Shadow price sensitivity analysis

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Define shadow price

Modeling in issues:

- Input for model constraints are often estimates
- Will changes to input change our solution?

Shadow Prices:

- *The change in optimal value of the objective function per unit increase in the right-hand-side for a constraint, given everything else remain unchanged.*

Context - Glass Company - Resource Planning:

Resource	Prod. A	Prod. B	Prod. C
Production hours	6	5	8
WH Capacity sq. ft.	10.5	20	10
Profit $US	$500	$450	$600

Constraints:

- Production Capacity Hours ≤ 60
- Warehouse Capacity ≤ 150 sq. ft.
- Max Production of A ≤ 8

Code example

```python
# Initialize Class, Define Vars., and Objective
model = LpProblem("Max Glass Co. Profits",
                  LpMaximize)
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)
model += 500 * A + 450 * B + 600 * C

# Constraint 1
model += 6 * A + 5 * B + 8 * C <= 60

# Constraint 2
model += 10.5 * A + 20 * B + 10 * C <= 150
```

```python
# Constraint 3
model += A  <= 8

# Solve Model
model.solve()
print("Model Status:
       {}".format(pulp.LpStatus[model.status]))
print("Objective = ", value(model.objective))
for v in model.variables():
    print(v.name, "=", v.varValue)
```

Example solution

Solution:

Products	Prod. A	Prod. B	Prod. C
Production Cases	6.667	4	0

Objective value is $5133.33

Review constraints

Decision Variable:

- A through C = Number of cases of respective A through C products

Constraints:

- $6A + 5B + 8C \leq 60$ (*limited production capacity*)
- $10A + 20B + 10C \leq 150$ (*limited warehouse capacity*)
- $A \leq 8$ (*max production of A*)

Print shadow price

Python Code:

```
o = [{'name':name, 'shadow price':c.pi}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Shadow prices explained

Output:

```
  name  shadow price
   _C1     78.148148
   _C2      2.962963
   _C3     -0.000000
```

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Constraint slack

`slack`:

- The amount of a resource that is unused.

Python:

```
o = [{'name':name, 'shadow price':c.pi, 'slack': c.slack}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Constraint slack explained

Output:

```
  name  shadow price     slack
   _C1     78.148148 -0.000000
   _C2      2.962963 -0.000000
   _C3     -0.000000  1.333333
```

More About Binding

- `slack` = 0, then ***binding***
- Changing ***binding*** constraint, ***changes*** solution

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Summary

- How to compute:
  - `shadow prices`
  - constraint `slack`
- Identify Binding Constraints
  - `slack` = 0, then ***binding***
  - `slack` > 0, then ***not-binding***

Try it out!

SUPPLY CHAIN ANALYTICS IN PYTHON

Capacitated plant location - case study P3

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Capacitated plant location model

Modeling

- Production at regional facilities
    - Two plant sizes (low / high)
- Exporting production to other regions
- Production facilities open / close

Expected ranges

What should we expected for values of our decision variables?

Production Quantities:

- High production in regions with low variable production and shipping costs
- Maxed production in regions that also have relatively low fixed production costs

Production Plant Open Or Closed:

- High capacity production plant in regions with high demand
- High capacity production plant in regions with relatively low fixed costs

Sensitivity analysis of constraints

Total Production = Total Demand:

- `shadow prices` = Represent changes in total cost per increase in demand for a region
- `slack` = Should be zero

Total Production ≤ Total Production Capacity:

- `shadow prices` = Represent changes in total costs per increase in production capacity
- `slack` = Regions which have excess production capacity

```python
from pulp import *
import pandas as pd

# Initialize Class
model =
    LpProblem("Capacitated Plant Location Model",
               LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts(
                "production_",
                [(i,j) for i in loc for j in loc],
                 lowBound=0, upBound=None,
                 cat='Continuous')
```

```python
y = LpVariable.dicts(
               "plant_",
               [(i,s) for s in size for i in loc],
                cat='Binary')
# Define Objective Function
model +=
  (lpSum([fix_cost.loc[i,s]*y[(i,s)]
          for s in size for i in loc])
 + lpSum([var_cost.loc[i,j]*x[(i,j)]
          for i in loc for j in loc]))

# Define the Constraints
for j in loc: model +=
  lpSum([x[(i, j)]
        for i in loc]) == demand.loc[j,'Dmd']
for i in loc: model +=
  lpSum([x[(i, j)] for j in loc]) <= lpSum(
             [cap.loc[i,s]*y[(i,s)]for s in size])
```

```python
# Solve
model.solve()
# Print Decision Variables and Objective Value
print(LpStatus[model.status])
o = [{'prod':"{} to {}".format(i,j), 'quant':x[(i,j)].varValue}
     for i in loc for j in loc]
print(pd.DataFrame(o))
o = [{'loc':i, 'lc':y[(i,size[0])].varValue, 'hc':y[(i,size[1])].varValue}
     for i in loc]
print(pd.DataFrame(o))
print("Objective = ", value(model.objective))
# Print Shadow Price and Slack
o = [{'name':name, 'shadow price':c.pi, 'slack': c.slack}
     for name, c in model.constraints.items()]
print(pd.DataFrame(o))
```

