51º SBPO LIMEIRA 2019

UFG

Introdução prática à Linguagem Julia para Programação Matemática

Prof. Dr. Helio Fuchigami
Universidade Federal de Goias (UFG)
Campus Aparecida de Goiânia

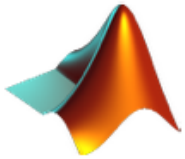
Limeira, 3 e 4 de setembro de 2019

Por que a linguagem Julia?

- ✓ Resultado de uma longa busca e testes
- ✓ Objetivos das buscas:
  - Linguagem de programação eficiente para implementação de heurísticas
  - Linguagem fácil, moderna e com IDE gratuito
  - Linguagem de modelagem para implementação de PLI com arquivos separados para modelo e dados (execução sequencial de vários exemplares)
  - Linguagem de modelagem fácil, gratuita e com o Cplex e outros resolvedores (também gratuitos)

Technical Computing Landscape

Por que o nome "Julia"?



	Matlab	Python	Julia
Introduced	1984	1991*	2012
Creators	MathWorks	Guido van Rossum	Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman
License	Proprietary	BSD-style	MIT
Name	"Matrix Laboratory"	Monty Python	?

*NumPy introduced in 1995 as Numeric and 2006 as NumPy

Linguagem Julia

http://julialang.org

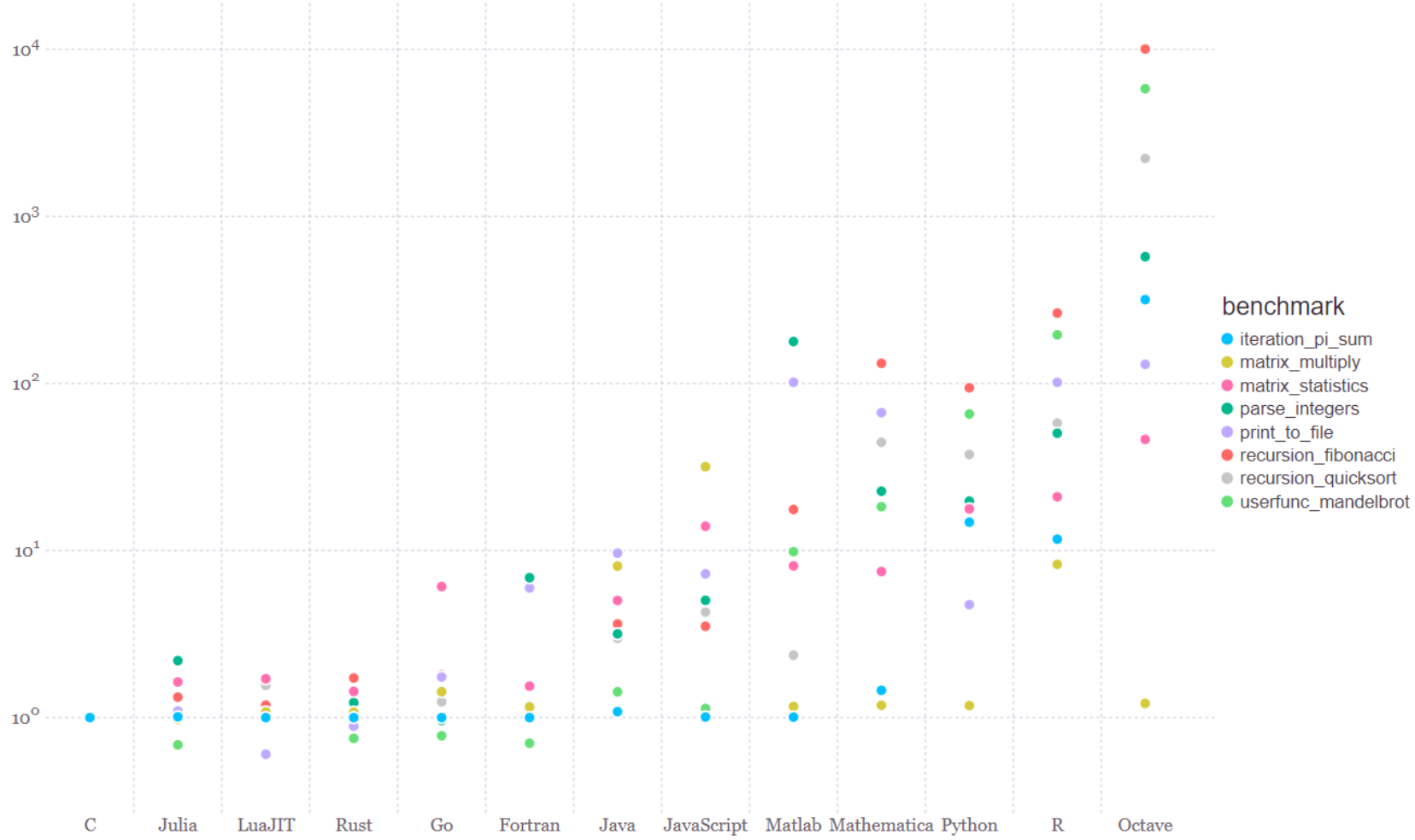
- ✓ Linguagem de programação de alto nível e alta performance para computação numérica e científica
- ✓ Código aberto, livre de licenças
- ✓ Desenvolvedores do MIT
- ✓ Extensa biblioteca de funções matemáticas
- ✓ Comunidade de desenvolvedores contribuem continuamente com pacotes (built-in package manager)

