

Tutorial - Gurobi (Python)

Vamos implementar o modelo de **Múltiplas Mochilas Binárias**.

$$\max(FO) \sum_{m=1}^{Mochilas} \sum_{i=1}^{Itens} valor_i \cdot x_{im}$$

$$\text{sujeito a (A)} \sum_{i=1}^{Itens} peso_i \cdot x_{im} \leq capacidade_m, \quad \forall m = 1 : Mochilas$$

$$(B) \sum_{m=1}^{Mochilas} x_{im} \leq 1, \quad \forall i = 1 : Itens$$

$$(C) x_{im} \in \{0,1\}, \quad \forall i = 1 : Itens, m = 1 : Mochilas$$

Explicando o modelo

O modelo tem um conjunto de mochilas e um conjunto de itens. Cada item tem um valor monetário e um peso. Cada mochila tem uma capacidade. O problema é escolher a melhor **combinação** de itens para serem colocados em cada mochila, de modo que se maximize o valor total e que não exceda a capacidade das mochilas.

- O objetivo é maximizar o valor total dos itens colocados nas mochilas (função objetivo **FO**).
- Para cada mochila **m**, a soma dos pesos dos itens não deve ultrapassar a capacidade dessa mochila **m** (restrições em **A**).
- Cada item **i** só deve aparecer, **no máximo, uma** vez em cada mochila **m** (restrições em **B**).
- As variáveis de decisão x_{im} são binárias, só podem assumir os valores **zero** ou **um** (restrições em **C**).

A variável x_{im} é igual a **1** quando o item **i** for colocado na mochila **m**.

Implementação

1. Percebemos a existência de 2 índices nas variáveis do modelo matemático. Portanto, vamos criar 2 listas de índices:
 - uma lista de índices para as mochilas (mochila1, mochila2, etc.)
 - uma lista de índices para os itens (item1, item2, etc.)
2. Os dados do problema são:
 - a lista (vetor) dos valores dos itens
 - a lista (vetor) dos pesos dos itens
 - a lista (vetor) das capacidades das mochilas

portanto, teremos 3 listas de dados

3. Para cada lista de dados do **Passo 2**, precisamos criar um dicionário com essas informações. A chave do dicionário será um índice criado no **Passo 1**.

- dict_valores: [item] -> [valor do item]
- dict_pesos: [item] -> [peso do item]
- dict_capacidades: [mochila] -> [capacidade da mochila]

4. Constrói-se o modelo, com objetos e métodos do Gurobi.

Vamos ao código

```
import gurobipy as gp

# PASSO 1: Criando os índices do modelo
# temos 4 mochilas e 20 itens.
# Como Python é indexado em 0, vamos somar 1 para que a contagem comece em 1
id_mochilas = [f"Mochila_{m + 1}" for m in range(4)]
id_itens = [f"Item_{i + 1}" for i in range(20)]

# imprime os ids das mochilas e dos itens
# print(id_mochilas, "\n")
# print(id_itens)

# PASSO 2: Entrando com os dados do problema
vetor_valores = [244, 230, 227, 113, 185, 179, 236, 164, 213, 132,
                 210, 207, 200, 100, 163, 163, 119, 105, 163, 133]

vetor_pesos = [21, 36, 19, 17, 33, 28, 13, 26, 49, 28,
              13, 37, 46, 20, 10, 45, 43, 36, 26, 38]

vetor_capacidades = [111, 74, 113, 98]

# PASSO 3: Criando os dicionários de dados
# valores e pesos dos 20 itens
dict_valores = {id_itens[i] : vetor_valores[i] for i in range(20)}
dict_pesos = {id_itens[i] : vetor_pesos[i] for i in range(20)}

# capacidades das 4 mochilas
dict_capacidades = {id_mochilas[m] : vetor_capacidades[m] for m in range(4)}

# imprime os dicionários
# print(dict_valores, "\n")
# print(dict_pesos, "\n")
# print(dict_capacidades)
```

