Estudo da utilização de Solvers para Programação Linear

GLPK

Felipe Mello Fonseca
felipe.mello@aluno.cefet-rj.br

Solver

Solver é um **software** ou uma lib **matemática** , que possui um conjunto de implementações para **solucionar problemas de otimização** [1]. Existindo tanto soluções comercias como o **CPLEX** ,como Software livre como **GLPK**

GLPK

O pacote GLPK (GNU Linear Programming Kit) destina-se a:

- resolver problemas de programação linear em larga escala
- programação inteira mista (MIP)

GLPK

- É um conjunto de **rotinas escritas em C** e organizadas na forma de uma biblioteca que pode ser importada.
- Suporta a linguagem de modelagem **GNU MathProg**, que é um subconjunto da linguagem **AMPL**

GLPK -> **Instalação -> Ambiente**

- Utilizou-se como sistema operacional o Linux Mint 20.3
- Python como linguagem de programação versão 3.8.10
- VSCode como IDE de desenvolvimento
- GLPK 4.65-2
- PyMathProg

GLPK -> **Instalação**

Para instalar o GLPK no linux foi utilizado o comando: ***sudo apt-get install glpk-utils libglpk-dev glpk-doc***

GLPK -> **PyMathProg**

Neste trabalho usaremos uma lib que utiliza GLPK em python chamada de **PyMathProg**. Dentre as existentes foi a que até o momento apresentou-se mais acessível e com melhor suporte.

GLPK -> **PyMathProg -> Instalação**

Para instalação no Linux do PyMathProg

sudo python -m pip install pymprog

Neste trabalho como é usado a virtualenv para criação de um ambiente isolado para rodar o projeto em python. Assim a instalação do PyMathProg ficou :

pip install pymprog

GLPK -> **PyMathProg -> Código**

```
1 from pymprog import *
2 c = (10, 6, 4)
3 A = [ ( 1, 1, 1),
4       ( 9, 4, 5),
5       ( 2, 2, 6) ]
6 b = (10, 60, 30)
7 begin('basic') # begin modelling
8 verbose(True)  # be verbose
9 x = var('x', 3) #create 3 variables
10 maximize(sum(c[i]*x[i] for i in range(3)))
11 for i in range(3):
12   sum(A[i][j]*x[j] for j in range(3)) <= b[i]
13 solve() # solve the model
14 print("###>Objective value: %f"%vobj())
15 sensitivity() # sensitivity report
16 end() #Good habit: do away with the model
```

1-Bloco de dados (linhas 2-6):

A matriz e os vetores são definidos.

2-Bloco modelo (linhas 7-12):
O `begin(name)`na linha 7. Ela cria uma nova instância de modelo com o nome dado para as etapas de modelagem posteriores a serem construídas.

A linha 8 ativa a opção **verbosidade**, que permite ao PyMathProg fornecer feedbacks em cada etapa de construção.
A linha 9 define as três variáveis e as organiza em uma lista: você simplesmente fornece o nome do grupo e o número de variáveis a serem criadas, que é 3 neste caso. **Por padrão, essas variáveis são contínuas e não negativas.**
A linha 10 define o objetivo: maximizar a soma dos termos b[i]*x[i], onde i vai de 0 a 3.
As linhas 11-12 definem as restrições com um loop for.
Isso é tudo para modelagem. Agora o código segue para

GLPK -> **PyMathProg -> Código**

```
1 from pymprog import *
2 c = (10, 6, 4)
3 A = [ ( 1, 1, 1),
4       ( 9, 4, 5),
5       ( 2, 2, 6) ]
6 b = (10, 60, 30)
7 begin('basic') # begin modelling
8 verbose(True)  # be verbose
9 x = var('x', 3) #create 3 variables
10 maximize(sum(c[i]*x[i] for i in range(3)))
11 for i in range(3):
12   sum(A[i][j]*x[j] for j in range(3)) <= b[i]
13 solve() # solve the model
14 print("###>Objective value: %f"%vobj())
15 sensitivity() # sensitivity report
16 end() #Good habit: do away with the model
```

3- Bloco de relatório (linhas 13-16):
A linha 13 emite uma chamada ***solve()*** para **resolver o modelo.**
A linha 14 imprime o valor objetivo.
A linha 15 produz o **relatório de sensibilidade**.
A linha 16 chama a função ***end()*** para eliminar o modelo.