Business questions

Likely Questions:

- What is the expected cost of this supply chain network model?
- If demand increases in a region how much profit is needed to cover the costs of production and shipping to that region?
- Which regions still have production capacity for future demand increase?

Summary

Reviewed:

- Expected ranges for decision variables
- Interpreted the output of sensitivity analysis (`shadow prices` and `slack`)
- Code to solve and output results
- Likely business related question

Great work! Your turn

SUPPLY CHAIN ANALYTICS IN PYTHON

Simulation testing solution

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Caution

- Problems that take a long time to solve should not be used with LP or IP

Overall concept

General Concept:

- Add random noise to key inputs you choose
- Solve the model repeatedly
- Observe the distribution

Why we might try

Why:

- Inputs are often estimates. There is a risk that they are inaccurate.
- Earlier Sensitivity Analysis only looked at changing one input at a time.

Context

Context - Glass Company - Resource Planning:

Resource	Prod. A	Prod. B	Prod. C
Profit $US	$500	$450	$600

Constraints:

- There are demand, production capacity, and warehouse Capacity constraints

Risks:

- Estimates of profits may be inaccurate

```python
# Initialize Class, & Define Variables
model = LpProblem("Max Glass Co. Profits", LpMaximize)
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)

# Define Objective Function
model += 500 * A + 450 * B + 600 * C

# Define Constraints & Solve
model += 6 * A + 5 * B + 8 * C <= 60
model += 10.5 * A + 20 * B + 10 * C <= 150
model += A  <= 8
model.solve()
```

Code example - step 2

```
a, b, c = normalvariate(0,25),
          normalvariate(0,25),
          normalvariate(0,25)
```

```
# Define Objective Function
model += (500+a)*A + (450+b)*B + (600+c)*C
```

```
# Initialize Class, & Define Variables
model = LpProblem("Max Glass Co. Profits",
                  LpMaximize)
```

```
A = LpVariable('A', lowBound=0)
B = LpVariable('B', lowBound=0)
C = LpVariable('C', lowBound=0)
a, b, c = normalvariate(0,25),
          normalvariate(0,25),
          normalvariate(0,25)
# Define Objective Function
model += (500+a)*A + (450+b)*B + (600+c)*C

# Define Constraints & Solve
model += 6 * A + 5 * B + 8 * C <= 60
model += 10.5 * A + 20 * B + 10 * C <= 150
model += A  <= 8
model.solve()
```

```python
def run_pulp_model():
    # Initialize Class
    model = LpProblem("Max Glass Co. Profits", LpMaximize)
    A = LpVariable('A', lowBound=0)
    B = LpVariable('B', lowBound=0)
    C = LpVariable('C', lowBound=0)
    a, b, c = normalvariate(0,25), normalvariate(0,25), normalvariate(0,25)

    # Define Objective Function
    model += (500+a)*A + (450+b)*B + (600+c)*C

    # Define Constraints & Solve
    model += 6 * A + 5 * B + 8 * C <= 60
    model += 10.5 * A + 20 * B + 10 * C <= 150
    model += A  <= 8
    model.solve()
    o = {'A':A.varValue, 'B':B.varValue, 'C':C.varValue, 'Obj':value(model.objective)}
    return(o)
```

Code example - step 4

```python
def run_pulp_model():
    # Initialize Class
    model = LpProblem("Max Glass Co. Profits",
                      LpMaximize)
    A = LpVariable('A', lowBound=0)
    B = LpVariable('B', lowBound=0)
    C = LpVariable('C', lowBound=0)
    a, b, c = normalvariate(0,25),
              normalvariate(0,25),
              normalvariate(0,25)

    # Define Objective Function
    model += (500+a)*A + (450+b)*B +
             (600+c)*C
```

```python
    # Define Constraints & Solve
    model += 6 * A + 5 * B + 8 * C <= 60
    model += 10.5 * A + 20 * B + 10 * C <= 150
    model += A  <= 8
    model.solve()
    o = {'A':A.varValue, 'B':B.varValue,
         'C':C.varValue,
         'Obj':value(model.objective)}
    return(o)
```

```python
for i in range(100):
    output.append(run_pulp_model())
df = pd.DataFrame(output)
```

