HW5_Lentz

October 9, 2017

0.0.1 Purpose: This notebook demonstrates the solution of linear programming problems using Python and Pulp

0.0.2 Problems are from 'Introduction to Operations Research' Tenth Edition by Fredrick S. Hillier and Gerald J. Lieberman

References http://coral.ie.lehigh.edu/~ted/files/talks/PythonModeling.pdf http://ksuweb.kennesaw.edu/~jgarrido/CS4491_notes/Sensitivity_rep.pdf

Problem 4.7-3

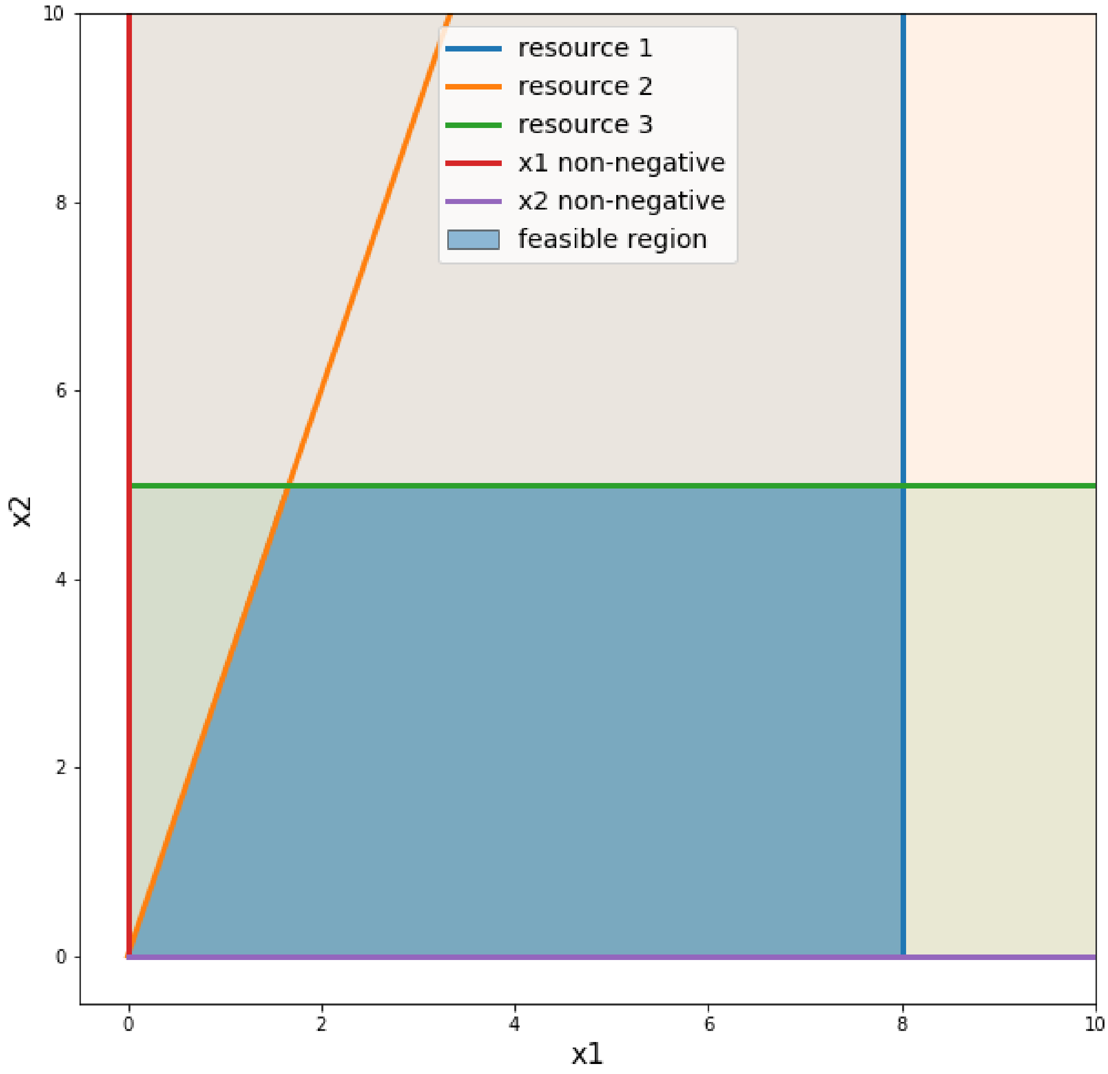
In [91]:
```
# get your instance setup
import os
#os.system("brew install glpk") # or sudo apt install glpk on linux
#os.system("pip install pulp")
from pulp import *

prob = LpProblem("p4.7-3", LpMaximize)

# Variables, lower bounds added here
x1 = LpVariable("x1",lowBound=0)
x2 = LpVariable("x2",lowBound=0)
z = LpVariable("z", 0)

# Objective
prob += 4*x1 + 2*x2

# Constraints
prob += 2*x1 <= 16 # resource 1
prob += (1*x1 + 3*x2) <= 17 # resource 2
prob += x2 <= 5 # resource 3

# Solve using GLPK
GLPK().solve(prob)

# Solution
for v in prob.variables():
    print (v.name, "=", v.varValue)
```

```python
        print ("objective=", value(prob.objective) )
```

```
('x1', '=', 8.0)
('x2', '=', 3.0)
('objective=', 38.0)
```

