Departamento de Engenharia Informática e de Sistemas

Metodologias de Otimização e Apoio à Decisão

Resolução de problemas de PL em Python

- Parte III -

EXEMPLO 4

Considere o seguinte problema:

O Sr. Francisco dedica-se à criação e venda de cães de determinada raça, com bastante procura no mercado. Como pretende que os seus animais cresçam saudáveis e bonitos, ele sabe que deve proporcionar-lhes uma alimentação equilibrada. Na verdade, o Sr. Francisco tem à sua disposição dois tipos de rações, A e B, com caraterísticas e preços diferentes. Cada kg de ração A custa 1 € e cada kg de ração B custa 0.5 €, sendo que a composição em termos de nutrientes é a da tabela seguinte:

	Rações	
Nutrientes	**A** (g/kg)	**B** (g/kg)
Sais minerais	20	50
Vitaminas	50	10
Cálcio	30	30

Segundo os veterinários, as quantidades mínimas de nutrientes, por semana, para uma alimentação equilibrada são as seguintes:

Nutrientes	**Quantidade mínima requerida (em g)**
Sais minerais	200
Vitaminas	150
Cálcio	210

Considerando x_1 e x_2 as quantidades (em kg) de ração dos tipos A e B, respetivamente, a dar a cada animal por semana, e que o objetivo é minimizar o custo semanal da alimentação de cada cachorro (mas cumprindo os requisitos nutricionais), o modelo matemático pode ser definido como se apresenta em seguida.

Minimizar $z = x_1 + 0.5\, x_2$ (*Custo semanal com alimentação*)

sujeito a

$20\, x_1 + 50\, x_2 \geq 200$ (*Sais minerais*)

$50\, x_1 + 10\, x_2 \geq 150$ (*Vitaminas*)

$30\, x_1 + 30\, x_2 \geq 210$ (*Cálcio*)

$x_1 \geq 0,\ x_2 \geq 0$

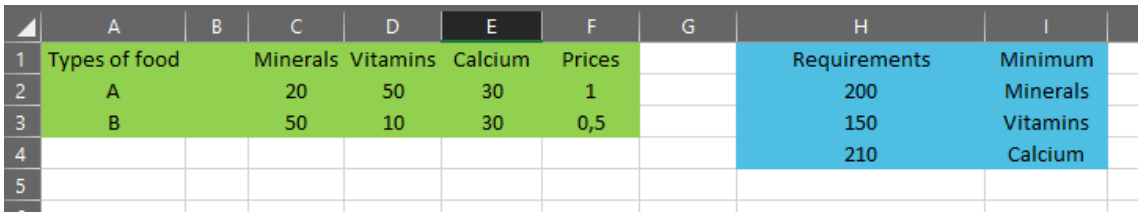
Neste exemplo, vamos inserir os dados do problema num ficheiro EXCEL e depois criar e resolver o modelo em Python importando os dados desse mesmo ficheiro. Esta forma de definir os dados, torna-se mais eficiente em problemas de grande dimensão.

Para esta implementação vamos recorrer ao Pandas que é um *package* do Python, vocacionado para manipulação e análise de dados. Em particular, oferece estruturas de dados e operações para manipulação de tabelas numéricas e séries temporais.

Usando a função *read_excel()* do Pandas, os dados da folha EXCEL são devolvidos num objeto do tipo *DataFrame*.

Podemos pensar num objeto *DataFrame* como uma espécie de dicionário, só que, enquanto um dicionário mapeia uma chave num valor, um objeto *DataFrame* mapeia um nome de uma coluna numa série de dados dessa mesma coluna.

O ficheiro EXCEL criado com os dados do problema em causa designa-se por "Data_EX4.xlsx" e tem o conteúdo ilustrado na imagem seguinte.



	A	B	C	D	E	F	G	H	I
1	Types of food		Minerals	Vitamins	Calcium	Prices		Requirements	Minimum
2	A		20	50	30	1		200	Minerals
3	B		50	10	30	0,5		150	Vitamins
4								210	Calcium
5									

Podemos agora desenvolver o código em Python, começando pela importação das bibliotecas Pandas e PuLP. No caso da primeira, apenas é necessária a função *read_excel()*.

Iniciamos também a leitura do ficheiro EXCEL começando pelos dados da área verde que dizem respeito à informação que depende diretamente das rações A e B. Os dados são devolvidos para o objeto *df1* do tipo *DataFrame*.

```
"""
Alimentação dos cachorros / Feeding the puppies
"""
from pandas import read_excel
from pulp import *

# Lê área verde do ficheiro EXCEL / Read green area from EXCEL file
df1 = read_excel("Data_EX4.xlsx",nrows=2,usecols=(['Types of food','Minerals',
                                                   'Vitamins','Calcium','Prices']))
print("==> Dataframe 1\n",df1)
```

Na função *read_excel()*, o 1º parâmetro é o nome do ficheiro, o 2º parâmetro é o nº de linhas de dados a ler do ficheiro (assume-se a existência de um *header*) e o 3º parâmetro é uma lista com os nomes das colunas a ler. Relembrar que nos objetos DataFrame os dados das colunas são mapeados pelo nome das colunas.