Code example - step 5

```
print(df['A'].value_counts())
print(df['B'].value_counts())
print(df['C'].value_counts())
```

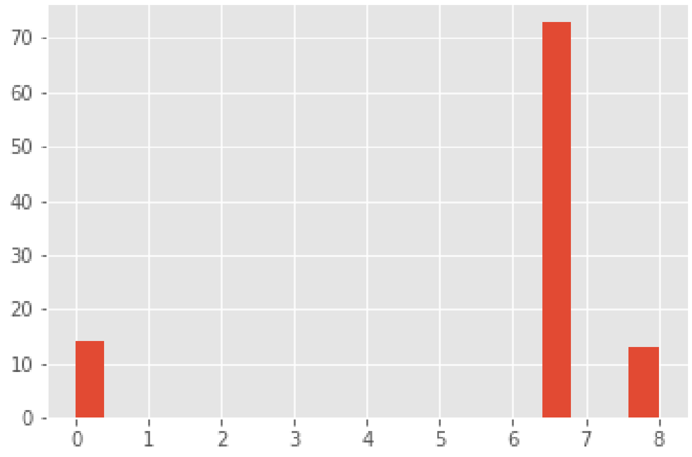
Output: (results may be different)

```
6.666667    73
0.000000    14
8.000000    13
Name: A, dtype: int64
```

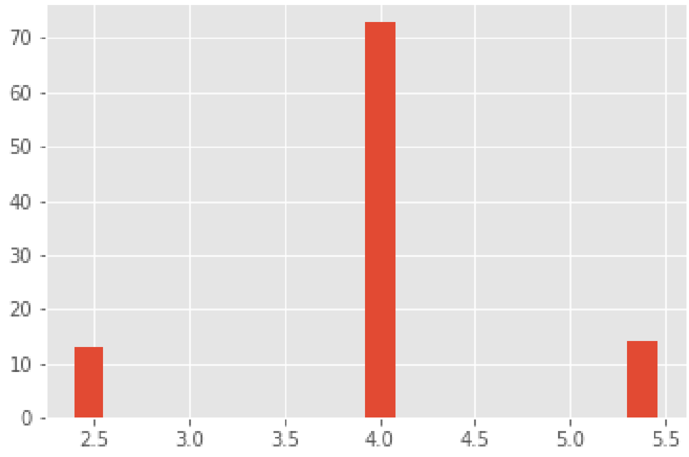
```
4.000000    73
5.454546    14
2.400000    13
Name: B, dtype: int64
0.000000    86
4.090909    14
```

Visualize as histogram

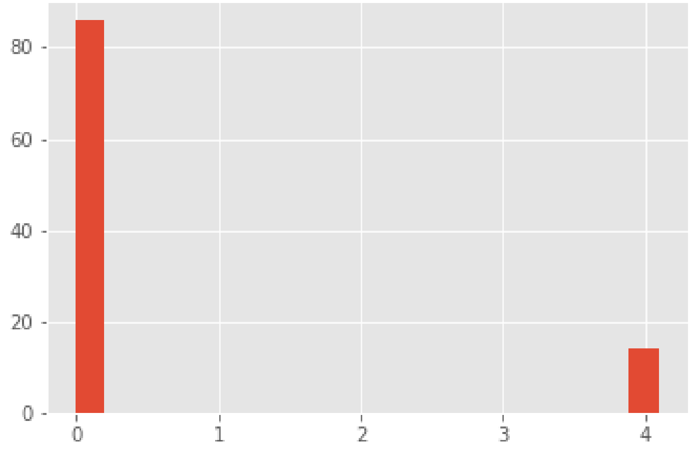
Product A:



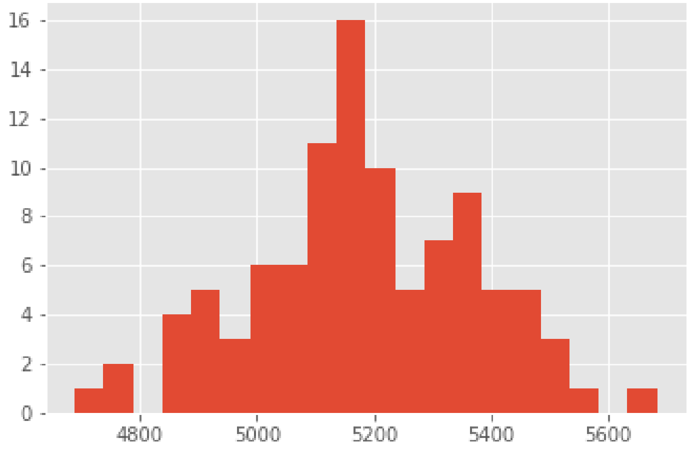
Product B:



Product C:



Objective Values:



Summary

- Should not be used on problems that take a long time to solve
- Benefits
  - View how optimal results change as model inputs change
- Steps
  1. Start with standard PuLP model code
  2. Add noise to key inputs using Python's `normalvariate`
  3. Wrap PuLP model code in a function that returns the model's output
  4. Create loop to call newly created function and store results in DataFrame
  5. Visualize results DataFrame

Try it out!

SUPPLY CHAIN ANALYTICS IN PYTHON

Capacitated plant location - case study P4

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr., Ingredion

Simulation vs. sensitivity analysis

With Sensitivity Analysis:

- Observe how changes in demand and costs affect production:
  - Where should production be added?
  - Does production move to a different region.
  - Which regions have stable production quantities?
- Observe multiple changes at once vs. one at a time with sensitivity analysis

Simulation modeling

We can apply simulation testing to our Capacitated Plant Location Model

Possible inputs for adding noise

- **Demand**
- **Variable costs**
- Fixed costs
- Capacity

```python
# Initialize Class
model = LpProblem(
            "Capacitated Plant Location Model",
             LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts(
       "production_",
       [(i,j) for i in loc for j in loc],
       lowBound=0, upBound=None, cat='Continuous')
y = LpVariable.dicts(
      "plant_", [(i,s)for s in size for i in loc],
       cat='Binary')
```

```python
# Define Objective Function
model +=(lpSum([fix_cost.loc[i,s]*y[(i,s)]
               for s in size for i in loc])
       + lpSum([var_cost.loc[i,j]*x[(i,j)]
                for i in loc for j in loc]))

# Define the Constraints
for j in loc: model +=
  lpSum([x[(i, j)] for i in loc]) == demand.loc[
                                        j,'Dmd']
for i in loc: model +=
  lpSum([x[(i, j)] for j in loc]) <= lpSum(
                            [cap.loc[i,s]*y[(i,s)]
                             for s in size])
# Solve
model.solve()
print(LpStatus[model.status])
```

Objective:

```
model %20= (lpSum([fix_cost.loc[i,s]*y[(i,s)] for s in size for i in loc])
          %20 lpSum([(var_cost.loc[i,j] %20 normalvariate(0.5, 0.5))*x[(i,j)]
                   for i in loc for j in loc]))
```

Total Demand:

```
for j in loc:
    rd = normalvariate(0, demand.loc[j,'Dmd']*.05)
    model %20= lpSum([x[(i,j)] for i in loc]) == (demand.loc[j,'Dmd']%20rd)
```

Code example - step 3

```
def run_pulp_model(fix_cost, var_cost, demand,
                   cap):
    # Initialize Class
    model = LpProblem(
              "Capacitated Plant Location Model",
               LpMinimize)

    # Define Decision Variables
    loc = ['A', 'B', 'C', 'D', 'E']
    size = ['Low_Cap','High_Cap']
    x = LpVariable.dicts(
                "production_",
                [(i,j) for i in loc for j in loc],
                lowBound=0, upBound=None,
                cat='Continuous')
```

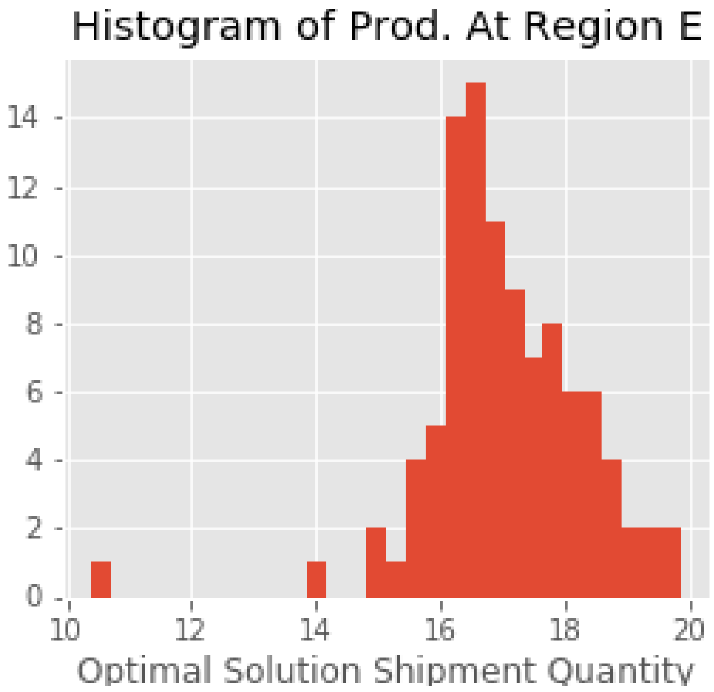
```
    y = LpVariable.dicts(
                  "plant_",
                  [(i,s) for s in size for i in loc],
                   cat='Binary')
    # Define the Constraints
    for j in loc: rd = normalvariate(
                         0, demand.loc[j,'Dmd']*.05)
        model += lpSum(
         [x[(i,j)] for i in loc]) == (
                              demand.loc[j,'Dmd']+rd)
    for i in loc: model +=
      lpSum([x[(i,j)] for j in loc]) \
        <= lpSum([cap.loc[i,s]*y[(i,s)]
            for s in size])
```