In [97]:
```python
# Next, we will solve this problem graphically
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib.path import Path
from matplotlib.patches import PathPatch

# use seaborn to change the default graphics to something nicer
# and set a nice color palette
import seaborn as sns

# create the plot object
fig, ax = plt.subplots(figsize=(10, 10))
x1 = np.linspace(0, 10)
x2 = np.linspace(0, 10)

# add resource 1 constraint: prob += 2*x1 <= 16 # resource 1
plt.plot( 8 * np.ones_like(x1), x1, lw=3, label='resource 1' )
plt.fill_between( np.linspace(0, 8), 0, 100, alpha=0.1 )

# add resource 2 constraint: 1*x1+3*x2 <= 17 # resource 2
plt.plot(x1, 3*x2, lw=3, label='resource 2')
plt.fill_between(np.linspace(0, 17), 0, 17, alpha=0.1)

# add resource 3 constraint: x2 <= 5 # resource 3
plt.plot(x2, 5 * np.ones_like(x2), lw=3, label='resource 3')
plt.fill_between(x1, 0, 5, alpha=0.1)

# add non-negativity constraints
plt.plot(np.zeros_like(s), s, lw=3, label='x1 non-negative')
plt.plot(s, np.zeros_like(s), lw=3, label='x2 non-negative')

# highlight the feasible region
path = Path([
    (0., 0.),
    (1.65, 5.),
    (8., 5.),
    (8., 0.),
    (0., 0.),
])
patch = PathPatch(path, label='feasible region', alpha=0.5)
```

```
ax.add_patch(patch)

# labels and stuff
plt.xlabel('x1', fontsize=16)
plt.ylabel('x2', fontsize=16)
plt.xlim(-0.5, 10)
plt.ylim(-0.5, 10)
plt.legend(fontsize=14)
plt.show()
```



In the above plot, we can see the feasible region shaded in blue using the graphical method.

Problem 4.7-6c

```
In [128]: sensitivity_file_name = 'sensit1.sen'
          prob = LpProblem("p4.7-6c", LpMaximize)
```

```
# Variables, lower bounds added here
x1 = LpVariable("x1",lowBound=0)
x2 = LpVariable("x2",lowBound=0)
x3 = LpVariable("x3",lowBound=0)
x4 = LpVariable("x4",lowBound=0)
z = LpVariable("z", 0)

# Objective
prob += 5*x1 + 4*x2 + -1*x3 + 3*x4, "Objective"

# Constraints
prob += 3*x1 +2*x2 -3*x3 + 1*x4 <= 24 , "resource constraint 1"
prob += 3*x1 +3*x2 +1*x3 + 3*x4 <= 36 , "resource constraint 2"

# save the linear program model
prob.writeLP("p4-7-6c.lp")

# Solve the optimization problem using the specified Solver
GLPK(options=['--ranges', sensitivity_file_name]).solve(prob)

# Solution
print ("Status:", LpStatus[prob.status])
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )
```

```
('Status:', 'Optimal')
('x1', '=', 11.0)
('x2', '=', 0.0)
('x3', '=', 3.0)
('x4', '=', 0.0)
('objective=', 52.0)
```

Next we perform a review of the sensitivity analysis report. Before we look at the report, lets review what the column names mean.

```
In [127]: from IPython.display import Image
          Image(filename='sen.png')
```

```
Out[127]:
```

Columns

Header	Values	Comment
`No`		ordinal number of column, 1 … *m*
`Column name`		symbolic name (blank if none)
`St`		column status
	`BS`	basic column
	`NL`	non-basic column with lower bound active
	`NU`	non-basic column with upper bound active
	`NS`	non-basic fixed column
	`NF`	non-basic free (unbounded) column
`Activity`		(primal) value of structural variable
`Obj coeff`		objective coefficient for structural variable
`Marginal`		reduced cost (dual activity) of structural variable
`Lower bound`		lower bound on structural variable (`-Inf` if open)
`Upper bound`		upper bound on structural variable (`+Inf` if open)
`Obj value`		objective value
`Limiting variable`		name of limiting variable

Abbreviations: RHS means right-hand side. Inf means infinity.

