Departamento de Engenharia Informática e de Sistemas

**Metodologias de Otimização e Apoio à Decisão**

# Resolução de problemas de PL em Python

## - Parte V -

Para resolver problemas de programação linear multi-objetivo (PLMO) com a biblioteca PULP vamos usar duas abordagens propostas em:
https://www.supplychaindataanalytics.com/multi-objective-linear-optimization-with-pulp-in-python/
Para simplificar, vamos considerar problemas com apenas duas funções objetivo.
A primeira abordagem consiste em otimizar um dos objetivos e depois adicioná-lo como uma restrição na otimização do outro objetivo. A segunda abordagem consiste em criar uma nova função objetivo que é uma soma ponderada das duas funções objetivo originais.
O exemplo que se segue pretende ilustrar a aplicação destas abordagens.

**EXEMPLO 6**

Considere o seguinte problema de programação linear multi-objetivo (retirado da fonte anteriormente referida):

$$
\begin{array}{l}
\text{Max } z_1 = 2\,x_1 + 3\,x_2 \\
\text{Max } z_2 = 4\,x_1 - 2\,x_2 \\
\text{sujeito a} \\
\quad x_1 + x_2 \le 10 \\
\quad 2\,x_1 + x_2 \le 15 \\
\quad x_1 \ge 0,\ x_2 \ge 0
\end{array}
$$

**<u>1ª Abordagem</u>**

De acordo com esta abordagem, maximizamos o primeiro objetivo, depois adicionamos a otimização dessa função objetivo como uma restrição ao problema original e maximizamos o segundo objetivo sujeito a todas as restrições, incluindo a restrição adicional.

O problema a resolver primeiramente é pois:

$$
\begin{array}{l}
\text{Max } z_1 = 2\,x_1 + 3\,x_2 \\
\text{sujeito a} \\
\quad x_1 + x_2 \le 10 \\
\quad 2\,x_1 + x_2 \le 15 \\
\quad x_1 \ge 0,\ x_2 \ge 0
\end{array}
$$

A implementação em Python é a que se mostra em seguida.

```python
"""
1º Problema de PLMO / 1st MOLP problem
"""

# Importar biblioteca / Import library
from pulp import *

# Criar modelo / Create model
model=LpProblem("PLMO 1/MOLP 1",LpMaximize)

# Criar variáveis de decisão / Create decision variables
x1=LpVariable("x1",lowBound=0)
x2=LpVariable("x2",lowBound=0)

# Adicionar função objetivo ao modelo / Add objective function to model
model+=2*x1 + 3*x2

# Adicionar restrições ao modelo / Add constraints to model
model+= x1 + x2 <= 10
model+= 2*x1 + x2 <= 15

print(model)

# Resolver modelo / Solve model
model.solve()

# Visualizar resultados / Visualize results
print("-------- Resultados / Results ---------")
print(f"Status = {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {value(model.objective)}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name,constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```

O resultado que surge na consola é.

```
In [1]: runfile('C:/Users/Teresa/Arquivos/MOAD/AULAS PRÁTICAS/PYTHON/
wdir='C:/Users/Teresa/Arquivos/MOAD/AULAS PRÁTICAS/PYTHON/Projetos')
PLMO_1/MOLP_1:
MAXIMIZE
2*x1 + 3*x2 + 0
SUBJECT TO
_C1: x1 + x2 <= 10

_C2: 2 x1 + x2 <= 15

VARIABLES
x1 Continuous
x2 Continuous

-------- Resultados / Results ---------
Status = 1 <=> Optimal
z* = 30.0
x1* = 0.0
x2* = 10.0
_C1: 0.0
_C2: -5.0
```

Agora, reformulamos o problema original de modo a que a segunda função objetivo seja maximizada sujeita a uma restrição adicional. Essa restrição adicional exige que o primeiro objetivo tenha pelo menos o valor 30.

O modelo reformulado que agora teremos que resolver é o seguinte:

$$\text{Max } z_2 = 4\,x_1 - 2\,x_2$$

sujeito a

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ 2\,x_1 + x_2 &\leq 15 \\ 2\,x_1 + 3\,x_2 &\geq 30 \\ x_1 \geq 0,\ x_2 &\geq 0 \end{aligned}$$

O código em Python, depois das alterações, é o seguinte:

```python
"""
1º Problema de PLMO / 1st MOLP problem
"""

# Importar biblioteca / Import library
from pulp import *

# Criar modelo / Create model
model=LpProblem("PLMO 1/MOLP 1",LpMaximize)

# Criar variáveis de decisão / Create decision variables
x1=LpVariable("x1",lowBound=0)
x2=LpVariable("x2",lowBound=0)

# Adicionar função objetivo ao modelo / Add objective function to model
model+=4*x1 - 2*x2       # Função objetivo 2 / Objective function 2

# Adicionar restrições ao modelo / Add constraints to model
model+= x1 + x2 <= 10
model+= 2*x1 + x2 <= 15
model+=2*x1 + 3*x2 >= 30 # Função objetivo 1 / Objective function 1
print(model)

