Homework 11

2025-11-06

Question 15.2

In the videos, we saw the "diet problem". (The diet problem is one of the first large-scale optimization problems to be studied in practice. Back in the 1930's and 40's, the Army wanted to meet the nutritional requirements of its soldiers while minimizing the cost.) In this homework you get to solve a diet problem with real data. The data is given in the file diet.xls

1) Formulate an optimization model (a linear program) to find the cheapest diet that satisfies the maximum and minimum daily nutrition constraints, and solve it using PuLP. Turn in your code and the solution. (The optimal solution should be a diet of air-popped popcorn, poached eggs, oranges, raw iceberg lettuce, raw celery, and frozen broccoli. UGH!)
2) Please add to your model the following constraints (which might require adding more variables) and solve the new model:
    a) If a food is selected, then a minimum of 1/10 serving must be chosen. (Hint: now you will need two variables for each food i: whether it is chosen, and how much is part of the diet. You'll also need to write a constraint to link them.)
    b) Many people dislike celery and frozen broccoli. So at most one, but not both, can be selected.
    c) To get day-to-day variety in protein, at least 3 kinds of meat/poultry/fish/eggs must be selected. [If something is ambiguous (e.g., should bean-and-bacon soup be considered meat?), just call it whatever you think is appropriate – I want you to learn how to write this type of constraint, but I don't really care whether we agree on how to classify foods!]

If you want to see what a more full-sized problem would look like, try solving your models for the file diet_large.xls, which is a low-cholesterol diet model (rather than minimizing cost, the goal is to minimize cholesterol intake). I don't know anyone who'd want to eat this diet – the optimal solution includes dried chrysanthemum garland, raw beluga whale flipper, freeze-dried parsley, etc. – which shows why it's necessary to add additional constraints beyond the basic ones we saw in the video! [Note: there are many optimal solutions, all with zero cholesterol, so you might get a different one. It probably won't be much more appetizing than mine.]

Boilerplate

Let's import our packages and read in our data. Note: I moved the constraints to a new sheet and added a "is protein" column to meet the 15.2.1.c constraint.

```
import pandas as pd
from pulp import (
    LpProblem,
    LpMinimize,
    LpVariable,
    LpConstraint,
    lpDot,
    LpStatus,
)


ingredients = pd.read_excel(
  "data 15.2/diet.xls",
  sheet_name="ingredients"
).set_index("Foods")

constraints = pd.read_excel(
  "data 15.2/diet.xls",
  sheet_name="constraints"
).set_index("Limit")
```

Question 15.2.1

Let's start by initializing our problem.

Note: PuLP's "A Blending Problem" case study was the basis for this solution https://coin-or.github.io/pulp/CaseStudies/a_blending_problem.html.

Despite continuous servings not making sense, to match the provided answer that's what we have to use. Interestingly, the integer solution substitutes kiwi for broccoli despite having a significantly higher cost.

```
diet = LpProblem("diet", LpMinimize)
servings = LpVariable.dicts("servings", ingredients.index, 0)
```

First we need to all the objective function to the problem. I came across the `lpDot` function which is a much cleaner syntax than what was in the tutorial.

```
diet += (
    lpDot(servings.values(), ingredients.price),
    "Total Cost of Diet",
)
```

We can get the nutrient constraints using the same `lpDot` and our `constraints` dataframe.

```
for nutrient in constraints.columns:
    # total nutrients consumed under diet
    total_nutrient_consumption = lpDot(servings.values(), ingredients[nutrient])

    # add lower bound constraint
    diet += (
        total_nutrient_consumption >= constraints.loc["lower", nutrient],
        f"{nutrient}_lower_bound",
    )

    # add upper bound constraint
    diet += (
        total_nutrient_consumption <= constraints.loc["upper", nutrient],
        f"{nutrient}_upper_bound",
    )
```

Now, let's solve our problem and print out the solution!

```
diet_solve = LpStatus[diet.solve()]
print(f"Solution status: {diet_solve}")
```

```
## Solution status: Optimal
```

```
if diet_solve == "Optimal":
    print(f"Diet found with total cost ${diet.objective.value():.2f}")
    for v in diet.variables():
        if v.varValue != 0:
            print(v.name, "=", v.varValue)
```

```
## Diet found with total cost $4.34
## servings_Celery,_Raw = 52.64371
## servings_Frozen_Broccoli = 0.25960653
## servings_Lettuce,Iceberg,Raw = 63.988506
## servings_Oranges = 2.2929389
## servings_Poached_Eggs = 0.14184397
## servings_Popcorn,Air_Popped = 13.869322
```

We match the solution provided, so the problem is formulated correctly!

Question 15.2.2

Let's start out with the same model as before.

```
diet = LpProblem("diet", LpMinimize)
servings = LpVariable.dicts("servings", ingredients.index, 0)

# objective function
diet += (
    lpDot(servings.values(), ingredients.price),
    "Total Cost of Diet",
)

# Nutritional value constraints
for nutrient in constraints.columns:
    # total nutrients consumed under diet
    total_nutrient_consumption = lpDot(servings.values(), ingredients[nutrient])

    # add lower bound constraint
    diet += (
        total_nutrient_consumption >= constraints.loc["lower", nutrient],
        f"{nutrient}_lower_bound",
    )

    # add upper bound constraint
    diet += (
        total_nutrient_consumption <= constraints.loc["upper", nutrient],
        f"{nutrient}_upper_bound",
    )
```

Now let's add our new "preference" constraints. We need to create a "is chosen" variable to make a minimum serving size requirement and link our two variables together. In the lecture there was a natural maximum to use as the linking value; however, the only way I could think of doing this was repeating a nutrition constraint.

```
# is chosen needed for multiple constraints
is_chosen = LpVariable.dicts("is_chosen", ingredients.index, cat="Binary")

# minimum portion constraint
# use calories as a linking constant
for food, serving in servings.items():
    diet += (
      ingredients.loc[food, "Calories"] * serving <=
      constraints.loc["upper", "Calories"] * is_chosen[food],
      f"{food}: serving/chosen constraint"
    )

    diet += serving >= 0.1*is_chosen[food], f"{food}: minimum serving size constraint"

# Vegetable variety constraint
diet += (
  is_chosen['Frozen Broccoli'] + is_chosen['Celery, Raw'] <= 1,
  "Picky veg contraint"
)

# Protein variety constraint
diet += (
  lpDot(is_chosen.values(), ingredients["is_protein"].astype(int)) >= 3,
  "Varied protein constraint"
)
```

And finally, let's solve the problem

```
diet_solve = LpStatus[diet.solve()]
print(f"Solution status: {diet_solve}")
```

```
## Solution status: Optimal
```

```
if diet_solve == "Optimal":
    print(f"Diet found with total cost ${diet.objective.value():.2f}")
    for v in diet.variables():
        if (v.varValue != 0) and v.name.startswith("servings"):
            print(v.name, "=", v.varValue)
```

```
## Diet found with total cost $4.51
## servings_Celery,_Raw = 42.399358
## servings_Kielbasa,Prk = 0.1
## servings_Lettuce,Iceberg,Raw = 82.802586
## servings_Oranges = 3.0771841
## servings_Peanut_Butter = 1.9429716
## servings_Poached_Eggs = 0.1
## servings_Popcorn,Air_Popped = 13.223294
## servings_Scrambled_Eggs = 0.1
```