Até aqui usamos apenas a linguagem Python. A partir de agora, vamos ao Gurobi para construir e resolver o modelo matemático.

```
# Criando um objeto do tipo gp.Model()
modelo = gp.Model("Múltiplas Mochilas Binárias")

# As variáveis x[i,m] do modelo têm 2 índices: "itens" e "mochilas"
# O domínio das variáveis é binário (restrições C; gp.GRB.BINARY)
# vamos adicionar um array de variáveis, por isso modelo.addVars(),
# se fôssemos adicionar uma variável escalar, usaríamos modelo.addVar().
# No nosso caso, é uma matriz, indexada em "itens" e "mochilas"
x = modelo.addVars(id_itens, id_mochilas, vtype=gp.GRB.BINARY)

# Para construir a função objetivo e as restrições, usaremos os dicionários
# com os dados (Passo 3); e para os loops "for" usaremos os ids criados (Passo 1)

# Construindo a função objetivo
# o somatório é feito pelo método gp.quicksum()
# leia o código de trás pra frente:
#   for m in id_mochilas
#   for i in id_itens
#   x[i,m] * dict_valores[i]
#   quicksum()
# sense é o sentido da otimização
modelo.setObjective(
    gp.quicksum( x[i, m] * dict_valores[i] for i in id_itens for m in id_mochilas),
    sense=gp.GRB.MAXIMIZE
)

# Adicionando as restrições de capacidade (restrições A)
# Como são diversas restrições, uma para cada mochila, use modelo.addConstrs()
# Caso fosse apenas uma restrição, usaríamos modelo.addConstr()
restrs_capacidade = modelo.addConstrs(
    gp.quicksum( x[i, m] * dict_pesos[i] for i in id_itens) <= dict_capacidades[m]
    for m in id_mochilas # --> forall m=1:Mochilas
)

# Adicionando as restrições de alocação dos itens (restrições B)
restrs_alocacao = modelo.addConstrs(
    gp.quicksum( x[i, m] for m in id_mochilas ) <= 1
    for i in id_itens # --> forall i=1:Itens
)

# Para mostrar o relatório ao chamar o solver, comente a configuração abaixo
modelo.setParam( gp.GRB.Param.OutputFlag, 0 ) # --> esconde o relatório

# resolvendo o modelo
modelo.optimize()
```

Terminamos a fase de construção e resolução do modelo, só nos falta imprimir os resultados.

```
# imprime o valor ótimo da função objetivo
print("O maior valor obtido (alocação ótima), foi de")
print(round(modelo.objVal, 2)) # --> arredondando para 2 casas decimais

# Use x[i,m].X para obter o valor ótimo da variável (se 0.0 ou 1.0)
for m in id_mochilas:
    print(f"Na {m} os seguintes itens foram alocados:")
    for i in id_itens:
        if round(x[i, m].X) == 1:
            print(f"\t{ i }") # --> \t faz uma tabulação
# imprimindo as folgas (Slack) das restrições de capacidade
print(f"\tSobrou [{ round(restrs_capacidade[m].Slack) }] de espaço.")
print("") # --> pula uma linha para separar
```

```
O maior valor obtido (alocação ótima), foi de
2834.0
```

```
Na Mochila_1 os seguintes itens foram alocados:
```

```
Item_6
Item_12
Item_13
Sobrou [0] de espaço.
```

```
Na Mochila_2 os seguintes itens foram alocados:
```

```
Item_9
Item_11
Item_15
Sobrou [2] de espaço.
```

```
Na Mochila_3 os seguintes itens foram alocados:
```

```
Item_1
Item_2
Item_3
Item_4
Item_14
Sobrou [0] de espaço.
```

```
Na Mochila_4 os seguintes itens foram alocados:
```

```
Item_5
Item_7
Item_8
Item_19
Sobrou [0] de espaço.
```