supplychaindataanalytics.com

Multi-objective LP with PuLP in Python - SCDA

Linnart Felkl M.Sc.

7-9 minutes

In some of my posts I used lpSolve or FuzzyLP in R for solving linear optimization problems. I have also used PuLP and SciPy.optimize in Python for solving such problems. In all those cases the problem had only one objective function.

In this post I want to provide a coding example in Python, using the PuLP module for solving a multi-objective linear optimization problem.

A multi-objective linear optimization problem is a linear optimization problem with more than just one objective function. This area of linear programming is also referred to as multi-objective linear programming or multi-goal linear programming.

Below I stated an examplaric multi-objective linear optimization problem with two objective functions:

$$
\begin{aligned}
\max_{x_1,x_2} \quad & 2x_1 + 3x_2 \\
\max_{x_1,x_2} \quad & 4x_1 - 2x_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \leq 10 \\
& 2x_1 + x_2 \leq 15 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{R}
\end{aligned}
$$

Assuming that in above problem statement the two objective

functions represent two different goals, such as e.g. service level and profit margin of some product portfolio, I test two alternative approaches for solving this problem.

The first approach will be to solve for one of the objectives, then fix the problem at the optimal outcome of that first problem by adding an additional constraint to a second optimization run where I will then maximize the second objective function (subject to the constraint of keeping the optimal objective value to the first sub-problem).

The second approach will be to add the two objectives together, i.e. to merge them into one objective function by applying weights. By sampling the weights and solving the combined problem for each sampled weight the optimal outcome can be reviewed in dependency of the weights.

Approach 1: Maximizing for one objective, then adding it as a constraint and solving for the other objective

Using PuLP I maximize the first objective first, then I add that objective function as a constraint to the original problem and maximize the second objective subject to all constraints, including that additional constraint.

In mathematical syntax, the problem we solve first can be stated as follows:

$$
\begin{aligned}
\max_{x_1, x_2} \quad & 2x_1 + 3x_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \leq 10 \\
& 2x_1 + x_2 \leq 15 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{R}
\end{aligned}
$$

Here is the implementation of above problem statement in Python, using the PuLP module:

```
# first, import PuLP
import PuLP

# then, conduct initial declaration of problem
linearProblem = PuLP.LpProblem("Maximizing for first
objective",PuLP.LpMaximize)

# delcare optimization variables, using PuLP
x1 = PuLP.LpVariable("x1",lowBound = 0)
x2 = PuLP.LpVariable("x2",lowBound = 0)

# add (first) objective function to the linear problem statement
linearProblem += 2*x1 + 3*x2

# add the constraints to the problem
linearProblem += x1 + x2 <= 10
linearProblem += 2*x1 + x2 <= 15

# solve with default solver, maximizing the first objective
solution = linearProblem.solve()

# output information if optimum was found, what the maximal
objective value is and what the optimal point is
print(str(PuLP.LpStatus[solution])+" ; max value =
"+str(PuLP.value(linearProblem.objective))+
      " ; x1_opt = "+str(PuLP.value(x1))+
      " ; x2_opt = "+str(PuLP.value(x2)))
```

```
Optimal ; max value = 30.0 ; x1_opt = 0.0 ; x2_opt = 10.0
```

Now, I re-state the original problem such that the second objective function is maximized subject to an additional constraint. That additional constraint requests that the first

objective must be at least 30. Using mathematical syntax the problem I now solve can be stated as follows:

$$\begin{aligned} \max_{x_1,x_2} \quad & 4x_1 - 2x_2 \\ \text{s.t.} \quad & \\ & x_1 + x_2 \leq 10 \\ & 2x_1 + x_2 \leq 15 \\ & 2x_1 + 3x_2 \geq 30 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{R} \end{aligned}$$

Here is the implementation of above problem statement in Python, using PuLP:

```
# remodel the problem statement
linearProblem = PuLP.LpProblem("Maximize second objective",PuLP.LpMaximize)
linearProblem += 4*x1 - 2*x2
linearProblem += x1 + x2 <= 10
linearProblem += 2*x1 + x2 <= 15
linearProblem += 2*x1 + 3*x2 >= 30

# review problem statement after remodelling
linearProblem
```

```
Maximize_second_objective:
MAXIMIZE
4*x1 + -2*x2 + 0
SUBJECT TO
_C1: x1 + x2 <= 10

_C2: 2 x1 + x2 <= 15

_C3: 2 x1 + 3 x2 >= 30

VARIABLES
```

x1 Continuous
x2 Continuous

Now, I solve this problem, using the default solver in PuLP:

```
# apply default solver
solution = linearProblem.solve()


# output a string summarizing whether optimum was found, and if so what the optimal solution
print(str(PuLP.LpStatus[solution])+" ; max value = "+str(PuLP.value(linearProblem.objective))+
      " ; x1_opt = "+str(PuLP.value(x1))+
      " ; x2_opt = "+str(PuLP.value(x2)))
```

Optimal ; max value = -19.999999999995993 ; x1_opt = 1.0018653e-12 ; x2_opt = 10.0

This approach suggests that x1 = 0 and x2 = 10 is the the optimal solution. The optimal objective values would be 30 for objective one, and -20 for objective two.