Linguagem Julia

http://julialang.org

- ✓ Fácil utilização e aprendizado
- ✓ Criada para agregar as vantagens de outras linguagens (Matlab, Python, Ruby, R, C/C++)
- ✓ "Walks like Python. Runs like C." (junolab.org)
- ✓ Linguagem jovem (de 2012) mas com rápida adesão (versão 1.0 em 2018 e há muitas publicações científicas e disciplinas acadêmicas)
- ✓ Conferência anual JuliaCon: juliacon.org

Compilador JIT de alta performance



JuMP: Linguagem de modelagem

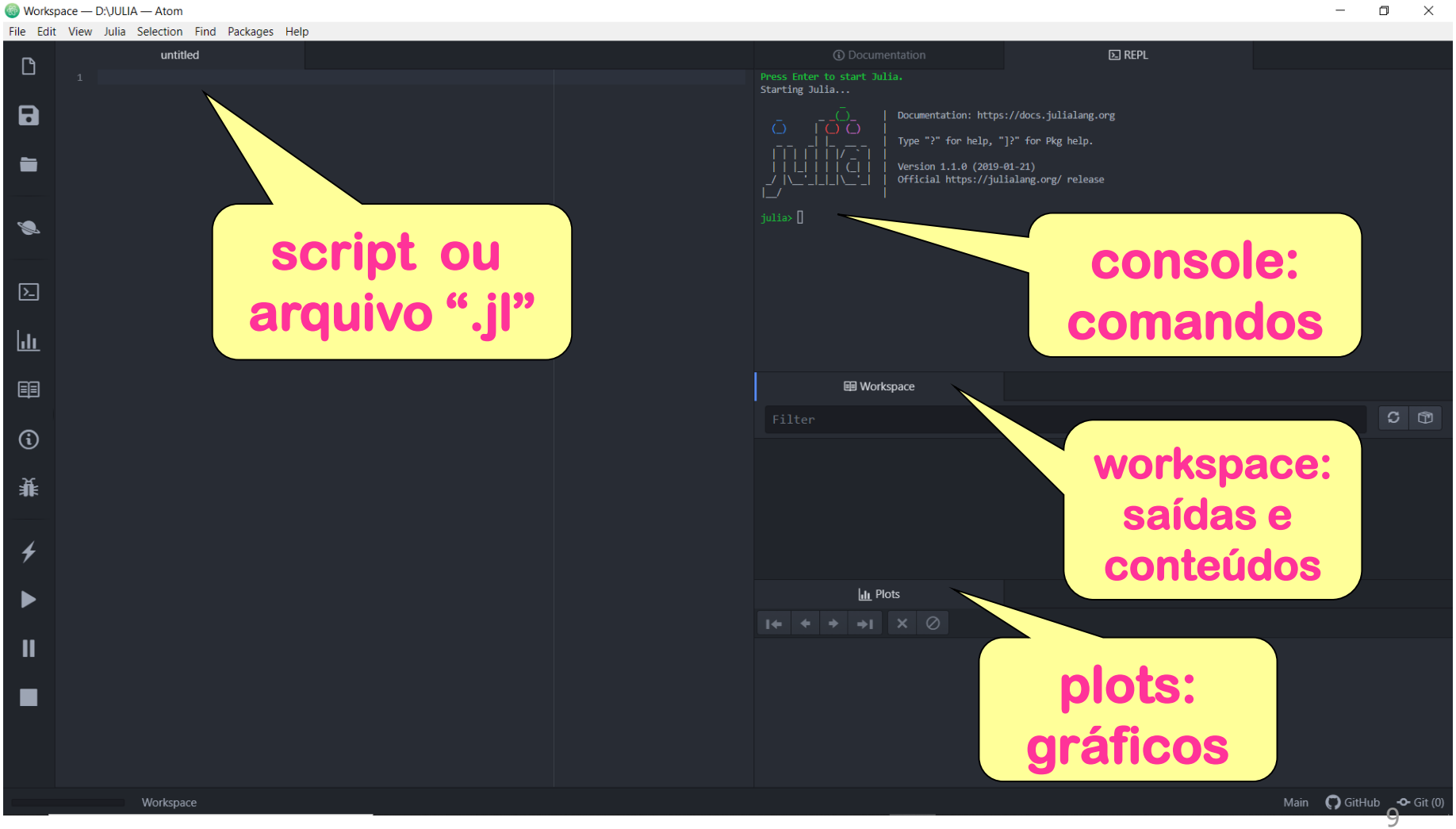
- ✓ Julia for Mathematical Programming
- ✓ Embutida nos desenvolvedores da linguagem
- ✓ Mais de 10 resolvedores (solvers) de Programação Linear, Linear Inteira Mista, Não Linear, Cônica e Semidefinida
  - CPLEX, Gurobi, CBC, GLPK
- ✓ Característica: implementação independente do resolvedor

Instalação: "JuliaPro Personal"

https://juliacomputing.com/products/juliapro.html

- ✓ Julia
  - Windows, Linux e MacOS
- ✓ JuMP e resolvedores
  - Pkg.add("JuMP"), Pkg.add("Cbc") etc.
- ✓ Ferramentas para desenvolvimento
  - JuliaPro (distribuição com compilador, debbuger, ambiente de desenvolvimento, pacotes e facilidades)
    - editor de texto "Atom" com plugin (funcionalidade) "Juno"
    - IDE = Integrated Development Environment

Interface Atom



Operadores básicos

- ✓ Aritméticos: **+**, **–**, ***, **/**

```
julia> 7 * 9 <enter>
julia> 50/4  <enter>
```

Resultado truncado: **div**

```
julia> div(50,4)  <enter>
```

- ✓ Resto da divisão inteira: **%**

```
julia> 50%4 <enter>
```

Operadores básicos

- ✓ Racional: //

```
julia> 2//3 + 2 <enter>
julia> 2/3+2  <enter>

julia> (2/3)+2  # melhor forma!
julia> rationalize(2.5) <enter>
```

- ✓ Divisão inversa: \

```
julia> 10\2 <enter>
```

Operadores básicos

- ✓ Exponencial: ^

```
julia> 3^2 <enter>
```

- ✓ Funções matemáticas: exp, sqrt

```
julia> exp(3) <enter>
julia> sqrt(49) <enter>
```

