Introdução à linguagem Julia aplicada à otimização

Leonardo D. Secchin (UFES)

leonardosecchin.github.io

Escola de Verão 2025 - PPGMAT/UFPI

Objetivos do minicurso:

- Apresentar o funcionamento básico da linguagem Julia
- Discutir como Julia pode ser útil na modelagem e resolução de problemas de otimização
- Apresentar os principais pacotes mantidos pela comunidade cientifica voltados à otimização
- Não é um curso de otimização!
  - Um curso de otimização com Julia em português no Youtube: vídeos de Abel Siqueira

Pré-requisitos:

- Modelos de otimização
- Métodos básicos (gradientes conjugados, método do gradiente, Newton)

Acesso aos slides e códigos:

github.com/leonardosecchin/verao_UFPI_Julia

Introdução

Existem várias linguagens de programação...

Por que escolher Julia?

Algumas linguagens para computação científica

C/C++/Fortran

- Pró: muito rápido se bem implementado (todo o código é compilado)
- Pró: possui ótimos compiladores livres

- Pró: versatilidade (liberdade total para o programador)
- Pró (Fortran): notação matricial
- Contra: +difícil, tempo longo para aprender a desenvolver bons códigos
- Contra: tarefas simples usualmente resultam em códigos longos
- Contra: não há gerenciamento de memória (risco de falhas de execução)
- Contra: integrar códigos de terceiros é difícil
- Contra: o lado ruim da versatilidade: toda tarefa fica para o programador $\Rightarrow$ aumento do risco de falhas e códigos mal implementados

Python

- Pró: linguagem popular, muito código pronto disponível
- Pró: é livre
- Pró: aprendizagem fácil
- Pró: possui boas bibliotecas de alto desempenho (p.ex. para rotinas de álgebra linear)
- Pró: linguagem de propósito geral
- Contra: *loops*/laços lentos, dado que Python não compila código (linguagem interpretada)
- Contra: não há tantas bibliotecas para otimização

Matlab

- Pró: fácil de programar e aprender
- Pró: possui boas bibliotecas para determinados nichos (p.ex. *Simulink* para sistemas dinâmicos)
- Contra: é pago, incluindo bibliotecas adicionais (e é bem caro!)
- Contra: código puro Matlab é lento, especialmente *loops*
- Contra: não há muitas bibliotecas para otimização (muito menos que Julia)

Julia

Desenvolvida para computação científica de alto desempenho. Criada no MIT, primeira versão publicada em 2012.

Por que usar Julia?

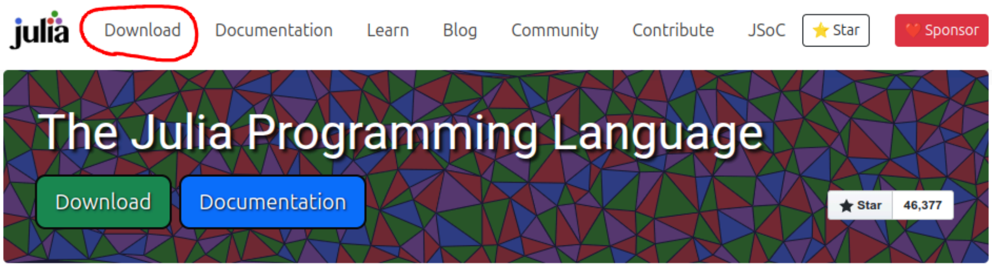
- É *software* livre. Assim, não requer licenças para uso
- É uma linguagem de alto nível, ou seja, é fácil programar e aprender (similar à Matlab)
- Por ser livre, a comunidade científica mantêm uma quantidade grande de códigos prontos (pacotes), que podem ser usados livremente. Isso traz enorme produtividade e economia de tempo
- Possui gerenciador de pacotes que torna a instalação e utilização de pacotes fácil

- Diferentemente do Matlab e Python, Julia compila o código. Isso traz eficiência comparada à linguagens de baixo nível como C/C++/Fortran
- Ao mesmo tempo que é fácil programar, tem foco no desempenho: os pacotes que rodam "por baixo" são implementados utilizando as melhores práticas/técnicas disponíveis (p.ex. rotinas de álgebra linear)
- **Tempo de aprendizado curto + produtividade + performance**
- Execução de códigos de outras linguagens (C/C++/Fortran) é fácil
- Bom para paralelismo

Ambiente Julia

Instalação no Windows

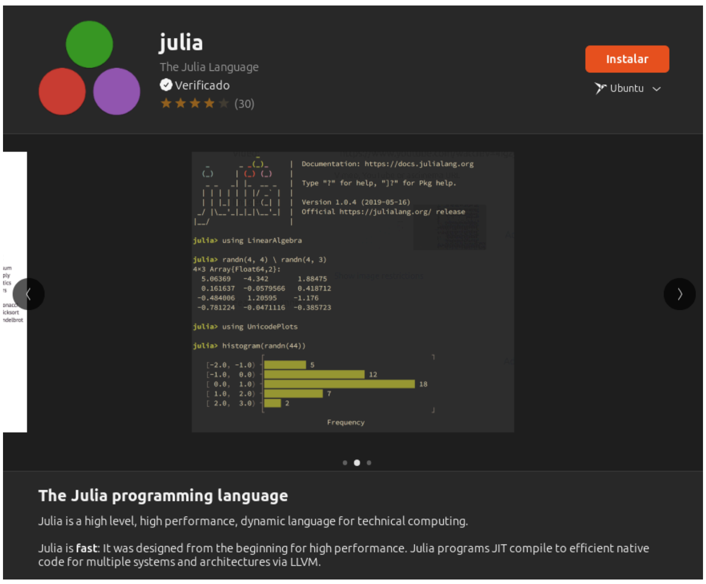
https://julialang.org



É recomendável baixar a última versão estável (*stable release*) 64 bits.

