



Previous: (4a) Installing PuLP at Home



Next: (5b) A Transportation Problem

(5a) A Blending Problem

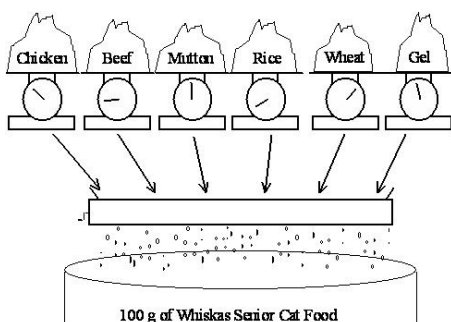
The Whiskas Problem

Problem Description



Whiskas cat food, shown above, is manufactured by Uncle Ben's. Uncle Ben's want to produce their cat food products as cheaply as possible while ensuring they meet the stated nutritional analysis requirements shown on the cans. Thus they want to vary the quantities of each ingredient used (the main ingredients being chicken, beef, mutton, rice, wheat and gel) while still meeting their nutritional standards.

INGREDIENTS: Selected meat derived from chicken, beef and mutton; rice; wheat bran; gel; all essential vitamins and minerals, including thiamine and taurine. No preservatives.	NUTRITIONAL ANALYSIS:	
	Minimum% Crude Protein	8.0
	Minimum% Crude Fat	6.0
	Maximum% Crude Fibre	2.0
	Max % Salt (Naturally Occurring)	0.4



The costs of the chicken, beef, and mutton are \$0.013, \$0.008 and \$0.010 respectively, while the costs of the rice, wheat and gel are \$0.002, \$0.005 and \$0.001 respectively. (All costs are per gram.) For this exercise we will ignore the vitamin and mineral ingredients. (Any costs for these are likely to be very small anyway.)

Each ingredient contributes to the total weight of protein, fat, fibre and salt in the final product. The contributions (in grams) per gram of ingredient are given in the table below.

	Protein	Fat	Fibre	Salt
Chicken	0.100	0.080	0.001	0.002
Beef	0.200	0.100	0.005	0.005
Mutton	0.150	0.110	0.003	0.007
Rice	0.000	0.010	0.100	0.002
Wheat bran	0.040	0.010	0.150	0.008
Gel	-	-	-	-

Simplified Formulation

First we will consider a simplified problem to build a simple Python model.

1. Identify the Decision Variables

Assume Whiskas want to make their cat food out of just two ingredients: Chicken and Beef. We will first define our decision variables:

x_1 = percentage of chicken meat in a can of cat food
 x_2 = percentage of beef used in a can of cat food

The limitations on these variables (greater than zero) must be noted but for the Python implementation, they are not entered or listed separately or with the other constraints.

2. Formulate the Objective Function

The objective function becomes:

$$\min \$0.013x_1 + \$0.008x_2$$

3. Formulate the Constraints

The constraints on the variables are that they must sum to 100 and that the nutritional requirements are met:

$$\begin{array}{rcl} 1.000x_1 & + & 1.000x_2 = 100.0 \\ 0.100x_1 & + & 0.200x_2 \geq 8.0 \\ 0.080x_1 & + & 0.100x_2 \geq 6.0 \\ 0.001x_1 & + & 0.005x_2 \geq 2.0 \\ 0.002x_1 & + & 0.005x_2 \geq 0.4 \end{array}$$

Solution to Simplified Problem

To obtain the solution to this Linear Program, we can write a short program in Python to call PuLP's modelling functions, which will then call a solver. This will explain step-by-step how to write this Python program. It is suggested that you repeat the exercise yourself.

The start of the your file should then be headed with a short commenting section outlining the purpose of the program. For example:

```
"""
The Simplified Whiskas Model Python Formulation for the PuLP Modeller

Authors: Antony Phillips, Dr Stuart Mitchell    2007
"""
```

Then you will import PuLP's functions for use in your code:

```
# Import PuLP modeler functions
from pulp import *
```

A variable called "prob" (although its name is not important) is created using the `LpProblem` function. It has two parameters, the first being the arbitrary name of this problem (as a string), and the second parameter being either `LpMinimize` or `LpMaximize` depending on the type of LP you are trying to solve:

```
# Create the prob variable to contain the problem data
prob = LpProblem("The Whiskas Problem", LpMinimize)
```