- ✓ Número π = perímetro/diâmetro: pi

```
julia> pi <enter>
```

Operadores básicos

- ✓ Funções trigonométricas: sin, cos, tan

```
julia> sin(pi/2) <enter>
julia> cos(pi) <enter>
julia> tan(pi/3) <enter>
```

- ✓ Números aleatórios: rand # U[0,1]

```
julia> rand() <enter>
julia> rand(5) <enter>
julia> rand(10:20) <enter>
```

Variáveis – tipos simples

- ✓ Inteiro: Int64
- ✓ Real: Float64
- ✓ Verdadeiro/Falso: Bool # true/false
- ✓ Caracteres: Char

- ✓ Não é necessário declarar o tipo da variável!
- ✓ Julia é case sensitive: a ≠ A

```
julia> x = 1 <enter>
julia> y = 2.0 <enter>
julia> z = true <enter>
julia> w = “texto” <enter>
```

Operadores

- ✓ Atribuição: **=**
- ✓ Lógicos: **==**, **!=**, **<**, **>**, **<=**, **>=**, **&&**, **||**

```
julia> x1 = 20; x2 = 35; x3 = 50
julia> x1 == x2
julia> x1 != x3
julia> x2 >= x3
julia> x1 <= 25 && x2 <= 30
julia> x1 <= 25 || x2 <= 30
```

Operadores

- ✓ Negação: !true ⇔ false

```
julia> y1 = false
julia> y2 = true

julia> !y1
julia> y1 == y2
julia> y1 == !y2
```

Vetores e matrizes: Array

✓ Precisam ser inicializados

- Declaração (inicializa com "lixo"):

```
julia> A = Array{Float64}(undef,3,4)
```

- Conteúdo zero:

```
julia> b = zeros(3)
```

- Valores iniciais:

```
julia> c = [5  9  7  1]
```

Vetores e matrizes: Array

- Vetor linha 1x3

```
julia> a = [6.1   2.3   5.4]
```

- Vetor coluna 3x1

```
julia> b = [6.1;  2.3;  5.4]
```

- Matriz 3x4

```
julia> A = [1  3  4  2; 2  8  5  4; 7  9  0  6]

julia> A = [1  3  4  2;
            2  8  5  4;
            7  9  0  6]
```

Vetores – índices e intervalos

- ✓ Indexa vetores a partir de [1] e não de [0]

```
julia> a
julia> a[2] =  9.7

julia> c = [30  20  60  40  10  90  70  80]
julia> c[2:4]  # elementos nas posições 2 a 4
julia> c[end]  # último elemento
julia> c[end-1]  # penúltimo elemento
julia> c[end-2:end]  # três últimos elementos
```

Operações com matrizes

```
julia> A
julia> A[3,:]  # terceira linha
julia> A[:,2:3]  # segunda e terceira colunas

julia> transpose(A) # transposta de A
julia> A’ # transposta de A

julia> zeros(4,5) # cria matriz de zeros
julia> ones(3,2) # cria matriz de 1
```

Funções pré-definidas

```
julia> round(3.75) # arrendonda
julia> trunc(3.75) # trunca
julia> min(x1, x2, x3) # menor valor
julia> max(x1, x2, x3) # maior valor
julia> isequal(x1, x2) # se é igual
julia> iseven(x2) # se é par
julia> isodd(x2) # se é ímpar
```

Funções pré-definidas

```
julia> c
julia> sum(c) # soma dos valores do vetor c
julia> minimum(c) # menor valor do vetor c
julia> maximum(c) # maior valor do vetor c
julia> sort(c, dims=2) # ordena o vetor c
# como é vetor linha, ordena colunas (dims=2)
# só ordena a saída na tela, verifique:
julia> c
```

Mais sobre Arrays

- ✓ Criando e preenchendo com valores

```
julia> A = zeros(7,3)

julia> B = ones(4,6)

julia> C = trues(2,4)

julia> D = falses(5,3)

julia> E = fill(NaN,4,3)

julia> F = fill(7,4,3)
```