# Resolver modelo / Solve model
model.solve()

# Visualizar resultados / Visualize results
print("-------- Resultados / Results ---------")
print(f"Status = {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {value(model.objective)}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name,constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```

O resultado final que surge na consola é.

```
PLMO_1/MOLP_1:
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

-------- Resultados / Results ---------
Status = 1 <=> Optimal
z* = -20.0
x1* = 0.0
x2* = 10.0
_C1: 0.0
_C2: -5.0
_C3: 0.0
```

<u>Interpretação dos resultados</u>

A abordagem usada sugere que $x_1 = 0$ e $x_2 = 10$ é a melhor solução do problema. Os valores das funções objetivo são, $z_1 = 30$ (confirmado por _C3=0) e $z_2 = -20$. Se tivéssemos otimizado $z_2$ em primeiro lugar e a considerássemos como restrição para maximizar $z_1$, a melhor solução seria $x_1 = 7.5$ e $x_2 = 0$, correspondendo a $z_1 = 15$ e $z_2 = 30$. Em ambos os casos, verifica-se que a melhor solução para o problema corresponde à solução ótima de um dos objetivos que sabemos que são soluções eficientes / não dominadas. No entanto, não é apresentada nenhuma alternativa que corresponda a um compromisso intermédio entre os dois objetivos.

## <u>2ª Abordagem</u>

A segunda abordagem consiste em transformar o problema multi-objetivo num problema mono-objetivo, sendo a nova função objetivo uma soma ponderada das duas funções objetivo originais. Concretamente, será definida uma função z da seguinte forma:

$$\boxed{z = \alpha * z_1 + (1-\alpha) * z_2, \text{ com } \alpha \in [0,1]}$$

Desta forma, quando $\alpha$=0, o valor de z coincidirá com $z_1$. Quando o valor de $\alpha$=1, o valor de z será igual a $z_2$.

A implementação que se segue, resolve o problema enunciado anteriormente usando este método da “soma ponderada”. Os resultados poderão ser visualizados sob a forma de tabela ou de gráfico.

```
"""
1ยบ Problema de PLMO / 1st MOLP problem
"""
# Importar biblioteca PuLP / Import PuLP library
from pulp import *

# Importar sub-modulo Pyplot da biblioteca Matplotlib / Import sub-module Pyplot from libray Matplotlib
import matplotlib.pyplot as plt

# Importar biblioteca Pandas para poder guardar dados em formato Dataframe
# Import Pandas library for being able to store data in DataFrame format
import pandas as pd

# Inicializar um DataFrame vazio para guardar resultados da otimização
# Initialize empty DataFrame for storing optimization results
results = pd.DataFrame(columns=["alpha","x1_opt","x2_opt","z1","z2"])

# Criar variáveis de decisão / Create decision variables
x1=LpVariable("x1",lowBound=0)
x2=LpVariable("x2",lowBound=0)

# Definir tamanho do passo (incremento) / Define step-size
step = 0.01

# Iterar para valores de alpha (peso) entre 0 e 1 com incremento step, e guardar resultados no DataFrame
# Iterate through alpha (weight) values from 0 to 1 with step, and save results into DataFrame
for i in range(0,101,int(step*100)):
        # Criar novo modelo / Create new model
        model=LpProblem("PLMO_1/MOLP_1",LpMaximize)
        # Adicionar funcao objetivo como soma ponderada de z1 e z2: z = alpha*z1 + (1-alpha)*z2
        # Add objective function as a weighted sum of z1 and z2: z = alpha*z1 + (1-alpha)*z2
        alpha=i/100
        model += alpha*(2*x1+3*x2)+(1-alpha)*(4*x1-2*x2)
        # Adicionar restricoes / Add constraints
        model += x1 + x2 <= 10
        model += 2*x1 + x2 <= 15
        # Resolver problema / Solve problem
        model.solve()
        # Guardar resultados no DataFrame / Save results into DataFrame
        results.loc[i] = [alpha,
                          value(x1),
                          value(x2),
                          value(alpha*(2*x1+3*x2)),
                          value((1-alpha)*(4*x1-2*x2))]

# Visualizar resultados de otimizacao numa tabela / Visualize optimization results in a table
for i in range(0,100,15):
       print(results[i:i+15])
       input("Press ENTER key to continue...")

# Visualizar resultados de otimizacao num grafico
# Visualize optimization results in a plot
# -- Inicializar tamanho da figura / Set figure size
plt.figure(figsize=(20,10))
# -- Criar grafico de linhas / Create line plot
plt.plot(results["alpha"],results["z1"],color="red",label="z1")
plt.plot(results["alpha"],results["z2"],color="blue",label="z2")
plt.legend(loc="upper left")
# -- Adicionar legendas dos eixos / Add axis labels
plt.xlabel("alpha (weight)",size=20)
plt.ylabel("objective_values",size=20)
# -- Adicionar tรญtulo ao grรกfico / Add plot title
plt.title("Objective Functions z1 and z2",size=32)
# -- Mostrar grafico / Show plot
plt.show()
```