GLPK -> **PyMathProg -> Hillier -> Exemplo 3.1**

TABELA 3.1 Dados para o problema da Wyndor Glass Co.

Fábrica	Tempo de Produção por Lote (em horas) — Produto 1	Tempo de Produção por Lote (em horas) — Produto 2	Tempo de Produção Disponível por Semana (em horas)
1	1	0	4
2	0	2	12
3	3	2	18
Lucro por lote	U$ 3.000	U$ 5.000	

GLPK -> PyMathProg -> Hillier -> Exemplo 3.1 -> Código

```
from pymprog import *
begin('WYNDOR GLASS CO. ')
verbose(True)
x, y = var('x, y') # variables
#Objetivo
maximize(3000 * x + 5000 * y)
#restrições
x<=4 #disponibilidade de tempo fábrica 1
2*y <= 12 # disponibilidade de tempo fábrica 2
3*x+2*y<=18 #disponibilidade de tempo fábrica 3

solve()
print("###>Objective value: %f"%vobj())
sensitivity() # sensitivity report
end()
```

GLPK -> PyMathProg -> Hillier -> Exemplo 3.1 -> Código->saída

```
(env) felipe@felipe-VirtualBox:~/Área de Trabalho/develop/solver$ python hiller3.1.py
Max : 3000 * x + 5000 * y
R1: 2 * y <= 12
R2: 3 * x + 2 * y <= 18
GLPK Simplex Optimizer 5.0
2 rows, 2 columns, 3 non-zeros
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (2)
*     2: obj =   3.600000000e+04 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
###>Objective value: 36000.000000

PyMathProg 1.0 Sensitivity Report Created: 2022/04/11 Mon 20:23PM
================================================================================
Variable          Activity   Dual.Value    Obj.Coef   Range.From   Range.Till
--------------------------------------------------------------------------------
*x                       2            0        3000            0         7500
*y                       6            0        5000         2000 1.79769e+308
================================================================================
```

GLPK -> **PyMathProg -> Glodbarg -> Exemplo 1**

1 – O Problema das Ligas Metálicas

Uma metalúrgica deseja maximizar sua *receita bruta*. A Tabela 2.1 ilustra a proporção de cada material na mistura para a obtenção das ligas passíveis de fabricação. O preço está cotado em Reais por tonelada da liga fabricada. Também em toneladas estão expressas as restrições de disponibilidade de matéria-prima. Formular o modelo de Programação Matemática.

TABELA 2.1 RESTRIÇÕES/CUSTOS DO EXEMPLO 1

	Liga Especial de Baixa Resistência (*)	*Liga Especial de Alta Resistência (*)*	*Disponibilidade de Matéria-prima*
Cobre	0,5	0,2	16 Ton
Zinco	0,25	0,3	11 Ton
Chumbo	0,25	0,5	15 Ton
Preço de Venda (R$ por Ton)	R$3.000	R$5.000	(*) $\frac{\text{Ton de minério}}{\text{Ton de liga}}$

GLPK -> PyMathProg -> Glodbarg -> Exemplo 1-> Código

```
from pymprog import *
begin('O Problema das Ligas Metalicas')
verbose(True)
x, y = var('x, y') # variables
#Objetivo
maximize(3000 * x + 5000 * y)
#restrições
0.5*x+0.2*y <=16 #disponibilidade de cobre (toneladas)
0.25*x+0.3*y <= 11 # disponibilidade de zinco (tonelada)
0.25*x+0.5*y<=15 # disponibilidade de Chumbo(tonelada)
# metal finishing limit

solve()
print("Z: %f"%vobj())
sensitivity() # sensitivity report
end()
```