The problem variables "x1" and "x2" are created using the `LpVariable` function. It has four parameters, the first is the arbitrary name of what this variable represents, the second is the lower bound on this variable, the third is the upper bound, and the fourth is essentially the type of data (discrete or continuous). The options for the fourth parameter are `LpContinuous` or `LpInteger`, with the default as `LpContinuous`. If we were modelling the number of cans to produce, we would need to input `LpInteger` since it is discrete data. The bounds can be entered directly as a number, or `None` to represent no bound (i.e. positive or negative infinity), with `None` as the default. If the first few parameters are entered and the rest are ignored (as shown), they take their default values. However, if you wish to specify the third parameter, but you want the second to be the default value, you will need to specifically set the second parameter as it's default value. i.e you cannot leave a parameter entry blank. (e.g `LpVariable("example", None, 100)`)

```
# The 2 variables Beef and Chicken are created with a lower limit of zero
x1 = LpVariable("ChickenPercent", 0)
x2 = LpVariable("BeefPercent", 0)
```

The variable "prob" now begins collecting problem data with the "+=" operator. The objective function is logically entered first, with an important comma , at the end of the statement and a short string explaining what this objective function is:

```
# The objective function is added to prob first
prob += 0.013*x1 + 0.008*x2, "Total Cost of Ingredients per can"
```

The constraints are now entered (Note: any "non-negative" constraints were already included when defining the variables). This is done with the "+=" operator again, since we are adding more data to the "prob" variable. The constraint is logically entered after this, with a comma at the end of the constraint equation and a brief description of the cause of that constraint:

```
# The five constraints are entered
prob += x1 + x2 == 100, "PercentagesSum"
prob += 0.100*x1 + 0.200*x2 >= 8.0, "ProteinRequirement"
prob += 0.080*x1 + 0.100*x2 >= 6.0, "FatRequirement"
prob += 0.001*x1 + 0.005*x2 <= 2.0, "FibreRequirement"
prob += 0.002*x1 + 0.005*x2 <= 0.4, "SaltRequirement"
```

Now that all the problem data is entered, the `writeLP` function can be used to copy this information into a `.lp` file into the directory that your code is running from. Once your code runs successfully, you can open this `.lp` file with Notepad to see what the above steps were doing. You will notice that there is no assignment operator (such as an equals sign) on this line. This is because the function/method called `writeLP` is being performed to the variable/object `"prob"` (and the string `"WhiskasModel.lp"` is an additional parameter). The dot `"."` between the variable/object and the function/method is important and is seen frequently in Object Oriented software (such as this):

```
# The problem data is written to a .lp file
prob.writeLP("WhiskasModel.lp")
```

The LP is solved using the solver that PuLP chooses. The input brackets after `solve` are left empty in this case, however they can be used to specify which solver to use (e.g `prob.solve(CPLEX())`):

```
# The problem is solved using PuLP's choice of solver
prob.solve()
```

Now the results of the solver call can be displayed as output to us. Firstly, we request the status of the solution, which can be one of "Not Solved", "Infeasible", "Unbounded", "Undefined" or "Optimal". The value of `prob.status` is returned as an integer, which must be converted to it's significant text meaning using the `LpStatus` dictionary. Since `LpStatus` is a dictionary, it's input must be in square brackets:

```
# The status of the solution is printed to the screen
print "Status:", LpStatus[prob.status]
```

The variables and their resolved optimum values can now be printed to the screen. The `for` loop makes `"v"` cycle through all the problem variable names (in this case just `"ChickenPercent"` and `"BeefPercent"`). Then it prints each variable name, followed by an equals sign, followed by its optimum value. `.name` and `.varValue` are properties of the object `v`.

```
# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print v.name, "=", v.varValue
```

The optimised objective function value is printed to the screen, using the value function. This ensures that the number is in the right format to be displayed. `.objective` is an attribute of the object `'prob'`:

```
# The optimised objective function value is printed to the screen
print "Total Cost of Ingredients per can = ", value(prob.objective)
```

Running this file should then produce the output to show that Chicken will make up 33.33%, Beef will make up 66.67% and the Total cost of ingredients per can is 96 cents.

The segmented .py file shown above can be obtained in full [here](#) . Opening the file normally will run it, which will execute the commands and exit in less than 1 second. You will need to open this file with IDLE or PyDev to run it and view the output properly.

Full Formulation

Now we will formulate the problem fully with all the variables. Whilst it could be implemented into Python with little addition to our method above, we will look at a better way which does not mix the problem data, and the formulation as much. This will make it easier to change any problem data for other tests. We will start the same way by algebraically defining the problem:

1. Identify the Decision Variables

For the Whiskas Cat Food Problem the decision variables are the percentages of the different ingredients we include in the can. Since the can is 100g, these percentages also represent the amount in g of each ingredient included. In STATS 255, the decisions were the amount in g in each cat food, but it is more convenient to use percentages.