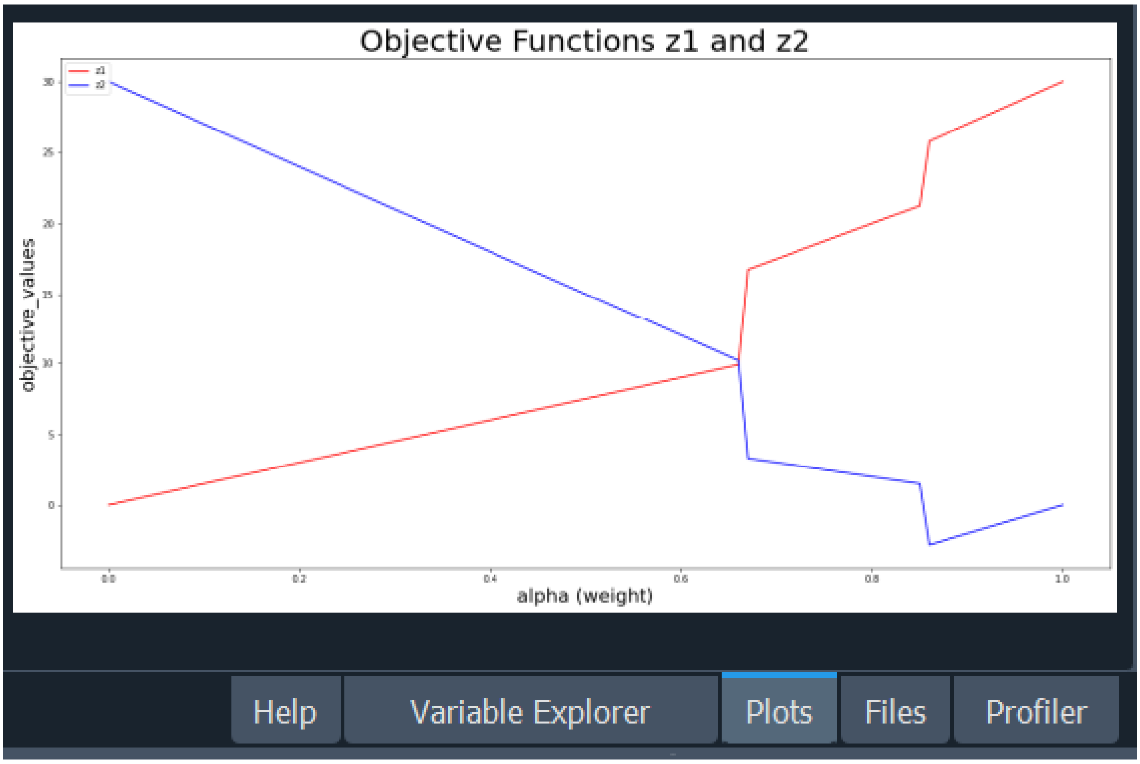
Note-se que para construir o gráfico recorremos ao sub-módulo Pyplot da biblioteca Matplotlib. Por outro lado, recorremos à biblioteca Pandas para podermos criar um objeto do tipo *DataFrame*.

Os resultados que surgirão na consola em formato de tabela serão:

```
In [48]: runfile('C:/Users/Teresa/Arquivos/MOAD/AULAS PRÁTICAS/
Projetos/Exemplo 6-v2.py', wdir='C:/Users/Teresa/Arquivos/MOAD/
PRÁTICAS/PYTHON/Projetos')
    alpha  x1_opt  x2_opt    z1    z2
0    0.00     7.5     0.0  0.00  30.0
1    0.01     7.5     0.0  0.15  29.7
2    0.02     7.5     0.0  0.30  29.4
3    0.03     7.5     0.0  0.45  29.1
4    0.04     7.5     0.0  0.60  28.8
5    0.05     7.5     0.0  0.75  28.5
6    0.06     7.5     0.0  0.90  28.2
7    0.07     7.5     0.0  1.05  27.9
8    0.08     7.5     0.0  1.20  27.6
9    0.09     7.5     0.0  1.35  27.3
10   0.10     7.5     0.0  1.50  27.0
11   0.11     7.5     0.0  1.65  26.7
12   0.12     7.5     0.0  1.80  26.4
13   0.13     7.5     0.0  1.95  26.1
14   0.14     7.5     0.0  2.10  25.8

Press any key to continue...
    alpha  x1_opt  x2_opt    z1    z2
15   0.15     7.5     0.0  2.25  25.5
```

E em formato de gráfico:




## Interpretação dos resultados

Pela análise da tabela percebe-se que, apesar dos 100 valores diferentes de $z_1$ e de $z_2$, decorrentes da variação de $\alpha$, apenas são apresentadas 3 soluções (pares de valores $x_1$ e $x_2$) que correspondem aos vértices da região eficiente (sugere-se resolução gráfica do problema para melhor compreensão). Note-se que as restantes soluções eficientes podem ser obtidas como combinação linear convexa destas últimas.