GLPK -> **PyMathProg -> Glodbarg -> Exemplo 1 -> Código->saída**

```
Max : 3000 * x + 5000 * y
R1: 0.5 * x + 0.2 * y <= 16
R2: 0.25 * x + 0.3 * y <= 11
R3: 0.25 * x + 0.5 * y <= 15
GLPK Simplex Optimizer 5.0
3 rows, 2 columns, 6 non-zeros
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (2)
*     2: obj =   1.600000000e+05 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Z: 160000.000000

PyMathProg 1.0 Sensitivity Report Created: 2022/04/12 Tue 19:13PM
================================================================================
Variable          Activity   Dual.Value    Obj.Coef   Range.From   Range.Till
--------------------------------------------------------------------------
*x                      20            0        3000         2500      4166.67
*y                      20            0        5000         3600         6000
================================================================================
```

GLPK -> **PyMathProg -> 3-Lista Exercicios**

Questão 3) Uma grande fábrica de móveis dispõe em estoque de 250 metros de tábuas, 600 metros de pranchas e 500 metros de painéis de conglomerado. A fábrica normalmente oferece uma linha de móveis composta por um modelo de escrivaninha, uma mesa de reunião, um armário e uma prateleira. Cada tipo de móvel consome uma certa quantidade de matéria-prima, conforme a Tabela 2. A escrivaninha é vendida por 100 unidades monetárias (u. m.), a mesa por 80 u.m., o armário por 120 u.m. e a prateleira por 20 u.m. Determine um modelo de Programação Linear que maximize a receita com a venda dos móveis.

Tabela 2: Informações fábrica de móveis

	Quantidade de material em metros consumidos por unidade de produto				**Disponibilidade do Recurso (m)**
	Escrivaninha	Mesa	Armário	Prateleira	
Tábua	1	1	1	4	250
Prancha	0	1	1	2	600
Painéis	3	2	4	0	500
Valor de Revenda (u.m.)	100	80	120	20	

GLPK -> PyMathProg -> 3-Lista Exercicios -> Código

```
from pymprog import *
begin('Fábrica de Móveis. ')
verbose(True)
x, y , z, w = var('x, y, z, w') # variables
#Objetivo
maximize(100 * x + 80 * y + 120 * z + 20 * w)
#restrições
x + y + z + 4 * w<= 250 #disponibilidade de tábua
y + z + 2*w <= 600 # disponibilidade de prancha
3*x + 2*y + 4*z <=500 #disponibilidade de tempo fábrica
3

solve()
print("###>Z: %f"%vobj())
sensitivity() # sensitivity report
end()
```

GLPK -> **PyMathProg -> 3-Lista Exercicios -> Código->saída**

```
Max : 100 * x + 80 * y + 120 * z + 20 * w
R1: x + y + z + 4 * w <= 250
R2: y + z + 2 * w <= 600
R3: 3 * x + 2 * y + 4 * z <= 500
GLPK Simplex Optimizer 5.0
3 rows, 4 columns, 10 non-zeros
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (4)
*     3: obj =   2.000000000e+04 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
###>Z: 20000.000000

PyMathProg 1.0 Sensitivity Report Created: 2022/04/17 Sun 21:14PM
================================================================================
Variable            Activity   Dual.Value     Obj.Coef   Range.From   Range.Till
--------------------------------------------------------------------------------
 x                         0        -17.5          100         -inf        117.5
*y                       250            0           80      68.3333 1.79769e+308
 z                         0          -35          120         -inf          155
*w                         0            0           20            0          160
================================================================================
Note: rows marked with a * list a basic variable.
```

GLPK -> PyMathProg -> Fábrica de Lâmpadas

Fábrica de lâmpadas

Os gestores de uma fábrica de lâmpadas querem fazer a programação da produção de um determinado tipo de lâmpada para um período de 3 meses. O estoque inicial é de 500 unidades e não deve haver estoque ao final do período.

A capacidade de produção em horário normal de trabalho é de 2500 lâmpadas por mês. Caso a capacidade de produção não atenda a demanda, podemos utilizar o turno extra, que comporta a produção de 1250 lâmpadas adicionais por mês.

O custo de produção unitária das lâmpadas é de R$ 3,50 em horário normal e R$ 4,25 quando feitas com hora extra. O custo estimado de estoque é de R$ 1,00 por unidade ao mês. O Quadro 39 indica a previsão de vendas dos três meses:

Quadro 39 – Informações do problema 7 – fábrica de lâmpadas

MÊS	***PREVISÃO DE VENDAS***
1	*2184*
2	*4945*
3	*2356*

Fonte: Fonte.