Approach 2: Combine objectives, using sampled weights and iterations with defined step-size

When applying this approach, we will restate the original problem as follows:

$$\max_{x_1,x_2} \quad \alpha(2x_1 + 3x_2) + (1-\alpha)(4x_1 - 2x_2)$$

s.t.

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ 2x_1 + x_2 &\leq 15 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{R} \\ \alpha &\in [0;1] \end{aligned}$$

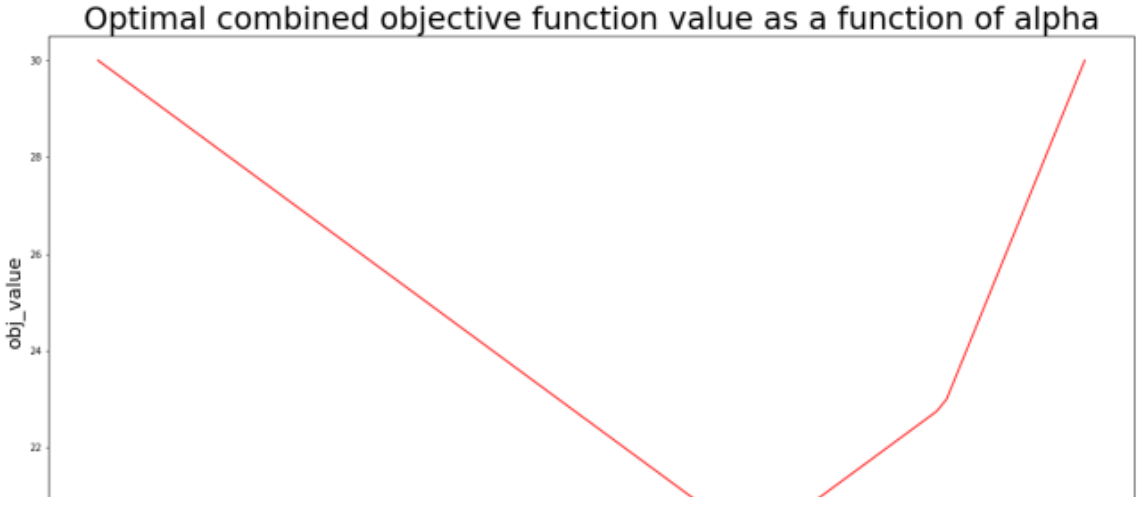
The question now is how to choose *α.*

A typical approach in a situation like this is to identify an efficient frontier. In economics this is e.g. known as “pareto optimality”. To construct such an approach I sample alpha in steps of 0.01. For each value of alpha I restate the problem, using PuLP – and solve it.

I store my results in a list and visualize the outcome using matplotlib.pyplot:

```
# import matplotlib.pyplot
import matplotlib.pyplot as plt

# import pandas and numpy for being able to store data in DataFrame format
import numpy as np
import pandas as pd

# define step-size
stepSize = 0.01

# initialize empty DataFrame for storing optimization outcomes
solutionTable = pd.DataFrame(columns=["alpha","x1_opt","x2_opt","obj_value"])

# iterate through alpha values from 0 to 1 with stepSize, and write PuLP solutions into solutionTable
for i in range(0,101,int(stepSize*100)):
    # declare the problem again
    linearProblem = PuLP.LpProblem("Multi-objective linear maximization",PuLP.LpMaximize)
    # add the objective function at sampled alpha
    linearProblem += (i/100)*(2*x1+3*x2)+(1-i/100)*(4*x1-2*x2)
    # add the constraints
```

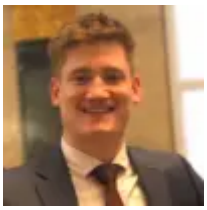
```
        linearProblem += x1 + x2 <= 10
        linearProblem += 2*x1 + x2 <= 15
        # solve the problem
        solution = linearProblem.solve()
        # write solutions into DataFrame
        solutionTable.loc[int(i/(stepSize*100))] = [i/100,
                                                    PuLP.value(x1),
                                                    PuLP.value(x2),

PuLP.value(linearProblem.objective)]

# visualize optimization outcome, using matplotlib.pyplot
# -- set figure size
plt.figure(figsize=(20,10))
# -- create line plot
plt.plot(solutionTable["alpha"],solutionTable["obj_value"],color="red")
# -- add axis labels
plt.xlabel("alpha",size=20)
plt.ylabel("obj_value",size=20)
# -- add plot title
plt.title("Optimal combined objective function value as a function
of alpha",size=32)
# -- show plot
plt.show()
```



To complete this article I print out the head of the optimization outcome DataFrame table:

solutionTable.head()

	alpha	x1_opt	x2_opt	obj_value
0	0.00	7.5	0.0	30.00
1	0.01	7.5	0.0	29.85
2	0.02	7.5	0.0	29.70
3	0.03	7.5	0.0	29.55
4	0.04	7.5	0.0	29.40



Data scientist focusing on simulation, optimization and modeling in R, SQL, VBA and Python