Instalação no GNU/Linux

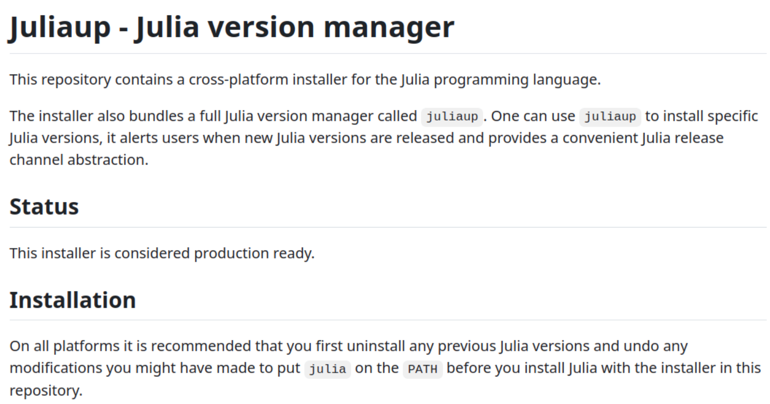
Opção 1: loja de aplicativos do sistema operacional (p.ex. Ubuntu)



Opção 2: script `juliaup`

Preferível, pois fornece melhor controle de versões. Passos:

1. Entre em https://github.com/JuliaLang/juliaup
2. Siga as instruções contidas no site



Juliaup - Julia version manager

This repository contains a cross-platform installer for the Julia programming language.

The installer also bundles a full Julia version manager called `juliaup`. One can use `juliaup` to install specific Julia versions, it alerts users when new Julia versions are released and provides a convenient Julia release channel abstraction.

Status

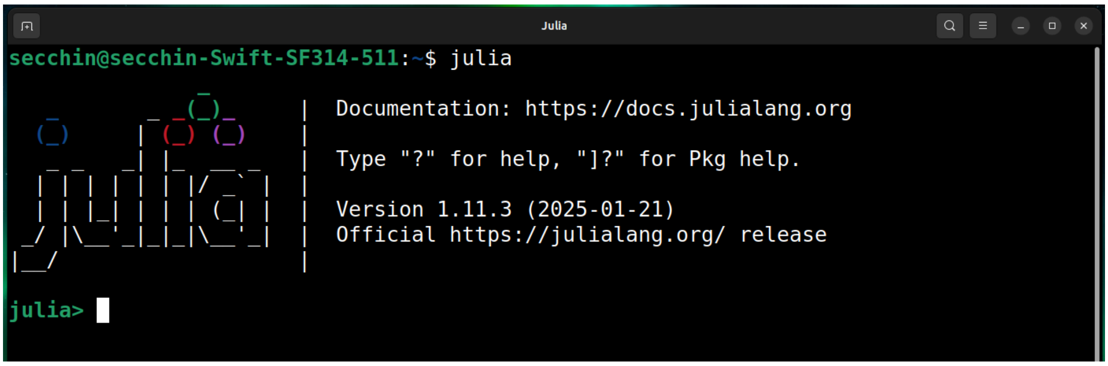
This installer is considered production ready.

Installation

On all platforms it is recommended that you first uninstall any previous Julia versions and undo any modifications you might have made to put `julia` on the `PATH` before you install Julia with the installer in this repository.

É recomendável instalar a última versão estável (*stable release*) 64 bits.

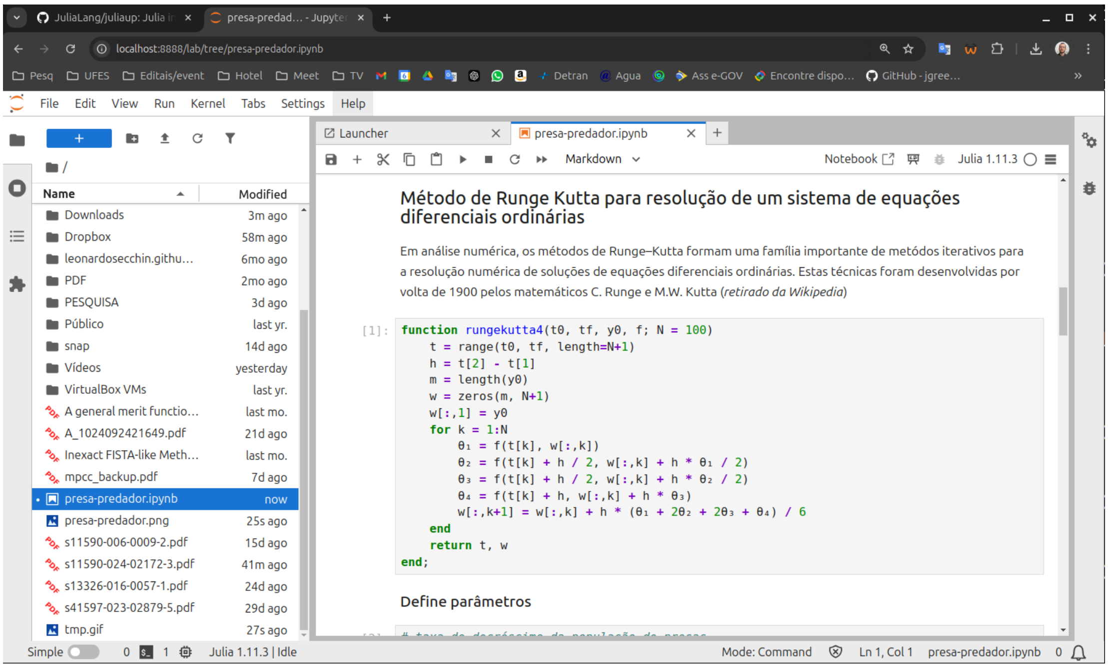
O ambiente Julia é executado em terminal de comandos (Powershell no Windows, terminal no GNU/Linux). Ao entrar, Julia está apto a receber comandos.