Mais sobre Arrays

```
julia> G = rand(8,6,3)
julia> H = rand(10:20,6,3)
```

✓ Operador ellipsis (para vetores: 1 dimensão)

```
julia> a = [1:5...]
julia> b = [1:2:9...]
julia> c = [1.2:2.2:15.0...]
```

Propriedades dos Arrays

- ✓ Tamanho do vetor

```
julia> length(c)
```

- ✓ Dimensões da matriz

```
julia> size(A)
julia> (n,m) = size(A) # guarda valores
```

Arrays

- ✓ Encontrar itens no Array

```
julia> 12 in H
julia> in(15,H)
```

- ✓ Listando operações com elementos

```
julia> [b[i]^2 for i=1:length(b)]
julia> [n^2 for n in b]
```

Strings

- ✓ Char (caracter)

```
julia> a1 = 'a'
```

- ✓ String

```
julia> a2 = "a"

julia> typeof(a1)
julia> typeof(a2)

julia> a3 = 'blablabla' #verifique!
```

String

- ✓ Concatenação

```
julia> string("Tudo ", "junto")
julia> a4 = "Hello "
julia> a5 = "Hello "
julia> string(a4,a5) # 1ª forma
julia> a4*a5 # 2ª forma
julia> *(a4,a5) # 3ª forma
```

Interpolação

```
julia> total = 250.0
julia> "A soma é $total."

julia> d = rand(1:9,8)
julia> ["Número $i" for i in d]

# no script
for i in d
     println("Número $i")
end
```

Interpolação

```
julia> a6 = "Olá"
julia> a7 = "mundo"
julia> esp = " "
julia> "$a6$esp$a6“
```

E quando queremos imprimir $?

```
julia> saldo = "100"
julia> print("Tenho \$$saldo em conta.")
```

String como Array

```
julia> str = "Você foi avisado!"

julia> str[end]

julia> str[1:4]

julia> str[10:end]

julia> length(str)
```

Funções de String

```
julia> split(str)


julia> repeat("Oi ",3)
julia> "Oi "^3
julia> ^("Oi ",3)


julia> uppercase("Machado de Assis")
julia> lowercase("Machado de Assis")
```

Funções de String

```
julia> reverse("Bentinho")


# elimina espaços no início e fim
julia> lstrip(" Machado de Assis ")
julia> rstrip(" Machado de Assis ")
julia> strip(" Machado de Assis ")
```

Comparações lexicográficas

```
julia> "Lucas" < "Luan"

julia> "g" < "G"

julia> "13 April" < "13 May"

julia> "super" == "Super"
```

Funções

✓ Função hipotenusa: $h^2 = a^2 + b^2$

```
# no script salve como "hipotenusa.jl"
function hipotenusa(a,b)
      h = sqrt(a^2 + b^2)
      return h
end
println(hipotenusa(3,4))

# Run ou no console: include("hipotenusa.jl")
# no console:
julia> hipotenusa(3,4)
```

Funções

- ✓ **Função Fibonacci:** imprime o n-ésimo elemento da sequência de Fibonacci

- ✓ Sequência de números inteiros:
  $F_n = F_{n-1} + F_{n-2}$, com $F_0 = 0$, $F_1 = 1$

- ✓ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

✓ Função Fibonacci

```
# no script salve como “fibonacci.jl"
function fibo(n)
    if n < 2
        return n
    else
        return fibo(n-1) + fibo(n-2)
    end
end
# Run ou no console: include("fibonacci.jl")
# no console
julia> fibo(10)
julia> @time fibo(40)
```

Desafio: como imprimir os 20 primeiros termos de Fibonacci?