In [129]:
```
print('')
with open(sensitivity_file_name) as f:
    data = f.read()
    print(data)
```

```
GLPK 4.63 - SENSITIVITY ANALYSIS REPORT

Problem:
Objective:  Objective = 52 (MAXimum)

   No. Row name     St      Activity         Slack   Lower bound       Activity      Obj coef  O
                                          Marginal   Upper bound          range         range
------ ------------ -- ------------- ------------- -------------  ------------- ------------- --
     1 resource_constraint_1
                    NU      24.00000         .              -Inf     -108.00000       -.66667
                                            .66667      24.00000       36.00000          +Inf
```

```
     2 resource_constraint_2
                    NU      36.00000        .              -Inf       24.00000      -1.00000
                                       1.00000      36.00000          +Inf          +Inf

GLPK 4.63 - SENSITIVITY ANALYSIS REPORT

Problem:
Objective:  Objective = 52 (MAXimum)

   No. Column name  St      Activity      Obj coef   Lower bound       Activity      Obj coef  O
                                          Marginal   Upper bound          range         range
------ ------------ -- ------------- ------------- -------------  ------------- ------------- --
     1 x1           BS      11.00000       5.00000        .              .            4.63636
                                           .              +Inf       11.00000          +Inf

     2 x2           NL        .            4.00000        .              -Inf          -Inf
                                          -.33333         +Inf       12.00000       4.33333

     3 x3           BS       3.00000      -1.00000        .            -3.60000      -2.33333
                                           .              +Inf       36.00000       1.66667

     4 x4           NL        .            3.00000        .              -Inf          -Inf
                                          -.66667         +Inf        6.00000       3.66667

End of report
```

The range of values of the coefficients in the objective function appear in column 'Obj coef range', for the three decision variables x1, x2, x3, and x4. These ranges of values corresponding to the coefficients of variables that retain the optimal value of the objective function. For example, the value of the coefficient of x1 can range from 4.63636 to a very high value, the coefficient of x2 can range from a very low value up to 4.33333, the coefficient of variable x3 can range from -2.33333 to 1.66667 and the coefficient of variable x4 can range from a very low value to 3.66667. Any values of the coefficient outside these ranges will change the conditions of the objective function to a suboptimal value.

Variable x2 has a reduced cost of .33333 and is shown in column 'Marginal.' This is the amount the objective function would change if the value of x2 is changed to 1. Likewise, variable x4 has a reduced cost of -.66667. This is the amount the objective function would change if the value of x4 is changed to 1. Variables x1 and x3 are basic variables (as indicated by NL in the State or St column) and have a zero reduced cost.

A one unit increase in resource_constraint_1 increases z by .66667
A one unit increase in resource_constraint_2 increases z by 1.00000
Thus the shadow prices are .66667 and 1.00000 respectively.

The objective functions have the following allowable range:
x1 between 4.63636 and +Inf
x2 between -Inf and 4.33333

x3 between -2.33333 and 1.66667
x4 between -Inf and 3.66667

The right-hand side has the following allowable range:
resource_constraint_1 from -.66667 to +Inf
resource_constraint_1 from -1.00000 to +Inf

Problem 7.3-4 (a, f, g, h)

```
In [131]: from IPython.display import Image
          Image(filename='7-3-4.png')
```

Out[131]:

		P 7.3-4				
		Resource Usage per Unit of Each Activity				
			Activity			
Resource	Produce Toys		Produce Subs		Amt of Re	Remaining
Sub A	2	2000	-1	1000	3000	0
Sub B	1	2000	-1	1000	1000	0
Unit Profit	3		-2.5			
	mak_toys	2000				
	mak_subs	1000				
		objective	3500			

f It is optimal to produce 2000 toys and make 1000 subcomponents.
If we increased the unit profit of making subcomponents 5.5 or more units or toys by 0 or more units, the optimal solution changes.
The reduced costs tell us how much the objective coefficients (unit profits) can be increased or decreased before the optimal solution changes.
Shadow prices tell us how much the optimal solution can be increased or decreased if we change the right hand side values with one unit.

g Toy profits values (2.0, 2.5,3.0,3.5,4.0) and subs (-3.5,-3.0,-2.5,-2.0,-1.5)

Two way t	3500	-3.5	-3	-2.5	-2	-1.5
	2	500	1000	1500	2000	2500
	2.5	1500	2000	2500	3000	3500
	3	2500	3000	3500	4000	4500
	3.5	3500	4000	4500	5000	5500
	4	4500	5000	5500	6000	6500

h

Within the context of the ranges of cost for the subcomponents, there are values that yield a superior return. These are shown above in green.
The inputs that yield a similar return are shown in yellow. Those inputs that are shown in red represent an inferior return.

```
In [132]: from IPython.display import Image
          Image(filename='7-3-4sens.png')
```