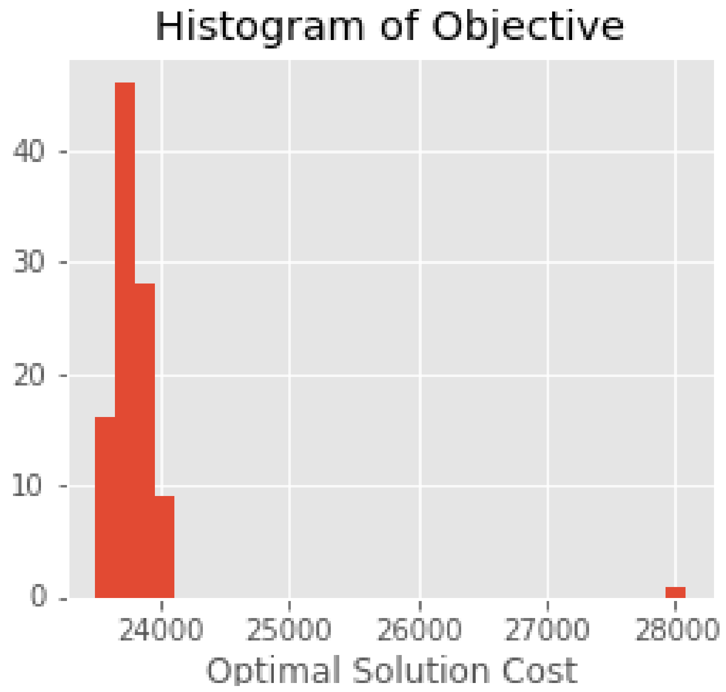
```python
    # Solve
    model.solve()
    o = {}
    for i in loc:
        o[i] = value(lpSum([x[(i, j)] for j in loc]))
    o['Obj'] = value(model.objective)
    return(o)
```

```python
for i in range(100):
    output.append(run_pulp_model(fix_cost, var_cost, demand, cap))
df = pd.DataFrame(output)
```

Results

```
import matplotlib.pyplot as plt
plt.title('Histogram of Prod. At Region E')
plt.hist(df['E'])
plt.show()
```





Summary

Capacitated Plant Model

- Simulation vs. sensitivity analysis
- Stepped through code example

Try it out!

Final summary

SUPPLY CHAIN ANALYTICS IN PYTHON

Aaren Stubberfield
Supply Chain Analytics Mgr.

Summary

- Reviewed what is Linear Programing (LP)
- Reviewed PuLP and how it can be used with LP
- Solving large scale models
  - `LpSum()`
  - `LpVariable.dicts()`
- Logical constraints
- Common constraint mistakes
- Solving PuLP model
  - printing decision variables, and objective

Summary

- Sanity checking solution
- Sensitivity Analysis
  - Shadow Prices
  - Slack
- Simulation Testing
- Capacitated Plant Location model - Case Study

Congratulations!

Additional resources

For more on PuLP check out these additional resources:

- https://www.coin-or.org/PuLP/
- https://www.coin-or.org/
- PuLP GitHub: https://github.com/coin-or/pulp
- Google group: https://groups.google.com/forum/#!forum/pulp-or-discuss

Additional resources

For books related to the subject, check out these:

- Bradley, Stephen P., et al. *Applied Mathematical Programming*. Addison-Wesley, 1977.
- Chopra, Sunil, and Meindl, Peter. *Supply Chain Management: Strategy, Planning, and Operations.* Pearson Prentice-Hall, 2007.

Thank you!

SUPPLY CHAIN ANALYTICS IN PYTHON