Gráficos básicos

- ✓ Adicione os pacotes no console:

```
julia> Pkg.add("Plots")
julia> Pkg.add("GR")

julia> using Plots
```

Gráficos básicos

✓ Exemplo 1

```
> plot(rand(5,5), linewidth=2.0, title="Aleatório")

> savefig("figura1.pdf") # só no plotpane

> plotly() # web backend
> gr() # plotpane backend
```

Gráficos básicos

- ✓ **Exemplo 2**

```
> x = range(0, stop=2*pi, length=1000);

> y = sin.(3*x+4*cos.(2*x));

> plot(x, y, color="red", linewidth=3.0,
linestyle=:solid)
```

- ✓ linestyle = :auto, :solid, :dash, :dot, :dashdot, :dashdotdot

Gráficos básicos

✓ **Exemplo 3**

```
> x = range(0, stop=2*pi, length=1000);

> y = sin.(3*x);

> plot(x, y, color="green", linewidth=2.0,
linestyle=:dash, title="Gráfico 1",
xlabel="valor de x", ylabel="seno 3x",
label="função 1")
```

Gráficos básicos

✓ **Exemplo 4 – Histograma**

```
> x = randn(1000);
> histogram(x)
```

Linguagem de modelagem Julia/JuMP

JuMP: Linguagem de modelagem

- ✓ Julia for Mathematical Programming
- ✓ Embutida nos desenvolvedores da linguagem
- ✓ Mais de 10 resolvedores (solvers) de Programação Linear, Linear Inteira Mista, Não Linear, Cônica e Semidefinida
  - CPLEX, Gurobi, CBC, GLPK
- ✓ Característica: implementação independente do resolvedor

Resolvedores (solvers)

http://jump.readthedocs.io/en/latest/installation.html#getting-solvers

- ✓ **Pkg.add("JuMP")**
- ✓ **Comerciais com licença acadêmica**
  - **Pkg.add("CPLEX")**
  - **Pkg.add("Gurobi")**
- ✓ **Open-source**
  - **Pkg.add("Clp")** # PL (instala também o "Cbc")
  - **Pkg.add("Cbc")** # PLIM
- ✓ **Outros para PLIM (testes):**
  - **"XpressSolver", "GLPKSolver[LP|MIP]", "MosekSolver", "OsilBonminSolver" etc.**

Estrutura básica

$$\max \; x_1 + x_2$$

$$\text{s. a} \; 2x_1 + x_2 \leq 8$$

$$0 \leq x_1 \leq +\infty$$

$$1 \leq x_2 \leq 20$$

```
using JuMP, Cbc

modelo = Model()

@variable(modelo, x1>=0)

@variable(modelo, 1<= x2 <=20)

@objective(modelo, Max, x1+x2)

@constraint(modelo, 2*x1+x2 <= 8)

optimize!(modelo)
```

optimize!(modelo, with_optimizer(Cbc.Optimizer))

Obtendo a solução

```
optimize!(modelo1, with_optimizer(Cbc.Optimizer))
```

- ✓ Ao resolver, **optimize!(modelo)** retorna:
  - **:Optimal** # solução ótima
  - **:Unbounded** # função objetivo ótima é infinita
  - **:Infeasible** # não há solução viável
- ✓ Valor da função objetivo: **objective_value(modelo)**
- ✓ Valor das variáveis: **value(x)**
- ✓ Imprime o modelo legível: **print(modelo)**

Programação Linear (exemplo1.jl)

```
using JuMP, Cbc

modelo = Model(with_optimizer(Cbc.Optimizer))

@variables modelo begin
    0<= x1 <= 10
    x2 >= 0
    x3 >= 0
end

@objective(modelo, Max, x1 + 2x2 + 5x3)

@constraints modelo begin
    -x1+x2+3x3 <= –5
    x1+3x2 –7x3<=10
end

print(modelo)
optimize!(modelo)

println("Função objetivo: ", objective_value(modelo))
println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
```

$$\max \; x_1 + 2x_2 + 5x_3$$

$$\text{s. a} \; -x_1 + x_2 + 3x_3 \leq -5$$
$$x_1 + 3x_2 - 7x_3 \leq 10$$
$$0 \leq x_1 \leq 10$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$