Out[132]:

M 1048576R x 1C 15.0 Sensitivity Report
Worksheet: [HW_5_LENTZ.xlsx]p3.5.6
Report Created: 10/7/2017 11:02:00 AM

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C8	mak_toys Resource Usage per Unit of Each Activity	2000	0	3	2	0.5
C9	mak_subs Resource Usage per Unit of Each Activity	1000	0	-2.5	1	0.5

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
G5	Sub A Remaining	0	-0.5	0	1000	1E+30
G6	Sub B Remaining	0	-2	0	1E+30	500

Problem 7.3-5 (a, b, f)

```
In [140]: from IPython.display import Image
          Image(filename='7-3-5.png')
```

Out[140]:

	A	B	C	D	E	F	G	H
1			P 7.3-4					
2			Resource Usage per Unit of Each Activity					
3				Activity				
4	Resource	Produce Toys		Produce Subs		Amt of Re	Remaining	
5	Sub A	2	1751	-1	750.5	3000	250	
6	Sub B	1	1751	-1	750.5	1000	0	
7	Unit Profit	3		-2.5				
8		mak_toys	1751					
9		mak_subs	750.5			total units	2501	
Height: 15.00 (20 pixels)			objective	3375.25		max	2501	
11								

```
In [138]: from IPython.display import Image
          Image(filename='7-3-5sens2500.png')
```

Out[138]:

Microsoft Excel 15.0 Sensitivity Report
Worksheet: [HW_7_3_5_LENTZ.xlsx]p7.3.5
Report Created: 10/7/2017 3:42:43 PM

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C8	mak_toys Resource Usage per Unit of Each Activity	1750	0	3	1E+30	0.5
C9	mak_subs Resource Usage per Unit of Each Activity	750	0	-2.5	5.5	0.5

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
G5	Sub A Remaining	250	0	0	250	1E+30
G6	Sub B Remaining	0	-2.75	0	3500	166.6666667
G9	total units Remaining	2500	0.25	2500	500	1500

b) Therefore, unless the market will bear more than 2500 toys at a $3 per unit profit, we should not pay a premimum for part A and up to 2.5 for part B. As shown, the shadow prices tell us how much the items increase or decrease if we alter the RHS by one unit. Here we see that the shadow price is 0 for sub A and -2.75 for sub B due to the limiting constrain of 2500. This is the same for 2501, etc. until we exhaust the supply currently offered.

c) Therefore, unless the market will bear more than 2500 toys at a $3 per unit profit, we should not pay a premimum for part A and up to 2.5 for part B.

In [139]:
```python
from IPython.display import Image
Image(filename='7-3-5sens2501.png')
```

Out[139]:

Microsoft Excel 15.0 Sensitivity Report
Worksheet: [HW_7_3_5_LENTZ.xlsx]p7.3.5
Report Created: 10/7/2017 3:43:51 PM

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C8	mak_toys Resource Usage per Unit of Each Activity	1750.5	0	3	1E+30	0.5
C9	mak_subs Resource Usage per Unit of Each Activity	750.5	0	-2.5	5.5	0.5

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
G5	Sub A Remaining	249.5	0	0	249.5	1E+30
G6	Sub B Remaining	0	-2.75	0	3501	166.3333333
G9	total units Remaining	2501	0.25	2501	499	1501

Problem 7.3-7 (a)

```
In [3]: from IPython.display import Image
        Image(filename='7-3-7a.png')
```

Out[3]:

Union Airways Personnel Scheduling Problem

	6am-2pm Shift	8am-4pm Shift	Noon-8pm Shift	4pm-midnight Shift	10pm-6am Shift				
Cost per Shift	$170	$160	$175	$180	$195				
						Total		Minimum	
Time Period	Shift Works Time Period? (1=yes, 0=no)					Working		Needed	Inc?
6am-8am	1	0	0	0	0	48	>=	48	x
8am-10am	1	1	0	0	0	79	>=	79	0
10am- 12pm	1	1	0	0	0	79	>=	65	x
12pm-2pm	1	1	1	0	0	117	>=	87	x
2pm-4pm	0	1	1	0	0	69	>=	64	x
4pm-6pm	0	0	1	1	0	82	>=	73	x
6pm-8pm	0	0	1	1	0	82	>=	82	0
8pm-10pm	0	0	0	1	0	44	>=	44	x
10pm-12am	0	0	0	1	1	59	>=	52	x
12am-6am	0	0	0	0	1	15	>=	15	0

	6am-2pm Shift	8am-4pm Shift	Noon-8pm Shift	4pm-midnight Shift	10pm-6am Shift		Total Cost
Number Working	48	31	38	44	15		$30,615