Deseja-se formular um modelo de programação linear que determine quanto deve ser produzido em cada mês nos turnos normal e extra de maneira a atender a previsão de demanda, minimizando o custo total e sem sobras de estoque ao final do período.

GLPK -> **PyMathProg -> Fábrica de Lâmpadas**

```
from pymprog import *
begin('Fábrica de Lâmpadas. ')
verbose(True)

y0, x1n, x1e,y1,x2n,x2e,y2,x3n,x3e,y3 = var('y0, x1n,
x1e,y1,x2n,x2e,y2,x3n,x3e,y3') # variables
#objetivo
minimize(1.00 * y0 + 3.50 * x1n + 4.25 * x1e + 1.00 *
y1 + 3.50 * x2n + 4.25 * x2e + 1.00 * y2 +3.50 * x3n +
4.25 * x3e + 1.00 * y3)
#restrições
y0 == 500
y0 + x1n + x1e - y1 == 2184
y1 + x2n + x2e - y2 == 4945
y2 + x3n + x3e - y3 == 2356
y3 = 0
```

GLPK -> PyMathProg -> Fábrica de Lâmpadas

```
x1n <= 2500
x2n <= 2500
x3n <= 2500
x1e <= 1250
x2e <= 1250
x3e <= 1250
solve()
print("###>Z: %f"%vobj())
sensitivity() # sensitivity report
end()
```

GLPK -> **PyMathProg -> Hillier -> Fábrica de Lâmpadas -> Código->saída**

```
Min : y0 + 3.5 * x1n + 4.25 * x1e + y1 + 3.5 * x2n + 4.25 * x2e + y2 + 3.5 * x3n + 4.25 * x3e + y3
R1: (y0 + x1n + x1e - y1==2184)
R2: (y1 + x2n + x2e - y2==4945)
R3: (y2 + x3n + x3e - y3==2356)
GLPK Simplex Optimizer 5.0
3 rows, 10 columns, 12 non-zeros
      0: obj =   5.000000000e+02 inf =   8.985e+03 (3)
      6: obj =   3.436425000e+04 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
###>Z: 34364.250000

PyMathProg 1.0 Sensitivity Report Created: 2022/04/19 Tue 19:29PM
================================================================================
Variable            Activity   Dual.Value     Obj.Coef   Range.From   Range.Till
--------------------------------------------------------------------------------
 y0                      500       -3.25            1        -inf          inf
 x1n                    2500       -0.75          3.5        -inf         4.25
*x1e                     379           0         4.25         3.5 1.79769e+308
*y1                     1195           0            1           0 1.79769e+308
 x2n                    2500       -1.75          3.5        -inf         5.25
 x2e                    1250          -1         4.25        -inf         5.25
 y2                        0        2.75            1       -1.75          inf
```

GLPK -> **PyMathProg -> Hillier -> Fábrica de Lâmpadas -> Código->saída**

```
*x3n                2356         0        3.5         -1       4.25
 x3e                   0      0.75       4.25        3.5        inf
 y3                    0       4.5          1       -3.5        inf
==================================================================================
Note: rows marked with a * list a basic variable.

==================================================================================
Constraint      Activity Dual.Value  Lower.Bnd  Upper.Bnd RangeLower RangeUpper
-------------------------------------------------------------------------
 R1                2184      4.25       2184       2184       1805       3055
 R2                4945      5.25       4945       4945       4566       5816
 R3                2356       3.5       2356       2356          0       2500
==================================================================================
Note: normally, RangeLower is the min for the binding bound, and RangeUpper
gives the max value. However, when neither bounds are binding, the row is
marked with a *, and RangeLower is the max for Lower.Bnd(whose min is -inf),
and RangeUpper is the min for Upper.Bnd(whose max value is inf). Then the
columns of RangeLower, RangeUpper and Activity all have identical values.

__del__ is deleting problem: Fábrica de Lâmpadas.
```

Referências

[1]Cattrysse, D. "LINEAR-PROGRAMMING AND NETWORK FLOWS-BAZARAA, MS, JARVIS, JJ, SHERALI, HD." European Journal of Operational Research 50.1 (1991): 94-94