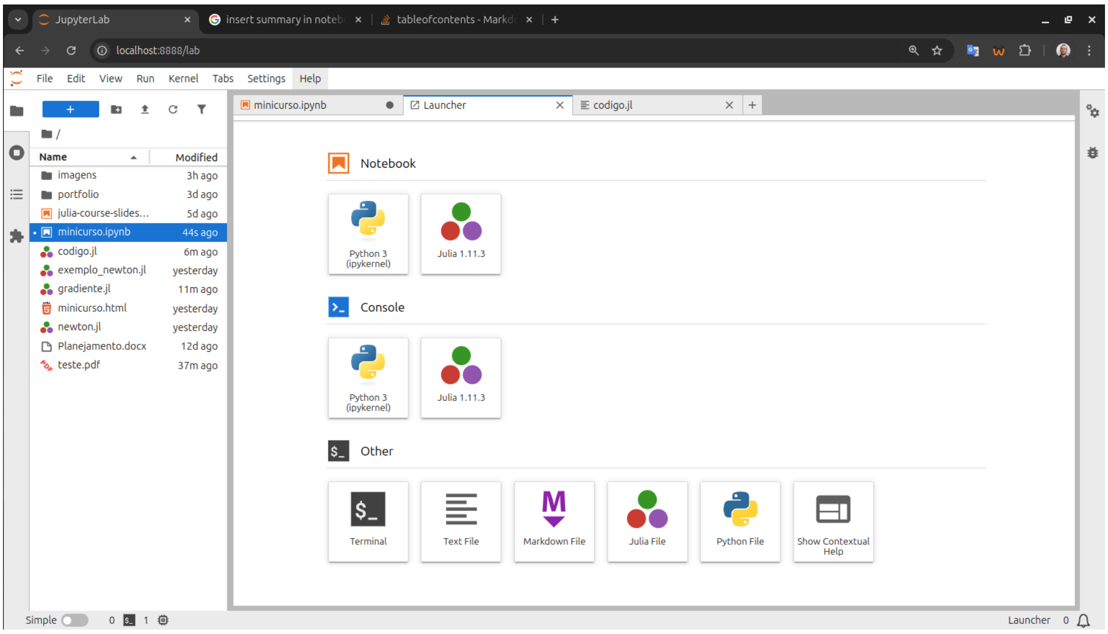


No GNU/Linux, execute "julia" a partir do terminal de comandos.

Modo gráfico com o Jupyter-lab

- Jupyter-lab é um ambiente de desenvolvimento simples, que funciona no navegador de internet.
- Pode ser executado a partir do sistema ou do Julia (veremos como).
- Trabalha com terminal e *notebooks*, extensão `.ipynb`, onde pode-se inserir textos, trechos de códigos, slides etc, tudo no mesmo documento.
- Os códigos Julia podem ser executados diretos no Jupyter-lab, onde as saídas do Julia são impressas.
- Oferece uma solução completa: é possível abrir terminais de comando, arquivos de códigos Julia, além de textos e slides.





Visual Studio Code

Ambiente completo de desenvolvimento da Microsoft, de uso livre. Possui diversas funcionalidades, porém é mais complexo.

- https://code.visualstudio.com/

Primeiros exemplos

Definindo funções

In [1]:
```
# o caracter # indica comentário, que não são executados!

# função simples escrita de maneira curta
f(x) = x^2
```

Out[1]: f (generic function with 1 method)

In [2]:
```
# especificando cabeçalho
function f(x)
    return x^2  # retorno da função
end
```

Out[2]: f (generic function with 1 method)

In [3]:
```
function h(A,b)
    m,n = size(A)
    println("A tem ordem $(m) x $(n)")
    if n != length(b)
        println("Dimensões de A e b incompatíveis!")
        return
    end
    Ab = A*b
    return Ab
end
```

Out[3]: h (generic function with 1 method)

In [4]:
```
f(3)
```

Out[4]: 9

In [5]:
```
A = rand(5,3)
b = rand(3)
h(A,b)
```

```
A tem ordem 5 x 3
```

Out[5]:
```
5-element Vector{Float64}:
 0.8140387719807002
 1.9598412364340252
 1.627675142950006
 1.005285938729986
 0.9692002998676558
```

Variáveis

Existem vários tipos de variáveis: números "reais" (Real, Float64, Float32 ...), números inteiros (Int, Int64, Int32 ...), vetores, matrizes, textos, boleano (true/false)...

