Preparação do trabalho (para fazer em casa)

- Para o trabalho é necessário instalar Julia: https://julialang.org,
- e instalar os pacotes necessários em Julia

```
> <caminho>/bin/julia
julia> import Pkg
julia> Pkg.add("JuMP")
julia> Pkg.add("GLPKMathProgInterface")
julia> Pkg.add("IJulia")
```

- Caso não já está instalado, também é necessário instalar o GNU Linear Programming Kit (GLPK)
  - Windows: Baixar e instalar o GNU Linear Programming Kit (GLPK) em http://gnuwin32.sourceforge.net/packages/glpk.htm.
  - Linux: `sudo apt-get install glpk`
- A documentação de Julia/JuMP está disponível em http://www.juliaopt.org/JuMP.jl/stable.

Exemplo

Para demonstrar a especificação de um problema nos formatas CPLEX lp e GNU mathprog, considere o exemplo de um importador de Whisky:

Um importador de Whisky tem as seguintes restrições de importação

- no máximo 2000 garrafas de *Johnny Ballantine* por 70 R$ cada uma,
- no máximo 2500 garrafas de *Old Gargantua* por 50 R$ cada uma,
- no máximo 1200 garrafas de *Misty Deluxe* por 40 R$ cada uma.

Dos Whiskies importados ele produz três misturas A, B, C, que ele vende por 68 R$, 57 R$ e 45 R$, respectivamente. As misturas são

- A: no mínimo 60% Johnny Ballantine, no máximo 20% Misty Deluxe,
- B: no mínimo 15% Johnny Ballantine, no máximo 60% Misty Deluxe,
- C: no máximo 50% Misty Deluxe.

Quais seriam as misturas ótimas, e quantas garrafas de cada mistura devem ser produzidas para maximizar o lucro? Formule como programa linear.

Observações:

- Use como variáveis o número de garrafas $x_{m,i}$ da marca m usadas na mistura i.
- Desconsidere a integralidade das garrafas.

Licença Creative Commons (Atribuição-NãoComercial-SemDerivações 4.0).

Especificação em Julia

A solução é

Julia Notebook

```
using JuMP
using GLPK
using Formatting

I=collect(1:3); V=collect(1:3);

m = Model(GLPK.Optimizer)
@variable(m, x[i in I, v in V] >= 0);

pv=[68, 57, 45];

ci=[70, 50, 40];

@variable(m, y[v in V]) ## total vendas
@variable(m, z[i in I]) ## total importação
@objective(m, Max, sum(pv[v]*y[v] for v in V)-sum(ci[i]*z[i] for i in
↪  I))

@constraint(m, [v in V], y[v]==sum(x[i,v] for i in I))
@constraint(m, [i in I], z[i]==sum(x[i,v] for v in V));

li=[2000,2500,1200];

@constraint(m, [i in I], z[i] <= li[i]);

@constraint(m,x[1,1] >= 0.6 *y[1]) ## no mínimo 60% Johnny
↪  Ballantine em A
@constraint(m, x[3,1] <= 0.2 *y[1]) ## no máximo 20% Misty Deluxe em
↪  A

@constraint(m, x[1,2] >= 0.15*y[2]) ## no mínimo 15% Johnny
↪  Ballantine em B
```


Licença Creative Commons (Atribuição-NãoComercial-SemDerivações 4.0).

```
@constraint(m, x[3,2] <= 0.6 *y[2]) ## no máximo 60% Misty Deluxe em
↪ B

@constraint(m, x[3,3] <= 0.6 *y[3]); ## no máximo 50% Misty Deluxe
↪ em C

println(m)
optimize!(m)

printfmt("O lucor máximo é {:.2f}.\n",objective_value(m))
printfmt("Importar {:.2f} garrafas de Johnny Ballantine, {:.2f}
↪ garrafas de Old Gargantua, e {:.2f} garrafas de Misty
↪ Deluxe.",value(z[1]),value(z[2]),value(z[3]) )
for v in V
    printfmt("Mistura {}: {:.2f} garrafas.\n","ABC"[v],value(y[v]))
    if value(y[v])>0
      printfmt("  {:.2f}% JB, {:.2f}% OG, {:.2f}% MD.\n","ABC"[v],
        ↪ value(x[1,v]) / value(y[v]), value(x[2,v]) / value(y[v]),
        ↪ value(x[3,v]) / value(y[v]))
    end
end
```


Licença Creative Commons (Atribuição-NãoComercial-SemDerivações 4.0 🅭🅯🄏🄍).

Laboratório 1

Questão 1 (Formulação Matemática)

Resolve o problema da Manufatura Peça Mil da primeira lista usando Julia/JuMP.

a) Formula o problema primeiramente explicitamente, com todos coeficientes.

b) Depois estuda uma forma mais compacta, usando vetores e matrizes para os dados, e somatórios. Começa por definir os dados. Por exemplo, usando índices $i = 1, 2, 3$ para fábricas e $j = 1, 2, 3, 4, 5$ para atacadistas podemos definir uma matriz c_{ij} de custos por:

```
julia> c=[[0.05 0.07 0.11 0.15 0.15]; [0.08 0.06 0.10
   0.12 0.15]; [0.10 0.09 0.09 0.10 0.16]]
3×5 Array{Float64,2}:
 0.05  0.07  0.11  0.15  0.15
 0.08  0.06  0.1   0.12  0.15
 0.1   0.09  0.09  0.1   0.16
```

Depois re-escreve a função objetivo e as restrições de forma compacta.

Questão 2 (Formulação Matemática)

Resolve o problem da Companhia Siderúrigica Jericó da primeira lista usando Julia/-JuMP.

Questão 3 (Formulação Matemática)

Resolve o problema de misturar o drinque ideal da primeira lista usando Julia/JuMP. (Como neste problema não tem dados, formula de forma genérica e depois gera alguns dados aleatórios. Podemos, por exemplo, definir um vetor de custos r para $n = 10$ ingredientes desta forma:

```
julia> r=100*rand(10)
10-element Array{Float64,1}:
  2.111057439604469
 62.08122447744915
 62.216802504853284
 39.31028824759333
 19.013908175781612
 27.74028421585395
 89.64090779389875
 91.84418467215028
```


Licença Creative Commons (Atribuição-NãoComercial-SemDerivações 4.0).

```
32.697177755765324
53.95516610718194
```

)

Questão 4 (Formulação Matemática)
Resolve o planejamento ótimo para construir uma casa da primeira lista usando Julia/-JuMP. Como poderia uma generalização para tarefas com dependências arbitrárias ser formulada?

Questão 5 (Formulação Matemática)
Resolve o problema da Polícia de Cidade Limpa da primeira lista usando Julia/JuMP.

Criando iPython notebooks com Julia A forma mais simples de trabalhar com a Julia é usar um Notebook. Para trabalhar com Notebooks vocês precisam alguma IDE que os apoia (e.g. vscode). Também é possível ter um servidor para notebooks na máquina local que roda no browser (e.g. em Ubuntu chamar `jupyter-notebook` na linha de comando; isso abre um navegador no browser).

Licença Creative Commons (Atribuição-NãoComercial-SemDerivações 4.0).