Range Name	Cells
CostPerShift	C5:G5
MinimumNeeded	J8:J17
NumberWorking	C21:G21
ShiftWorksTimePeriod	C8:G17
TotalCost	J21
TotalWorking	H8:H17

Above we see the solver based Union Airways problem with the added sensitivity column (Inc?). Here an X means we can add without increasing Total Cost.

```
In [2]: from IPython.display import Image
        Image(filename='7-3-7a_sens.png')
```

Out[2]:

Microsoft Excel 15.0 Sensitivity Report
Worksheet: [HW_5_LENTZ_p7_3_7.xls]Union Airways
Report Created: 10/7/2017 8:23:41 PM

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
C21	Number Working Shift	48	0	170	1E+30	10
D21	Number Working Shift	31	0	160	10	160
E21	Number Working Shift	39	0	175	5	175
F21	Number Working Shift	43	0	180	1E+30	5
G21	Number Working Shift	15	0	195	1E+30	195

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
H8	6am-8am Working	48	10	48	6	48
H9	8am-10am Working	79	160	79	1E+30	6
H10	10am- 12pm Working	79	0	65	14	1E+30
H11	12pm-2pm Working	118	0	87	31	1E+30
H12	2pm-4pm Working	70	0	64	6	1E+30
H13	4pm-6pm Working	82	0	73	9	1E+30
H14	6pm-8pm Working	82	175	82	1E+30	6
H15	8pm-10pm Working	43	5	43	6	6
H16	10pm-12am Working	58	0	52	6	1E+30
H17	12am-6am Working	15	195	15	1E+30	6

The above highlighted selection gives the range of the increase before the optimized Total Cost will be adversely impacted. This range reflects line management's operational model flexibility. Note, the three 1E+30s here mean that the shift cannot have any increase in 'Minimum number of Agents Needed' without impacting Total Cost.

Problem 7.4-4

```
In [9]: from pulp import *

        prob = LpProblem("p7.4-4a", LpMaximize)

        # The estimates and ranges of uncertainty for the uncertain parameters are
        # defined in the next block.
        a11 = 4
        a22 = -1
        a33 = 3
        b1  = 30
        b2  = 20
        c2  = -8
```

```
c3 = 4

# Variables, lower bounds added here
x1 = LpVariable("x1",lowBound=0)
x2 = LpVariable("x2",lowBound=0)
x3 = LpVariable("x3",lowBound=0)

z = LpVariable("z", 0)

# Objective
prob += 5*x1 + c2*x2 + c3*x3

# Constraints
prob += a11*x1 -3*x2 + 2*x3 <= b1 # resource 1
prob += 3*x1 + a22*x2 + x3 <= b2 # resource 2
prob += 2*x1 -4*x2 + a33 * x3 <= 20 # resource 3

# Solve using GLPK
GLPK().solve(prob)

# Solution
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )
```

```
('x1', '=', 5.71429)
('x2', '=', 0.0)
('x3', '=', 2.85714)
('objective=', 40.000009999999996)
```

a) The below output indicates the objective value and the values for x1, x2, and x3