Ao definir uma variável sem especificar o tipo, o Julia identifica o melhor tipo automaticamente.

In [6]:
```
a = 1
b = 1.5;
```

In [7]:
```
typeof(a)
```

Out[7]: Int64

In [8]:
```
typeof(b)
```

Out[8]: Float64

In [9]:
```
v = [1; 3; 6]
```

Out[9]:
```
3-element Vector{Int64}:
 1
 3
 6
```

In [10]:
```
u = [1; 3; 6.0]
```

Out[10]:
```
3-element Vector{Float64}:
 1.0
 3.0
 6.0
```

Às vezes é interessante forçar um tipo para uma variável passada para uma função de modo a impedir o uso com argumentos inválidos.

In [11]:
```
function resto(a::Int, b::Int)   # a e b devem ser inteiros
    r = mod(a,b)
    return r
end
```

Out[11]: resto (generic function with 1 method)

In [12]:
```
resto(10,3)
```

Out[12]: 1

In [13]:
```
resto(10.0,3)
```

```
MethodError: no method matching resto(::Float64, ::Int64)
The function `resto` exists, but no method is defined for this combination
of argument types.

Closest candidates are:
  resto(::Int64, ::Int64)
   @ Main In[11]:1


Stacktrace:
 [1] top-level scope
   @ In[13]:1
```

Uma das características importantes do Julia é o **múltiplo despacho**: podemos definir uma mesma função para diferentes tipos de cabeçalho. Isso é interessante por questões de eficiência e organização do código.

In [14]:
```
# função resto com parâmetros reais
function resto(a::Real, b::Real)
    println("Os dados de entrada são números reais.")
    r = mod(floor(a),floor(b))    # floor(x) é o maior inteiro menor ou i
```

```
    return r
end
```

Out[14]: resto (generic function with 2 methods)

In [15]:
```
# ambos parâmetros são inteiros, Julia executa a primeira versão
resto(10,3)
```

Out[15]: 1

In [16]:
```
# Um dos parâmetros é não inteiro, Julia executa a segunda versão.
# O segundo parâmetro 3 é convertido para Real.
resto(10.0,3)
```

```
Os dados de entrada são números reais.
```

Out[16]: 1.0

In [17]:
```
# Ambos parâmetros são não inteiros
resto(10.5,3.56)
```

```
Os dados de entrada são números reais.
```

Out[17]: 1.0

Se ... então

Segue a mesma lógica de outras linguagens. Sintaxe:

```
if [condição]
    ...
else
    ...
end
```

In [18]:
```
x = 5
if x > 4
    println("x é grande...")
else
    println("x é pequeno")
end
```

```
x é grande...
```

Observação

`=` é usado para **atribuir** valores à variáveis. Para **comparar** o valor de duas variáveis, use `==`

In [19]:
```
if x == 5
    println("x é igual a 5")
end
```

```
x é igual a 5
```

Laços

Seguem a lógica de várias outras linguagens.

In [20]:
```
# for: laço de tamanho pré-determinado
for i in 1:5
    print(i, " ")
end
```

```
1 2 3 4 5
```

In [21]:
```
# while: laço de tamanho indeterminado
k = rand(3:6) # sorteia um número inteiro entre 3 e 6
while (k > 0)
    print(k, " ")
    k = k - 1
end
```

```
5 4 3 2 1
```

Laços podem ser finalizados com `break`. Interessante para parar um `while` ao verificar uma condição de parada de um método.

In [22]:
```
function parada(x)
    pare = false
    if abs(x) < 1e-4    # o mesmo que abs(x) < 0.0001
        pare = true
    end
    return pare
end

maxiter = 5; k = 0; x = 1.0
while (true)
    println("Iteração $(k), x = $(x)")
    if (k >= maxiter)
        println("Número máximo de iterações atingido.")
        break
    end

    x /= 10   # o mesmo que x = x/10
    if parada(x)
        println("Critério de parada atingido, x = $(x)")
        break
    end
    k += 1   # o mesmo que k = k + 1
end
```

```
Iteração 0, x = 1.0
Iteração 1, x = 0.1
Iteração 2, x = 0.01
Iteração 3, x = 0.001
Iteração 4, x = 0.0001
Critério de parada atingido, x = 1.0e-5
```

Vetores e matrizes

Em Julia, vetores são sempre vetores-colunas.

