Linear Programming

Example #2: Dahlby Outfitters

Dahlby Outfitters wants to introduce a new trail mix. Each ingredient contain amounts of vitamins, minerals, proteins, and calories (must contain at least the amount shown in table). Product needs to have certain minimal nutritional profile. Want to find the optimal mix that minimizes cost.

Information

	Grams per Pound					
Component	Seeds	Raisins	Flakes	Pecans	Walnuts	Nutritional Requirements
Vitamins	10	20	10	30	20	20
Minerals	5	7	4	9	2	10
Protein	1	4	10	2	1	15
Calories	500	450	160	300	500	600
Cost/pound	$4	$5	$3	$7	$6	

```
# Create a new model
m = Model("Example 2")
# Create variables
s = m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="Seeds")
r = m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="Raisins")
f = m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="Flakes")
p = m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="Pecans")
w = m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="Walnuts")
# Set objective
m.setObjective(4*s + 5*r + 3*f + 7*p + 6*w, GRB.MINIMIZE)
# Add constraints
m.addConstr(10*s + 20*r + 10*f + 30*p + 20*w >= 20, "Vitamins")
m.addConstr(5*s + 7*r + 4*f + 9*p + 2*w >= 10, "Minerals")
m.addConstr(1*s + 4*r + 10*f + 2*p + 1*w >= 15, "Protein")
m.addConstr(500*s + 450*r + 160*f + 300*p + 500*w >= 600, "Calories")
# Optimize
m.optimize()
# Print results for each constraint
for v in m.getVars():
    print('%s: %g' % (v.varName, v.x))
# Print result for Objective using optimized constraints
print('Obj: %g' % m.objVal)
```

Output

Solved in 3 iterations and 0.04 seconds
Optimal objective 7.535799523e+00
Seeds: 0.477327
Raisins: 0.334129
Flakes: 1.31862
Pecans: 0
Walnuts: 0
Obj: 7.5358

Binding Constraints

A constraint is **binding** if the LHS of a constraint equals the RHS of the constraint (LHS is the value of the constraint when the solution values are used…RHS is the original value of the parameter set by the problem)

For example, in previous example, take the Mineral constraint:

Mineral: **5***Seeds + **7***Raisins + **4***Flakes + **9***Pecans + **2***Walnuts $\geq$ 10

5*(0.4773270) + **7***(0.3341289) + **4***(1.3186158) + **9***(0) + **2***(0) = 10

Seeds	Raisins	Flakes	Pecans	Walnuts
0.4773270	0.3341289	1.3186158	0.0000000	0.0000000

Binding Constraints

In Python, this information is called “slack” (think….how much slack do I have from RHS to LHS)

```
for c in m.getConstrs():
    print('%s: %g' % (c.ConstrName, c.slack))
```

From this example
BINDING CONSTRAINTS!

Vitamins: -4.642
Minerals: 0
Protein: 0
Calories: 0

Sensitivity

How sensitive is the solution to changes in the parameter values (i.e. values in the constraints)? ***SHADOW PRICE*** (shadow price and binding constraints go hand-in-hand)

How sensitive is the solution to changes in the coefficients in the objective function? ***REDUCED COST***

We can perform a sensitivity analysis to investigate these ideas further.

Note: Shadow price pertains to the constraints

Reduced cost pertains to the objective function

Shadow price

Illustration of shadow price

Let's go back to the furniture example and see if there are any binding constraints:
Remember:

Objective function:

- MAXIMIZE: 15C + 24D + 18T

Constraints:

- Fabri: 4C + 6D + 2T $\leq$ 1850
- Assem: 3C + 5D + 7T $\leq$ 2400
- Shipp: 3C + 2D + 4T $\leq$ 1500
- C $\leq$ 360
- D $\leq$ 300
- T $\leq$ 100

"Slack"

Fabrication: 0

Assembly: 325

Shipping: 550

Chair Demand: 360

Desk Demand: 25

Table Demand: 0

So the first constraint (fabrication) is a binding constraint (and so is Table)
See what happens when we increase fabrication by 1 hour, keeping everything else the same

Illustration of shadow price

Original fabrication is 1850…..objective value is 8400

Illustration of shadow price

Original fabrication is 1850…..objective value is 8400

Increase fabrication to 1851…..objective value is now 8404

Illustration of shadow price

Original fabrication is 1850…..objective value is 8400

Increase fabrication to 1851…..objective value is now 8404

Increase fabrication to 1852…..objective value is now 8408

So we see that increasing fabrication by 1 hour will increase profit by $4!