We must formally define our decision variables, being sure to state the units we are using.

```
x1 = percentage of chicken meat in a can of cat food
x2 = percentage of beef used in a can of cat food
x3 = percentage of mutton used in a can of cat food
x4 = percentage of rice used in a can of cat food
x5 = percentage of wheat bran used in a can of cat food
x6 = percentage of gel used in a can of cat food
```

Note that these percentages must be between 0 and 100.

2. Formulate the Objective Function

For the Whiskas Cat Food Problem the objective is to minimise the total cost of ingredients per can of cat food.

We know the cost per g of each ingredient. We decide the percentage of each ingredient in the can, so we must divide by 100 and multiply by the weight of the can in g. This will give us the weight in g of each ingredient.

$$\min \$0.013x_1 + \$0.008x_2 + \$0.010x_3 + \$0.002x_4 + \$0.005x_5 + \$0.001x_6$$

3. Formulate the Constraints

The constraints for the Whiskas Cat Food Problem are that:

1. The sum of the percentages must make up the whole can (= 100%).
2. The stated nutritional analysis requirements are met.

The constraint for the "whole can" is:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 100$$

To meet the nutritional analysis requirements, we need to have at least 8g of Protein per 100g, 6g of fat, but no more than 2g of fibre and 0.4g of salt. To formulate these constraints we make use of the previous table of contributions from each ingredient. This allows us to formulate the following constraints on the total contributions of protein, fat, fibre and salt from the ingredients:

$$\begin{array}{rrrrrrrr}
 2.100x_1 & +0.290x_2 & +0.150x_3 & +0.090x_4 & +0.040x_5 & +0.0x_6 & \leq & 8.0 \\
 2.080x_1 & +0.100x_2 & +0.110x_3 & +0.010x_4 & +0.010x_5 & +0.0x_6 & \leq & 6.0 \\
 2.001x_1 & +0.005x_2 & +0.008x_3 & +0.100x_4 & +0.150x_5 & +0.0x_6 & \leq & 2.0 \\
 2.002x_1 & +0.005x_2 & +0.007x_3 & +0.002x_4 & +0.008x_5 & +0.0x_6 & \leq & 0.4
 \end{array}$$

4. Identify the Data

We know the total weight of the can. We also know the cost of the ingredients, the nutritional analysis requirements and the contribution (per gram) of each ingredient in terms of the nutritional analysis. This is enough to formulate the necessary mathematical programme, but we will reconsider our data after solving the mathematical programme and performing some post-optimal analysis.

Solution to Full Problem

To obtain the solution to this Linear Program, we again write a short program in Python to call PuLP's modelling functions, which will then call a solver. This will explain step-by-step how to write this Python program with it's improvement to the above model. It is suggested that you repeat the exercise yourself.

As with last time, it is advisable to head your file with commenting on it's purpose, and the author name and date. Importing of the PuLP functions is also done in the same way:

```

"""
The Full Whiskas Model Python Formulation for the PuLP Modeller

Authors: Antony Phillips, Dr Stuart Mitchell    2007
"""

# Import PuLP modeler functions
from pulp import *
```

Next, before the "prob" variable or type of problem are defined, the key problem data is entered into dictionaries. This includes the list of Ingredients, followed by the cost of each Ingredient, and it's percentage of each of the four nutrients. These values are clearly laid out and could easily be changed by someone with little knowledge of programming. The ingredients are the reference keys, with the numbers as the data.

```
# Creates a list of the Ingredients
Ingredients = ["CHICKEN", "BEEF", "MUTTON", "RICE", "WHEAT", "GEL"]

# A Dictionary of the costs of each of the Ingredients is created
costs = {"CHICKEN": 0.013,
         "BEEF" : 0.008,
         "MUTTON" : 0.010,
         "RICE" : 0.002,
         "WHEAT" : 0.005,
         "GEL" : 0.001}

# A dictionary of the protein percent in each of the Ingredients is created
proteinPercent = {"CHICKEN": 0.100,
                  "BEEF" : 0.200,
                  "MUTTON" : 0.150,
                  "RICE" : 0.000,
                  "WHEAT" : 0.040,
                  "GEL" : 0.000}

# A dictionary of the fat percent in each of the Ingredients is created
fatPercent = {"CHICKEN": 0.080,
              "BEEF" : 0.100,
              "MUTTON" : 0.110,
              "RICE" : 0.010,
              "WHEAT" : 0.010,
              "GEL" : 0.000}

# A dictionary of the fibre percent in each of the Ingredients is created
fibrePercent = {"CHICKEN": 0.001,
                "BEEF" : 0.005,
                "MUTTON" : 0.003,
                "RICE" : 0.100,
                "WHEAT" : 0.150,
                "GEL" : 0.000}

# A dictionary of the fibre percent in each of the Ingredients is created
saltPercent = {"CHICKEN": 0.002,
               "BEEF" : 0.005,
               "MUTTON" : 0.007,
               "RICE" : 0.002,
               "WHEAT" : 0.008,
               "GEL" : 0.000}
```