`;` separa **linhas**

espaço separa **colunas**

In [23]:
```julia
# Definindo um vetor
v = [1;2;3;4;5]
```

Out[23]:
```
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

In [24]:
```julia
# Definindo uma matriz
A = [1.0  2.5  3.9;  4.0  5.9  6.7;  7.0  8.0  9.1]
```

Out[24]:
```
3×3 Matrix{Float64}:
 1.0  2.5  3.9
 4.0  5.9  6.7
 7.0  8.0  9.1
```

Alguns comandos básicos:

In [25]:
```julia
n = 3; m = 2;
# sorteia um vetor de tamanho n com entradas entre 0 e 1
w = rand(n)
```

Out[25]:
```
3-element Vector{Float64}:
 0.3101348786753917
 0.8525252618656493
 0.2451734489152705
```

In [26]:
```julia
# sorteia uma matriz m x n com entradas entre 0 e 1
M = rand(m,n)
```

Out[26]:
```
2×3 Matrix{Float64}:
 0.878734  0.893365  0.605282
 0.576111  0.501236  0.125074
```

Operações

In [27]:
```julia
A = rand(3,5);
B = rand(5,4);
C = rand(3,5);
a = rand(5);
b = rand(5);
```

In [28]:
```julia
# Adicão/multiplicação por escalar
S = 3*A - 1e-2*C
```

Out[28]:
```
3×5 Matrix{Float64}:
 1.35445  0.314963  2.10284   0.804091  1.26053
 1.30618  0.886344  0.150679  0.463586  0.0335426
 2.51708  1.75457   0.877765  0.470525  0.231764
```

In [29]:
```julia
C = A*b
```

Out[29]:
```
3-element Vector{Float64}:
 1.2319076729925185
 0.5083583261705475
 1.0491900419534557
```

In [30]:
```
C = A*B
```

Out[30]:
```
3×4 Matrix{Float64}:
 1.41296   0.825009  0.607541  0.928643
 0.598537  0.240546  0.381758  0.291714
 1.31629   0.61092   0.745596  0.655538
```

In [31]:
```
# Transposição
At = A'
```

Out[31]:
```
5×3 adjoint(::Matrix{Float64}) with eltype Float64:
 0.454411  0.438014   0.839693
 0.105476  0.297132   0.586087
 0.703656  0.0533054  0.293077
 0.270943  0.155317   0.157164
 0.421712  0.0119948  0.077817
```

Matrizes especiais

In [32]:
```
A = [1 2 3; 4 5 6; 7 8 9]
```

Out[32]:
```
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
```

In [33]:
```
using LinearAlgebra
# Matriz simétrica
Symmetric(A)
```

Out[33]:
```
3×3 Symmetric{Int64, Matrix{Int64}}:
 1  2  3
 2  5  6
 3  6  9
```

In [34]:
```
# Usando o triângulo inferior
Symmetric(A, :L)
```

Out[34]:
```
3×3 Symmetric{Int64, Matrix{Int64}}:
 1  4  7
 4  5  8
 7  8  9
```

In [35]:
```
# Matriz identidade de ordem n
using LinearAlgebra
n = 3
I(n)
```

Out[35]:
```
3×3 Diagonal{Bool, Vector{Bool}}:
 1  ⋅  ⋅
 ⋅  1  ⋅
 ⋅  ⋅  1
```

In [36]:
```
# Matriz diagonal
```