O resultado da instrução *print()* anterior será:

```
==> Dataframe 1
   Types of food  Minerals  Vitamins  Calcium  Prices
0              A        20        50       30     1.0
1              B        50        10       30     0.5
```

Em seguida é criada uma lista com os tipos de ração.

```
# Cria lista com tipos de ração/ Create list with types of food
food_types = list(df1['Types of food'])
print("==> Food types\n",food_types)
```

Como resultado, surge na consola:

```
==> Food types
 ['A', 'B']
```

A lista anterior servirá como lista de chaves em dicionários que vão conter os dados da quantidade de sais minerais, vitaminas e cálcio presentes em cada um dos tipos de ração. Será também criado um dicionário para os preços das rações.

```
# Cria dicionário de sais minerais / Create a dictionary of minerals
# indexado por tipo de ração       / indexed by food type
minerals = dict(zip(food_types,df1['Minerals']))
print("==> Minerals\n",minerals)

# Cria dicionário de vitaminas / Create a dictionary of vitamins
# indexado por tipo de ração   / indexed by food type
vitamins = dict(zip(food_types,df1['Vitamins']))
print("==> Vitamins\n",vitamins)

# Cria dicionário de cálcio  / Create a dictionary of calcium
# indexado por tipo de ração / indexed by food type
calcium = dict(zip(food_types,df1['Calcium']))
print("==> Calcium\n",calcium)

# Cria dicionário de preços  / Create a dictionary of prices
# indexado por tipo de ração / indexed by food type
prices = dict(zip(food_types,df1['Prices']))
print("==> Prices\n",prices)
```

O resultado na consola será o seguinte:

```
==> Minerals
 {'A': 20, 'B': 50}
==> Vitamins
 {'A': 50, 'B': 10}
==> Calcium
 {'A': 30, 'B': 30}
==> Prices
 {'A': 1.0, 'B': 0.5}
```

Nas instruções anteriores, foi usada a função *zip()* que retorna uma sequência de tuplos, em que o $i^{ésimo}$ tuplo contém o $i^{ésimo}$ elemento de cada uma das sequências passadas como argumentos. Se as sequências de entrada tiverem tamanho desigual, a geração da sequência de saída terminará quando se atingir a dimensão da menor.

Portanto, fornecendo como argumentos à função *zip(),* a lista do tipo de rações (*food_types*) e uma série de valores de uma coluna do objeto *df1* (que irá variar entre os diversos tipos de nutrientes e preços), esta função criará um conjunto de tuplos em que o 1º elemento será o tipo de ração e o 2º elemento será a quantidade do nutriente/preço.

Aplicando a função *dict()* a esta lista de tuplos, é criado um dicionário cuja chave será o 1º elemento do tuplo (tipo de ração) e o valor será o 2º elemento do tuplo (quantidade do nutriente/preço).

Criados os dicionários anteriormente referidos, vamos ler do ficheiro EXCEL os dados da área azul que são relativos aos requerimentos mínimos de nutrientes (lado direito das restrições).

```
# Lê área azul do ficheiro EXCEL / Read blue area from EXCEL file
df2 = read_excel("Data_EX4.xlsx",nrows=3,usecols=(['Requirements']))
print("==> Dataframe 2\n",df2)
```

O resultado que aparecerá na consola será:

```
==> Dataframe 2
    Requirements
0            200
1            150
2            210
```

A partir do objeto *df2*, criamos uma lista com os valores dos requerimentos mínimos.

```
# Cria lista de requisitos nutricionais / Create a list of nutricional requirements
req = list(df2['Requirements'])
print("==> Requirements\n",req)
```

Na consola surgirá:

```
==> Requirements
 [200, 150, 210]
```

Em seguida, criamos o modelo e as variáveis de decisão.

```
# Cria modelo / Create model
model=LpProblem("Dieta_cachorros/Puppies_diet",LpMinimize)

# Cria variáveis de decisão / Create decision variables
x = LpVariable.dicts("x",food_types,lowBound=0)
```

Para a criação destas últimas, usamos o método *.dicts()* da classe *LpVariable*, o qual cria um dicionário de objetos *LpVariable* com os parâmetros especificados. Neste exemplo, os três parâmetros são: “x” - prefixo para o nome de cada variável criada; *food_types* - lista das chaves para o dicionário de variáveis e a parte principal do próprio nome da variável; *lowBound=0* - limite inferior dos valores das variáveis.

Segue-se a definição da função objetivo e restrições.

```
# Cria função objetivo / Create objective function
model += lpSum([prices[i]*x[i] for i in food_types])

# Cria restrições / Create constraints
model += lpSum([minerals[i] * x[i] for i in food_types]) >= req[0]
model += lpSum([vitamins[i] * x[i] for i in food_types]) >= req[1]
model += lpSum([calcium[i] * x[i] for i in food_types])  >= req[2]
```

O restante código é semelhante ao usado nos exemplos anteriores.

```
# Resolve 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 do bloco de intruções anterior surgirá finalmente na consola.

```
-------- Resultados / Results ---------
Status = 1 <=> Optimal
z* = 4.5
x_A* = 2.0
x_B* = 5.0
_C1: 90.0
_C2: 0.0
_C3: 0.0
```

Interpretação dos resultados

O Sr. Francisco deverá alimentar cada cachorro com 2 kg de ração A e 5 Kg de ração B, por semana, com um custo mínimo de 4.5€. O valor que surge na 1ª restrição (_C1) significa que cada animal irá consumir mais 90g de sais minerais, do que o valor mínimo recomendado.