Now let's see how the objective function changes if we change a constraint that is NOT binding!!

Illustration of shadow price

Original assembly is 2400…..objective value is 8400

Change assembly to 2401 (keeping everything else as originally stated)….objective value is 8400

Illustration of shadow price

Original assembly is 2400…..objective value is 8400

Change assembly to 2401 (keeping everything else as originally stated)….objective value is 8400

Now change assembly to 2402 ….objective value is 8400

Shadow/Dual Prices

The **shadow price** of a constraint indicates the amount by which the objective function value changes given a unit increase (or decrease…depending on the direction of the constraint) in the RHS (or parameter) of the constraint, assuming all other parameters remain constant.

“If we had one additional (or one less)unit of a constraint, then the objective function would change by the shadow price.” Also referred to as the dual price.

Shadow prices for nonbinding constraints are always zero.

Getting shadow prices/dual prices

In Gurobi:

```
shadow_prices = m.getAttr('Pi')
m.printAttr('Pi')
```

```
 Constraint          Pi
----------------------
 Fabrication          4
Table Demand         10
```

Slack versus Pi in Python

IF a constraint is binding (RHS = LHS), then the slack for that constraint is 0 (think about it as there is NO slack!).

IF a constraint is binding (RHS = LHS), then Pi will give you the shadow price (dual price) for that constraint….i.e. as we change the “parameter” by one, how does the objective value change.

Reduced cost

Reduced Costs

The ***reduced cost*** is the amount by which an objective function coefficient would have to improve before it would be possible for a corresponding variable to assume a positive value in the optimal solution.

Will be nonzero when the optimal amount of a decision variable is 0. For example, in the Veerman Furniture Company example, the optimal solution contains NO chairs. The reduced cost would indicate what the profit would need to be in order for the optimal solution to contain some value for chairs.

Reduced Cost

In Gurobi:

```
for v in m.getVars():
    print('%s: %g' % (v.varName, v.rc))
```

So need to increase profit on chairs from $15 to $16.

Chair: -1
Desk: 0
Table: 0

Another Example

Blending Models

Blending models are usually stated as proportions (for example, we want chairs to constitute at least 25% of the overall product mix).

We can rewrite this percent in terms of a linear constraint:

$$\frac{C}{C + D + T} \geq 0.25$$

Which, mathematically is…

$0.75C - 0.25D - 0.25T \geq 0$

Example

Diaz Coffee Company blends three types of coffee beans (Brazilian, Colombian, Peruvian). These different blends have different aroma and strength ratings (see table below). They would like an average aroma rating of at least 78 and an average strength rating of at least 16.

Goal: Make 4,000,000 pounds with lowest cost.

Bean	Aroma Rating	Strength Rating	Cost/lb	Pounds available
Brazilian	75	15	$0.5	1,500,000
Colombian	60	20	$0.6	1,200,000
Peruvian	85	18	$0.7	2,000,000

```
## COFFEE BLENDING EXAMPLE ##
m = Model("Blending Model")
# Create variables
b = m.addVar(vtype=GRB.CONTINUOUS, lb=0,
name="Brazilian")
c = m.addVar(vtype=GRB.CONTINUOUS, lb=0,
name="Colombian")
p = m.addVar(vtype=GRB.CONTINUOUS, lb=0,
name="Peruvian")
# Set objective
m.setObjective(0.5*b + 0.6*c + 0.7*p, GRB.MINIMIZE)
# Add constraints
m.addConstr(-3*b - 18*c + 7*p >= 0, name='Aroma')
m.addConstr(-1*b + 4*c + 2*p >= 0, name='Strength')
m.addConstr(b <= 1500000, name='Brazilian Avail')
m.addConstr(c <= 1200000, name='Colombian Avail')
m.addConstr(p <= 2000000, name='Peruvian Avail')
m.addConstr(b + c + p == 4000000, name='Blend Total')
# Optimize
m.optimize()
```

```
# Print results for each constraint
for v in m.getVars():
    print('%s: %g' % (v.varName, v.x))
# Print result for
print('Obj: %g' % m.objVal)
# Get shadow prices
m.printAttr('Pi')
```

Output

Optimal objective 2.448000000e+06
Brazilian: 1.5e+06
Colombian: 520000
Peruvian: 1.98e+06
Obj: 2.448e+06

Constraint	Pi
Aroma	0.004
Brazilian Avail	-0.16
Blend Total	0.672