```
D = Diagonal([1;4;6])
```

Out[36]:
```
3×3 Diagonal{Int64, Vector{Int64}}:
 1  ⋅  ⋅
 ⋅  4  ⋅
 ⋅  ⋅  6
```

In [37]:
```
# Triangular inferior
tril(A)
```

Out[37]:
```
3×3 Matrix{Int64}:
 1  0  0
 4  5  0
 7  8  9
```

In [38]:
```
# Triangular superior
triu(A)
```

Out[38]:
```
3×3 Matrix{Int64}:
 1  2  3
 0  5  6
 0  0  9
```

In [39]:
```
# Vetor/Matriz de 1's
w = ones(10);
W = ones(10,4);
```

In [40]:
```
# Vetor/Matriz de zeros
z = zeros(10);
Z = zeros(10,4);
```

Atenção!

Supõe que queiramos fazer uma cópia de um vetor `a` em um vetor `b'...

In [41]:
```
a = rand(2)
```

Out[41]:
```
2-element Vector{Float64}:
 0.7471662012847251
 0.529573138383799
```

In [42]:
```
b = a
```

Out[42]:
```
2-element Vector{Float64}:
 0.7471662012847251
 0.529573138383799
```

Aparentemente, `b` é uma cópia de `a`. Porém, o trecho de código acima não copia `a` na memória, apenas faz uma referência à `a`. Assim, se alterarmos `b`, alteraremos `a`:

In [43]:
```
b[1] = 0
a
```

Out[43]:
```
2-element Vector{Float64}:
 0.0
 0.529573138383799
```

Uma cópia na memória pode ser feita:

In [44]:
```
a = rand(2)
b = deepcopy(a)   # aloca uma nova cópia de "a" memória
```

Out[44]:
```
2-element Vector{Float64}:
 0.599432714687276
 0.16206363053273476
```

In [45]:
```
# alterando b
b[1] = 0
b
```

Out[45]:
```
2-element Vector{Float64}:
 0.0
 0.16206363053273476
```

In [46]:
```
# a não é alterado
a
```

Out[46]:
```
2-element Vector{Float64}:
 0.599432714687276
 0.16206363053273476
```

O Julia evita fazer cópias na memória por uma questão de eficiência. Assim, o programador deve decidir conscientemente alocar nova memória. Evite fazer cópias quando desnecessário, sobretudo dentro de laços, pois isso deixará seu código mais lento.

Alocando vetores

In [47]:
```
n = 2
# alocando um vetor e preenchendo com zeros
v = zeros(n)
```

Out[47]:
```
2-element Vector{Float64}:
 0.0
 0.0
```

In [48]:
```
# alocando um vetor com mesma estrutura de v,
# mas sem preencher (só reserva o espaço de memória)
u = similar(v)
```

Out[48]:
```
2-element Vector{Float64}:
 6.4306804040713e-310
 2.0e-323
```

O comando `similar` reserva um espaço de memória, mas não grava as coordenadas (os valores que aparecem acima significam nada).

No trecho acima, `u` tem o mesmo tamanho e tipo numérico que `v`. Isto é, `u` é um vetor de $\mathbb{R}^4$ com dados do tipo `Float64` (numero real).

Ao não gravar as coordenadas de `u`, economizamos tempo. Isso é útil quando apenas necessitamos reservar um espaço de memória para só depois gravar as coordenadas. Muito útil em implementações de métodos de otimização, como vamos utilizar mais a frente.

ATENÇÃO: Se executarmos `u = v`, isso substituirá o vetor alocado pela referência à `v`, como antes.

Para copiar `v` sobre `u`, precisamos gravar **coordenada a coordenada**.

In [49]:
```
# primeira forma: fazer um laço que corre todas as coordenadas
for i in 1:length(u)
    u[i] = v[i]
end
```

In [50]:
```
# segunda forma: notação .= (preferível)
u .= v
```

Out[50]:
```
2-element Vector{Float64}:
 0.0
 0.0
```

Indexação

A manipulação de matrizes e vetores no Julia é muito rica.

Exemplos:

In [51]:
```
v = [10;20;30;40;-50];
```

In [52]:
```
v[4]   # coordenada 4 de v
```

Out[52]:
```
40
```

In [53]:
```
v[3:5]     # vetor coordenadas de 3 a 5
```

Out[53]:
```
3-element Vector{Int64}:
  30
  40
 -50
```

In [54]:
```
coords = [1;3;5]   # coordenadas 1, 4 e 6
v[coords]
```

Out[54]:
```
3-element Vector{Int64}:
  10
  30
 -50
```

In [55]:
```
v[v .> 30]    # coordenadas maiores que 30
```

Out[55]:
```
1-element Vector{Int64}:
 40
```

In [56]:
```
v[(v .> 30) .| (v .< 0)]   # coordenadas maiores que 30 ou menores que 0
```

Out[56]:
```
2-element Vector{Int64}:
  40
 -50
```

In [57]:
```
# coordenadas indicadas por vetores de V ou F
C = [true;false;false;true;true]
v[C]
```

Out[57]:
```
3-element Vector{Int64}:
  10
  40
 -50
```

In [58]:
```
# índices correspondentes à negação de C
v[.!C]
```

Out[58]:
```
2-element Vector{Int64}:
 20
 30
```

Podemos atribuir valor a qualquer vetor, coordenada, ou subvetor.

In [59]:
```
v = [10;20;30;40;-50;35];
```

In [60]:
```
v[1] = 312
```

Out[60]:
```
312
```

In [61]:
```
v
```

Out[61]:
```
6-element Vector{Int64}:
 312
  20
  30
  40
 -50
  35
```

In [62]:
```
# Atribuindo zero para as entradas de 3 a 5.
# Usamos .= para vetores ao invés de = para mudar 1 coordenada
v[3:5] .= 0;
v
```

Out[62]:
```
6-element Vector{Int64}:
 312
  20
   0
   0
   0
  35
```

O mesmo vale para matrizes. É possível ler/mudar linhas e colunas.

In [63]:
```
A = [1 2 3; 4 5 6; 7 8 9]
```

Out[63]:
```
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
```

In [64]:
```
A[1,2] = 0;
A
```

Out[64]:
```
3×3 Matrix{Int64}:
 1  0  3
 4  5  6
 7  8  9
```

In [65]:
```
# mudando a coluna 1 de A
A[:,1] .= [-1;-2;-3];
A
```

Out[65]:
```
3×3 Matrix{Int64}:
 -1  0  3
 -2  5  6
 -3  8  9
```

In [66]:
```
# alterando as entradas 2 e 3 d linha 3
A[3,2:3] .= [10;20];
A
```

Out[66]:
```
3×3 Matrix{Int64}:
 -1   0   3
 -2   5   6
 -3  10  20
```

Códigos salvos em arquivos

Todo código escrito em Julia pode ser salvo em arquivos para ser posteriormente carregado.

A extensão dos arquivos de código é `.jl`

Para carregar os códigos de um arquivo salvo, basta executar

`include("arquivo.jl")`

Se o arquivo está dentro de uma pasta, deve-se passar o caminho relativo contendo o nome da pasta.

Para facilitar, você pode salvar todos os arquivos `.jl` numa mesma pasta.

Exemplo

In [67]:
```
# Inclui o arquivo "codigo.jl"
include("codigo.jl");
```

Nele, estão definidas duas funções `teste` e `teste2` e um vetor `dados`. Ao incluirmos, esses objetos ficam disponíveis.

In [68]:
```
dados
```

Out[68]:
```
4-element Vector{Float64}:
 3.0
 4.0
 7.2
 6.9
```

In [69]:
```
teste()
```

```
Esta função está escrita no arquivo codigo.jl
```

In [70]:
```julia
teste2(dados)
```

```
Vetor de entrada: [3.0, 4.0, 7.2, 6.9]
```

In [71]:
```julia
teste2([2;5;7;8;12])
```