The "prob" variable is created to contain the formulation, and the usual parameters are passed into LpProblem.

```
# Create the prob variable to contain the problem data
prob = LpProblem("The Whiskas Problem", LpMinimize)
```

A dictionary called vars is created which contains the LP variables, with their defined lower bound of zero. The reference keys to the dictionary are the Ingredient names, and the data is Ingr_IngredientName. (e.g. MUTTON: Ingr_MUTTON)

```
# A dictionary called Vars is created to contain the referenced variables
vars = LpVariable.dicts("Ingr", Ingredients, 0)
```

Since costs and vars are now dictionaries with the reference keys as the Ingredient names, the data can be simply extracted with a list comprehension as shown. The lpSum function will add the elements of the resulting list. Thus the objective function is simply entered and assigned a name:

```
# The objective function is added to prob first
prob += lpSum([costs[i] * vars[i] for i in Ingredients]), "Total Cost of Ingredients per can"
```

Further list comprehensions are used to define the other 5 constraints, which are also each given names describing them.

```
# The five constraints are added to prob
prob += lpSum([vars[i] for i in Ingredients]) == 100, "PercentagesSum"
prob += lpSum([proteinPercent[i] * vars[i] for i in Ingredients]) >= 8.0 "ProteinRequirement"
prob += lpSum([fatPercent[i] * vars[i] for i in Ingredients]) >= 6.0 "FatRequirement"
prob += lpSum([fibrePercent[i] * vars[i] for i in Ingredients]) <= 2.0, "FibreRequirement"
prob += lpSum([saltPercent[i] * vars[i] for i in Ingredients] <= 0.4, "Salt Requirement"
```

Following this, the `prob.writeLP` line etc follow exactly the same as in the simplified example. To see the entire file, it is available [here](#).

The optimal solution is 60% Beef and 40% Gel leading to a Objective Function value of 52 cents per can.

Post-Optimal Analysis

For The Whiskas Cat Food Problem we will not perform any post-optimal analysis. However, as you will remember from STATS255, a Sensitivity Analysis and a Parametric Analysis can be performed.

Validation

To validate our solution for The Whiskas Cat Food Problem, we can do a quick check that our solution makes sense. First, the percentages should add up to 100%. If not, there is something wrong. Next, we can do a quick check of the constraints to ensure none of them are violated.

Note For large models you won't always be able to check the solution by hand.

The final validation is to write up a *management summary* for your manager and/or client and see if they think our solution is a valid one. If they identify some (or all) of the solution that is not valid, then you should discuss with them the reasons why it is invalid and start a "feedback" loop in your optimisation process.

Presentation of Solution and Analysis

The solution for The Whiskas Cat Food Problem is a simple one to summarise. Here is an example management summary (with the numbers removed):

The Whiskas Cat Food Problem

Stuart Mitchell, 2007

Uncle Ben's want to produce their cat food products as cheaply as possible while ensuring they meet the stated nutritional analysis requirements shown on the cans. Thus they want to vary the quantities of each ingredient used (the main ingredients being chicken, beef, mutton, rice, wheat and gel) while still meeting their nutritional standards.

The stated nutritional analysis requirements are: _____

The cost for each ingredient is: _____

To minimise the cost per 100g can of Whiskas cat food, Whiskas should use
_____ g of Chicken,
_____ g of Beef,

The cost per can is _____

IMPORTANT When presenting your solution you must be careful about the number of decimal places you use. You should not use a greater accuracy than your data allows.

Implementation and Ongoing Monitoring

The first step towards Implementation is a good management summary. You may also want to discuss any issues that may arise from the solution you have found with Uncle Ben's (for example, if your solution can be implemented on their production line).

Ongoing Monitoring may take the form of:

1. Updating your data files and resolving as the data changes (changing costs, nutritional requirements, etc);
2. Resolving our model for new products (e.g., 200g cans);
3. Looking for possible savings (e.g., analysing the cost of the ingredients to see if any price changes will affect the ingredient mix).



Previous: (4a) Installing PuLP at Home



Next: (5b) A Transportation Problem