Programação Linear (exemplo2.jl)

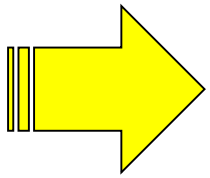
```
using JuMP, CPLEX

function  modelo1()
    modelo = Model(with_optimizer(CPLEX.Optimizer))

    @variable(modelo, 0<= x1 <= 10)
    @variable(modelo, x2 >= 0)
    @variable(modelo, x3 >= 0)

    @objective(modelo, Max, x1 + 2x2 + 5x3)

    @constraint(modelo, –x1+x2+3*x3 <= –5)
    @constraint(modelo, x1+3*x2 –7*x3<=10)

    print(modelo)
    optimize!(modelo)

    println(“Função objetivo: ”, objective_value(modelo))
    println(“x1 = ”, value(x1))
    println(“x2 = ”, value(x2))
    println(“x3 = ”, value(x3))
end

@time modelo1()
```

Exibe o tempo de execução (CPU) de resolução do modelo

PL – Notação matricial

$$\max \; x_1 + 2x_2 + 5x_3$$

$$\text{s. a} \; -x_1 + x_2 + 3x_3 \leq -5$$
$$x_1 + 3x_2 - 7x_3 \leq 10$$
$$0 \leq x_1 \leq 10$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$



$$\max \; cx$$

$$\text{s. a} \; Ax \leq b$$
$$x \geq 0$$

$$A = \begin{bmatrix} -1 & 1 & 3 \\ 1 & 3 & -7 \end{bmatrix}$$

$$b = [-5;\; 10]$$

$$c = [1 \;\; 2 \;\; 5]$$

$$x_1 \leq 10$$

m = 2 linhas (restrições)

n = 3 colunas (variáveis)

PL – Notação matricial (exemplo3.jl)

```
using JuMP, CPLEX

A = [ –1   1  3; 1   3 –7]
b = [–5;  10]
c = [ 1  2  5]
(m, n) = size(A)

modelo = Model(with_optimizer(CPLEX.Optimizer))

@variable(modelo, x[1:n] >= 0)

@objective(modelo, Max, sum(c[j]*x[j] for j=1:n))

@constraints modelo begin
     c1[i=1:m], sum(A[i,j]*x[j] for j=1:n) <= b[i]
     c2, x[1] <= 10
end

print(modelo)
optimize!(modelo)

println("Função objetivo: ", objective_value(modelo))
for j = 1:n
     println("x[$j] = ", value(x[j]))
end
```

Alternativa para índices:

```
indx  = 1:m
indrestr = 1:n

for i in indx
…

for j in indrestr
…
```

Programação Linear Inteira Mista

(exemplo4.jl)

$$\max \; x_1 + 2x_2 + 5x_3$$

$$\text{s. a} \; -x_1 + x_2 + 3x_3 \leq -5$$
$$x_1 + 3x_2 - 7x_3 \leq 10$$
$$0 \leq x_1 \leq 10$$
$$x_2 \geq 0 \text{ e inteiro}$$
$$x_3 \in \{0, 1\}$$

```
using JuMP, CPLEX
modelo = Model(with_optimizer(CPLEX.Optimizer))

@variables modelo begin
     0<= x1 <= 10
    x2 >= 0, Int
    x3, Bin
end
```

Atividade

Uma empresa siderúrgica possui 3 usinas e cada uma delas requer uma quantidade mensal mínima de minério para operar. A empresa adquire minério de 4 minas diferentes. Cada uma das minas tem uma capacidade máxima de produção mensal estabelecida. Por imposições contratuais, o custo do minério para a empresa é composto por um custo fixo mensal para cada mina (este valor é pago em caso de haver produção na mina), mais um custo de transporte ($/t) que varia de acordo com a distância entre as minas e usinas (cada par mina/usina tem um custo diferente). Construa e implemente um modelo de otimização para determinar a quantidade de minério a ser comprada de cada mina e levada a cada usina de forma a minimizar o custo total de compra do minério.