```
Vetor de entrada: [2, 5, 7, 8, 12]
```

Observações:

1. Você pode usar qualquer editor tipo "bloco de notas" para escrever arquivos `.jl`. O Jupyter-lab possui um editor que edita/cria arquivos do tipo.
2. Você pode incluir arquivos em arquivos, isto é, o comando `include` pode ser usado dentro de um arquivo `.jl` para incluir outro arquivo. Se por exemplo `arquivo1.jl` inclue `arquivo2.jl`, ao carregar `arquivo1.jl`, `arquivo2.jl` também será incluído.
3. A separação de códigos em vários arquivos é uma questão de organização apenas. Você quem decidirá como organizar seu código!

Gerenciamento de pacotes

Além das funções básicas que o Julia traz nativamente, podemos usar funções/algoritmos prontos para tarefas específicas.

Um pacote nada mais é que um conjunto de intruções pré-implementadas para um determinado fim.

Por ser *software* livre, qualquer pessoa pode implementar pacotes. A comunidade de otimização é bastante ativa, e logo há inúmeros pacotes disponíveis para uso dentro do Julia.

O gerenciador de pacotes pode ser acessado de dentro do Julia teclando `]`

- **Adicionar um pacote:** `]add PACOTE`
  Isso fará o download automático e instalará o pacote. Uma vez feito, não é necessário instalar novamente o pacote, ele sempre estará disponível para uso
- **Remover um pacote:** `]rm PACOTE`
- **Atualizar todos os pacotes instalados:** `]up`

Para uma lista de pacotes disponíveis, consulte https://juliapackages.com

Alguns pacotes frequentemente utilizados

- `LinearAlgebra`
  Rotinas típicas de álgebra linear (operações eficientes com vetores/matrizes, fatorações, resolução de sistemas lineares)

- `SparseArrays`
  Armazenamento eficiente de matrizes esparsas

- `Plots`
  Plotagem de figuras/gráficos

- `DataFrames`
  Ferramentas para manipulação de dados organizados em tabelas

- `JuMP`
  Escrita de modelos de otimização de "forma natural"

- `NLPModels`, `NLPModelsJuMP`
  Cálculo automático de derivadas dos dados de um modelo de otimização

Usando pacotes

A instalação de um pacote é feita uma única vez:

`]add PACOTE`

A partir de então, o pacote estará disponível para uso sempre que requisitado.

O pacote deverá ser **carregado** antes do uso com o comando

`using PACOTE`

Exemplo:

In [72]:
```
n = 100;
x = rand(n);
```

In [73]:
```
norma = norm(x)
println("norma de x = ",norma)
```

```
norma de x = 6.072663018105091
```

In [74]:
```
# carrega pacote LinearAlgebra (previamente instalado)
using LinearAlgebra

# agora o comando norm está disponível!
norma = norm(x)
println("norma de x = ",norma)
```

```
norma de x = 6.072663018105091
```

Executando códigos

Ao executar uma função/código, principalmente aqueles carregados de arquivos `.jl`, Julia irá compilá-lo na primeira execução. É como se o Julia executasse um compilador ao executar uma função pela primeira vez.

Portanto, é normal que a primeira execução demore mais tempo.

As execuções seguintes serão rápidas, justamente porque o código já estará compilado.

A inserção de pacotes pela primeira vez também costuma levar um tempo maior. Depois de inseridos, eles estarão carregados na memória para rápida execução.

Você perceberá isso na medida em que usa o Julia.

Reforçando: depois de compilados, os códigos em Julia rodam MUITO mais rápido que Matlab! Em processos iterativos, como os algoritmos de otimização, isso traz grande granho em eficiência.

Mais exemplos

Plotando um gráfico - pacote `Plots`

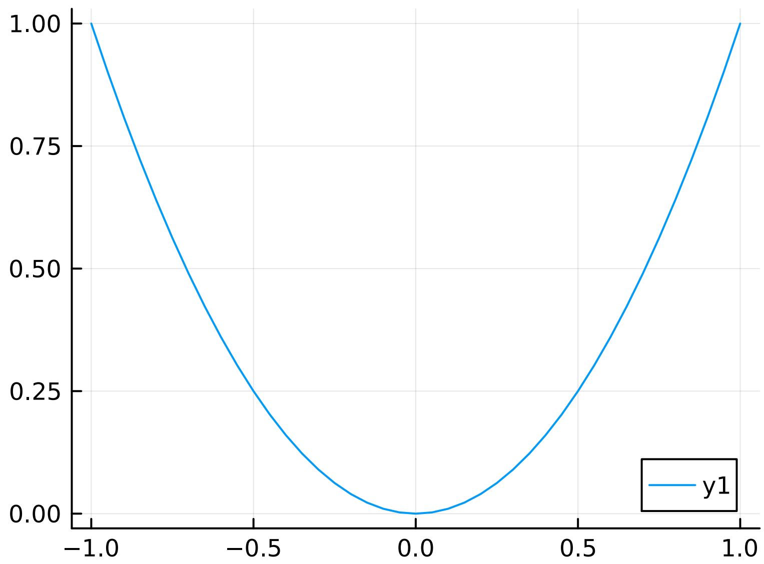
In [75]:
```
using Plots
```