('x1', '=', 5.71429) ('x2', '=', 0.0) ('x3', '=', 2.85714) ('objective=', 40.000009999999996)

```
In [1]: from pulp import *
        import numpy as np
        import datetime
        from functools import reduce
        from operator import mul
        import math


        how_many_minutes_we_can_wait = 1
        characterized_performance_ops = 36000
        number_of_parameter_dimensions = 7
        steps_per_range = math.pow( (how_many_minutes_we_can_wait * characterized_performance_op
```

```
# if using distributions, random selection, and time
now = datetime.datetime.now()
stop_time = now + datetime.timedelta(minutes=1)

max_objective_value = float("-Inf")

a11_best = 0.0
a22_best = 0.0
a33_best = 0.0
b1_best = 0.0
b2_best = 0.0
c2_best = 0.0
c3_best = 0.0

a11_range = np.linspace(3.6, 4.4, num=math.floor(steps_per_range))
a22_range = np.linspace(-1.4, -0.6, num=math.floor(steps_per_range))
a33_range = np.linspace(2.5, 3.5, num=math.floor(steps_per_range))
b1_range = np.linspace(27.0, 33.0, num=math.floor(steps_per_range))
b2_range = np.linspace(19.0, 22.0, num=math.floor(steps_per_range))
c2_range = np.linspace(-9.0, -7.0, num=math.floor(steps_per_range))
c3_range = np.linspace(3.0, 5.0, num=math.floor(steps_per_range))

# this is okay if we can use huerisitics
# a11_range = np.arange(3.6, 4.4, 0.1)
# a22_range = np.arange(-1.4, -0.6, 0.1)
# a33_range = np.arange(2.5, 3.5, 0.1)
# b1_range = np.arange(27.0, 33.0, 0.1)
# b2_range = np.arange(19.0, 22.0, 0.1)
# c2_range = np.arange(-9.0, -7.0, 0.1)
# c3_range = np.arange(3.0, 5.0, 0.1)

parameter_optimization_ranges = [a11_range, a22_range, a33_range, b1_range, b2_range, c2
# total_operations_count = reduce(mul, [ len(rng) for rng in parameter_optimization_rang
# print("this will take " + str(total_operations_count/36000) + " minutes.")
# exit()

counter = 0

# Variables, lower bounds added here
x1 = LpVariable("x1",lowBound=0)
x2 = LpVariable("x2",lowBound=0)
x3 = LpVariable("x3",lowBound=0)

z = LpVariable("z", 0)

# loop over parameter ranges
for a11 in a11_range:
    for a22 in a22_range:
```

```
        for a33 in a33_range:
            for b1 in b1_range:
                for b2 in b2_range:
                    for c2 in c2_range:
                        for c3 in c3_range:
                            counter = counter + 1

                            prob = LpProblem("p7.4-4b", LpMaximize)

                            # Objective
                            prob += 5*x1 + c2*x2 + c3*x3

                            # Constraints
                            prob += a11*x1 -3*x2 + 2*x3 <= b1 # resource 1
                            prob += 3*x1 + a22*x2 + x3 <= b2 # resource 2
                            prob += 2*x1 -4*x2 + a33 * x3 <= 20 # resource 3

                            # Solve using GLPK
                            GLPK().solve(prob)

                            # If solution is better, update best parameters
                            if value(prob.objective) > max_objective_value:
                                max_objective_value = value(prob.objective)
                                a11_best = a11
                                a22_best = a22
                                a33_best = a33
                                b1_best = b1
                                b2_best = b2
                                c2_best = c2
                                c3_best = c3

                #if datetime.datetime.now() >= stop_time:
                #    print(counter)
                #    exit()


# from multiprocessing import Pool
# if __name__ == '__main__':

#     p = Pool(8)
#     p.map(a3dProcess,subject_uuids_list)

# Solve solution again with best parameters
a11 = a11_best
a22 = a22_best
a33 = a33_best
b1 = b1_best
b2 = b2_best
```

```
    c2 = c2_best
    c3 = c3_best
    print("Solving with optimum parameters: ")
    print("a11 " + str(a11))
    print("a22 " + str(a22))
    print("a33 " + str(a33))
    print("b1 " + str(b1))
    print("b2 " + str(b2))
    print("c2 " + str(c2))
    print("c3 " + str(c3))

    for v in prob.variables():
        print (v.name, "=", v.varValue)

    print ("objective=", value(prob.objective) )
```

```
  File "<ipython-input-1-608ef4f70429>", line 1
    break
SyntaxError: 'break' outside loop
```

b) For the robust optimization, I used naive search over the parameter space and a simple computational estimate was used to define paramater range specificity. Here are the robust optimal results, z ~= 200.0:

a11 3.6
a22 -1.4
a33 2.857140873
b1 27.0
b2 22.0
c2 -7.0
c3 5.0
x1 = 0.0
x2 = 15428400.0
x3 = 21599800.0
objective= 200.0

Problem 7.5-1 - Effect of reduced standard deviation of chance constraints on total profit

```
In [15]: from pulp import *
         prob = LpProblem("p7.5-1a", LpMaximize)  # a.k.a Wyndor Glass Co.

         # Variables, lower bounds added here
         x1 = LpVariable("x1",lowBound=0)
         x2 = LpVariable("x2",lowBound=0)
         z = LpVariable("z", 0)
```

```
# Objective
prob += 3*x1 + 2*x2, "Objective"

# Chance Constraints (RHS)
# b1 = 4
# b2 = 12
# b3 = 18

# z_score = 1.645
# #    =  mean - critical value * std_dev
# b1 = 4 - z_score * (.2)
# b2 = 12 - z_score * (.5)
# b3 = 18 - z_score * (.1)


# print("RHS before refinement: " )
# print("z_score " + str(z_score))
# print("b1 "+ str(b1))
# print("b2 "+ str(b2))
# print("b3 "+ str(b3))

z_score = 2.326
#  =  mean - critical value * std_dev
b1 = 4 - z_score * (.1)
b2 = 12 - z_score * (.25)
b3 = 18 - z_score * (.5)

print("RHS after refinement of estimates: " + str(z_score))
print("b1 "+ str(b1))
print("b2 "+ str(b2))
print("b3 "+ str(b3))

# Constraints
prob += x1 <= b1 , "constraint 1"
prob += 2*x2 <= b2 , "constraint 2"
prob += 3*x1 +2*x2 <= b3 , "constraint 3"


# Solve the optimization problem using the specified Solver
GLPK().solve(prob)

# Solution
print ("Status:", LpStatus[prob.status])
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )
```

```
RHS after refinement of estimates: 2.326
b1 3.7674
b2 11.4185
b3 16.837
('Status:', 'Optimal')
('x1', '=', 3.7674)
('x2', '=', 2.7674)
('objective=', 16.837)
```

a) Deterministic equivalents of change constraints before...

