new objective
obj = 45*x2 + 100*x3 + 35*x4
m.setObjective(obj, GRB.MAXIMIZE)
# add constraints
con1 = m.addConstr(x1 + x2 + x3 + 0.8*x4 <= 1e4, name = 'total seat')
con2 = m.addConstr(x1 >= 500, name = 'reserved for media')
con3 = m.addConstr(x2 - 0.5*x3 + x4 >= 0, name = 'seating constraint')
m.optimize()
#print solution and objective

```

```

## Optimize a model with 3 rows, 4 columns and 8 nonzeros
## Coefficient statistics:
##   Matrix range      [5e-01, 1e+00]
##   Objective range   [4e+01, 1e+02]
##   Bounds range      [1e+00, 1e+00]
##   RHS range         [5e+02, 1e+04]
## Presolve removed 1 rows and 1 columns
## Presolve time: 0.00s
## Presolved: 2 rows, 3 columns, 6 nonzeros
##
## Iteration    Objective          Primal Inf.    Dual Inf.      Time
##           0    9.4995500e+05    5.935750e+02    0.000000e+00    0s
##           1    7.9732143e+05    0.000000e+00    0.000000e+00    0s
##
## Solved in 1 iterations and 0.00 seconds
## Optimal objective  7.973214286e+05

```

```

for i in m.getVars():
    print(i.varName, i.x)

```

```
## media 500.0
## university 0.0
## public 6785.714285714286
## scrunch seat 3392.857142857143
```

```
print('Obj:', m.objVal)
# constraint table
```

```
## Obj: 797321.4285714286
```

```
constraint = OrderedDict([
    ('Name', ['total seat', 'reserved for media', 'seat constraint']),
    ('Shadow Price', [con1.Pi, con2.Pi, con3.Pi]),
    ('RHS Coeff', [1e4, 0, 500]),
    ('Slack', [con1.Slack, con2.Slack, con3.Slack]),
    ('Lower Range', [con1.SARHSLow, con2.SARHSLow, con3.SARHSLow]),
    ('Upper Range', [con1.SARHSUp, con2.SARHSUp, con3.SARHSUp])
])
print(pd.DataFrame.from_dict(constraint))
```

```
##           Name  Shadow Price  RHS Coeff  Slack  Lower Range  |
## 0      total seat      83.928571    10000.0    0.0        502.8
## 1  reserved for media    -83.928571         0.0    0.0         -0.0
## 2      seat constraint    -32.142857     500.0    0.0       -4748.6
##
##      Upper Range
## 0  1.000000e+100
## 1  9.997200e+03
## 2  1.187500e+04
```

[Linear programming \(/tags/linear-programming\)](/tags/linear-programming/)

[Python \(/tags/python\)](/tags/python/)

## Related

- [Optimization with Gurobi \(/post/optimization-with-gurobi/\)](/post/optimization-with-gurobi/)
- [Multi-period Optimization with Gurobi \(/post/multi-period-optimization-with-gurobi/\)](/post/multi-period-optimization-with-gurobi/)