Usinas / Minas	Usina 1	Usina 2	Usina 3	Cap. máx. (t/mês)	Custo fixo ($)
Mina 1 ($/t)	10	8	13	11.500	50.000
Mina 2 ($/t)	7	9	14	14.500	40.000
Mina 3 ($/t)	6,5	10,8	12,4	13.000	30.000
Mina 4 ($/t)	8,5	12,7	9,8	12.300	25.500
Qtd. req. (t/mês)	10.000	15.400	13.300	-	-

Atividade

Parâmetros e índices:

m: número de minas

n: número de usinas

i: minas, i=1, ..., m

j: usinas, j=1, ..., n

f_i: custo fixo da mina i

c_{ij}: custo de transporte da mina i para a usina j

Cap_i: capacidade máxima de produção da mina i

d_j: demanda da usina j

Variáveis de decisão:

x_{ij}: quantidade de minério a ser transportado da mina i para a usina j

y_i = 1, se a mina i for usada, e 0, caso contrário

Atividade

✓

$$Min\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i$$

$$sujeito\ a \sum_{j=1}^{n} x_{ij} \leq Cap_i, i = 1, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} = d_j, j = 1, \dots, n$$

$$y_i \geq \frac{\sum_{j=1}^{n} x_{ij}}{Cap_i}, i = 1, \dots, m$$

$$x_{i,j} \geq 0, y_i \in \{0,1\}, i = 1, \dots, m, j = 1, \dots, n$$

Referências

KWON, C. **Julia Programming for Operational Research**: a primer on computing. San Bernardino, 2019. 2ª ed

BALBAERT, I. **Getting started with Julia Programming Language**. Packt Publishing, 2015.

BALBAERT, I.; SENGUPTA, A.; SHERRINGTON, M. **Julia: High Performance Programming** – Learning Path. Packt Publishing, 2016.

BEZANSON, J.; KARPINSKI, S.; SHAH, V.; EDELMAN, A. et al. **Julia Language Documentation**. Release 0.6.0-dev. 2016.

CASTELLUCCI, P.B. Julia e JuMP: novas ferramentas para programação matemática. **Pesquisa Operacional para o Desenvolvimento**, v. 9, n. 2, p. 48-61, 2017.

Referências (cont.)

DIFFERENCES <https://docs.julialang.org/en/stable/manual/noteworthy-differences/.> Acesso em 08 Jan 2018.

JULIA LANGUAGE. <http://julialang.org>. Acesso em 03 Jan 2018.

JULIA MANUAL. <http://docs.julialang.org/en/stable/>. Acesso em 08 Jan 2018.

KAMINSKI, B. **The Julia Express**. Tutorial. 2017.

LEARN X IN Y MINUTES <https://learnxinyminutes.com/docs/julia/>. Acesso em 08 Jan 2018.

PEREIRA, J.M. Introdução à linguagem de programação Julia: A ambiciosa linguagem de programação que quer substituir Python, R e Matlab. **III Escola Regional de Informática do Piauí** (anais), v. 1, n. 1, p. 315-335, 2017.

Referências (cont.)

PEREIRA, J.M; SIQUEIRA, M.B.B. Linguagem de programação Julia: uma alternativa open source e de alto desempenho ao Matlab. **Principia**, n. 34, p. 132-140, 2017.

SHERRINGTON, M. **Mastering Julia**. Packt Publishing, 2015.

SNELL, J. **Julia for Machine Learning**. Machine Learning Group Tutorial, 2014.

SOLVERS <http://jump.readthedocs.io/en/latest/installation.html#getting-solvers> Acesso em 14 Jan 2018.

SOUZA, M.J.F. **Otimização combinatória**. Departamento de Computação, Universidade Federal de Ouro Preto. Notas de aula, 2009.

OR-Library <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>