RHS before refinement:
z_score 1.645
b1 3.671
b2 11.1775
b3 17.8355
('Status:', 'Optimal')
('x1', '=', 3.671)
('x2', '=', 3.41125)
('objective=', 17.8355)
and deterministic equivalents of change constraints after refinement:
RHS after refinement of estimates: 2.326
b1 3.7674
b2 11.4185
b3 16.837
('Status:', 'Optimal')
('x1', '=', 3.7674)
('x2', '=', 2.7674)
('objective=', 16.837)

b) The total profit is estimated to decrease (17.84 - 16.837) ~= 1 unit based on the refined estimates in increased alpha level.

Thus the total profit per week is expected to decrease by one unit based on the careful investigation that the cut the standard deviations in half.

Problem 7.5-4 - Effect of mutually independent normal distributions on optimization

```
In [8]: from pulp import *
        import scipy.stats as st
        prob = LpProblem("p7.5-4", LpMaximize)

        # Variables, lower bounds added here
        x1 = LpVariable("x1",lowBound=0)
        x2 = LpVariable("x2",lowBound=0)
        x3 = LpVariable("x3",lowBound=0)
        z = LpVariable("z", 0)
```

```
# Objective
prob += 20*x1 + 30*x2 + 25*x3, "Objective"

z_score_b1 = st.norm.ppf(.975)
z_score_b2 = st.norm.ppf(.95)
z_score_b3 = st.norm.ppf(.90)
#  =  mean - critical value * std_dev
b1 = 90 - z_score_b1 * (3)
b2 = 150 - z_score_b2 * (6)
b3 = 180 - z_score_b3 * (9)

print("RHS: ")
print("b1 "+ str(b1))
print("b2 "+ str(b2))
print("b3 "+ str(b3))

# Constraints
prob += 3*x1 + 2*x2 + 1*x3 <= b1 , "constraint 1"
prob += 2*x1 + 4*x2 + 2*x3 <= b2 , "constraint 2"
prob += 1*x1 + 3*x2 + 5*x3 <= b3 , "constraint 3"


# Solve the optimization problem using the specified Solver
GLPK().solve(prob)

# Solution
print ("Status:", LpStatus[prob.status])
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )

# part a
x1 = 7
x2 = 22
x3 = 19

# z_score = (obs - mean) / std
print(3*x1 + 2*x2 + 1*x3)
p_c1 = 1-st.norm.cdf(3*x1 + 2*x2 + 1*x3,90,3)
print( p_c1 )

print(2*x1 + 4*x2 + 2*x3)

p_c2 =  1-st.norm.cdf(2*x1 + 4*x2 + 2*x3,150,6)
print( p_c2 )

print(1*x1 + 3*x2 + 5*x3)
```

```
    p_c3 = 1-st.norm.cdf(1*x1 + 3*x2 + 5*x3,180,9)
    print( p_c3 )

    print("A: Overall p = p(c1)*p(c2)*p(c3) : "+ str(p_c1*p_c2*p_c3))


    # part c
    x1 = 7.02733
    x2 = 21.9645
    x3 = 19.109

    print(3*x1 + 2*x2 + 1*x3)
    p_c1 = 1-st.norm.cdf(3*x1 + 2*x2 + 1*x3,90,3)
    print( p_c1 )


    print(2*x1 + 4*x2 + 2*x3)
    p_c2 = 1-st.norm.cdf(2*x1 + 4*x2 + 2*x3,150,6)
    print( p_c2 )


    print(1*x1 + 3*x2 + 5*x3)
    p_c3 = 1-st.norm.cdf(1*x1 + 3*x2 + 5*x3,180,9)
    print( p_c3 )

    print("C: Overall p = p(c1)*p(c2)*p(c3) : "+ str(p_c1*p_c2*p_c3))
```

```
RHS:
b1 84.1201080464
b2 140.130878238
b3 168.46603591
('Status:', 'Optimal')
('x1', '=', 7.02733)
('x2', '=', 21.9645)
('x3', '=', 19.109)
('objective=', 1277.2066)
84
0.977249868052
140
0.952209647727
168
0.908788780274
A: Overall p = p(c1)*p(c2)*p(c3) : 0.845670448283
84.11999
0.975002299654
140.13066
0.950003751245
168.46583
```

```
0.90000401515
C: Overall p = p(c1)*p(c2)*p(c3) : 0.833633976986
```

a)

$$p(c1) = 0.977249868052$$

$$p(c2) = 0.952209647727$$

$$p(c3) = 0.908788780274$$

Overall prob, part a:

$$p(c1) * p(c2) * p(c3) = 0.845670448283$$

b)

RHS: b1 84.1201080464
b2 140.130878238
b3 168.46603591

Results:
('Status:', 'Optimal')
('x1', '=', 7.02733)
('x2', '=', 21.9645)
('x3', '=', 19.109)
('objective=', 1277.2066)

c)

$$p(c1) = 0.975002299654$$

$$p(c2) = 0.950003751245$$

$$p(c3) = 0.90000401515$$

Overall

$$p = p(c1) * p(c2) * p(c3) = 0.833633976986$$

Problem 7.6-1

```
In [53]: from pulp import *
         prob = LpProblem("p7.5-1a", LpMaximize)   #

         # Variables, lower bounds added here
         x1 = LpVariable("x1",lowBound=0)
         x21 = LpVariable("x21",lowBound=0)
         x22 = LpVariable("x22",lowBound=0)
         z = LpVariable("z", 0)

         # Objective
```

```
p_sitation_1 = .5
p_situation_2 = .5

prob += p_sitation_1*(3*x1 + 5*x21) + p_situation_2*(3*x1 + 1*x22), "Objective"


b1 = 4
b2 = 12
b3 = 18

# Constraints
prob += x1 <= b1 , "plant constraint 1"
prob += 2*x21 <= b2 , "plant constraint 2 sit 1"
prob += 2*x22 <= b2 , "plant constraint 2 sit 2"
prob += 3*x1 +2*x21 <= b3, "plant constraint 3 sit 1"
prob += 3*x1 +6*x22 <= b3, "plant constraint 3 sit 2"
# Solve the optimization problem using the specified Solver
GLPK().solve(prob)

# Solution
print ("Status:", LpStatus[prob.status])
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )
```

```
('Status:', 'Optimal')
('x1', '=', 2.0)
('x21', '=', 6.0)
('x22', '=', 2.0)
('objective=', 22.0)
```

Let's compare the objective results between the prior belief state, where Scenario Two has a probability of 0.75 and the current belief state, where we have updated information stating that the scenarios are equally likely.

p(S2) = .75, p(S1) = .25 ('Status:', 'Optimal')
('x1', '=', 4.0)
('x21', '=', 3.0)
('x22', '=', 1.0)
('objective=', 16.5)

p(S2) = p(S1) = .5 ('Status:', 'Optimal')
('x1', '=', 2.0)
('x21', '=', 6.0)
('x22', '=', 2.0)
('objective=', 22.0)

We can see that the adverse effects of Scenario Two are reduced as the probability of that event decrease from .75 to .50. Intuitively, this makes sense since more competition results in a lower price and other needed changes that reduce profitability. Thus a reduction in the factor for the adverse event allow an increase in the objective using stochastic programming.

Problem 7.6-3

In [69]:
```
from pulp import *
prob = LpProblem("p7.6-3", LpMaximize)   #

# Variables, lower bounds added here
x1 = LpVariable("test_market_advert",lowBound=5,upBound=10)
x21 = LpVariable("national_market_advert_vf",lowBound=0)
x22 = LpVariable("national_market_advert_bf",lowBound=0)
x23 = LpVariable("national_market_advert_uf",lowBound=0)
z = LpVariable("z", 0)

# Objective
p_very_favorable = .25
p_barely_favorable = .25
p_unfavorable = .5

prob += .5*(x1) + p_very_favorable*( 2*x21 ) + p_barely_favorable*( 0.2*x22 ) + p_unfav


# Constraints
prob += x21 + x22 + x23 + x21 <= 100 , "plant constraint 1"

# Solve the optimization problem using the specified Solver
GLPK().solve(prob)

# Solution
print ("Status:", LpStatus[prob.status])
for v in prob.variables():
    print (v.name, "=", v.varValue)

print ("objective=", value(prob.objective) )
```

```
('Status:', 'Optimal')
('national_market_advert_bf', '=', 0.0)
('national_market_advert_uf', '=', 0.0)
('national_market_advert_vf', '=', 50.0)
('test_market_advert', '=', 10.0)
('objective=', 30.0)
```

10M should be spent in the test market and 50M should be spent nationally if the tests are very favorable. No money should be spent nationally if the tests are unfavorable or barely favorable.

The net cost is the objective profit of 30M less the fixed development costs of 40M or a loss of 10M. In a statistical sense, the company should not go ahead since the expected total net profit is negative 10M.