```
WARNING: using Plots.coords in module Main conflicts with an existing iden
tifier.
```

In [76]:
```
x = -1:0.05:1;    # intervalo [-1,1] com passo 0.05
# função y = x^2
y(x) = x^2;
```

In [77]:
```
fig = plot(x, y, size=(400,300))    # constrói a figura
```

Out[77]:

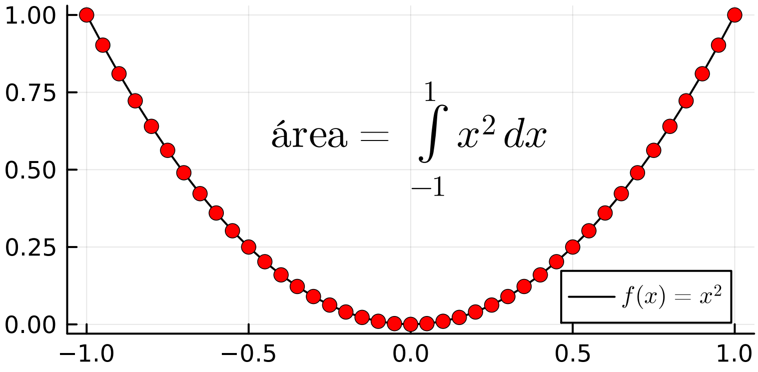


Plots aceitam personalização e sobreposição na mesma figura (observe os comandos com `!`)

In [78]:
```
using LaTeXStrings

# legenda com código LaTex
fig = plot(x, y, label=L"f(x)=x^2", color=:black, size=(400,200))
# pontos laranja em cada x_i
```

```
fig = scatter!(x, y, label="", color=:red);
# anotação na posição (0,0.6)
fig = annotate!(0, 0.6, L"\textrm{área}=\int_{-1}^1 x^2 \, dx");
# salva a figura
savefig(fig, "exemplo.png");
# exibe na tela
display(fig);
```



Algumas opções de personalização

Textos:

- Texto do título: `title="Texto do título"`
- Texto dos eixos: `xlabel="x"` , `ylabel="y"`
- Texto da legenda: `label="f"`

Eixos:

- Marcas nos eixos: `xtick=(0:0.5:10, ["\$ $(i) \$" for i in 0:0.5:10])` , `ytick=-1:0.5:1`
- Limites nos eixos: `xlims=(0,10)` , `ylims=(-1,1)`
- Escala dos eixos: `xscale/yscale=:identity :log10`
- Forçar mesma proporção entre eixos: `aspect_ratio=:equal`

Fontes:

- Tamanho da fonte do título: `titlefont=font(40)`
- Tamanho da fonte dos eixos: `xguidefont=font(30),` `yguidefont=font(20)` ou `guidefont=font(20)`
- Tamanho da fonte das marcas: `xtickfont=font(15),` `ytickfont=font(20)` ou `tickfont=font(10)`
- Tamanho da fonte da legenda: `legendfont=font(12)`
- Mudar tudo para fonte padrão do Latex: `fontfamily="Computer Modern"`

Linhas dos gráficos:

- Espessura da linha do gráfico em pixels: `lw=3`
- Estilo da linha do gráfico: `ls=:solid` (padrão) `:dot` `:dash` `:auto` `:dashdot` `:dashdotdot`

- Cor da linha do gráfico: `color=:black` `:red` `:blue` `:yellow` `:cyan` `:orange` ... ou `color=RGB(.1, .3, 1)`
- Marcas no gráfico: `markershape=:none` (padrão) `:auto` `:circle` `:rect` `:star5` `:diamond` `:hexagon` `:cross` `:xcross` `:utriangle` `:dtriangle` `:rtriangle` `:ltriangle` `:pentagon` `:heptagon` `:octagon` `:star4` `:star6` `:star7` `:star8` `:vline` `:hline` `:+`
- Tamanho das marcas do gráfico em pixels: `markersize=4`

Outras configurações da legenda:

- Posição da legenda: `legend=:right` `:left` `:top` `:bottom` `:inside` `:best` `:topright` `:topleft` `:bottomleft` `:bottomright`
- Ocultar legenda: `leg=false`
- Cor do fundo da legenda: `background_color_legend=:[COR]` ou `background_color_legend=:transparent` (fundo transparente)

Imagem:

- Tamanho da imagem em pixels: `size=(500,400)`
- Preencher área abaixo do gráfico: `fill=(0,:orange,0.5)` (altura referência $y = 0$, cor laranja, 50% de opacidade)

Resolvendo sistemas lineares médios/pequenos

A resolução de sistemas lineares no Julia pode ser feita pelo comando `\`.

In [79]:
```
A = [3 2 1;
     4 6 5;
     7 8 9]
b = [0; 1; 2]

# Resolvendo o sistema Ax = b
x = A\b
```

Out[79]:
```
3-element Vector{Float64}:
 -0.06666666666666665
 -0.06666666666666674
  0.33333333333333337
```

In [80]:
```
norm(A*x - b)
```

Out[80]:
```
5.551115123125783e-17
```

`\` implementa várias técnicas: dependendo da matriz A, Julia decide qual a melhor forma de resolver $Ax = b$. Por exemplo,

- Se A é triangular, então o sistema é resolvido por substituição (A não é fatorada)
- Se A for quadrada e não triangular, $Ax = b$ é resolvido usando fatoração LU
- Se A for retangular, `x = A\b` será a solução de quadrados mínimos computada usando fatoração QR
- Se A for esparsa, LDL^t é usada

Portanto, é útil **fornecer a estrutura da matriz caso você já saiba!** Isso resultará em ganho de eficiência.

In [81]:
```
# Matriz triangular superior
A = [3 2 1;
     0 6 5;
     0 0 9]
```

Out[81]:
```
3×3 Matrix{Int64}:
 3  2  1
 0  6  5
 0  0  9
```

In [82]:
```
b = [1;2;3]
x = A\b    # apesar de A ser triangular, é tratada como uma matriz qualqu
```

Out[82]:
```
3-element Vector{Float64}:
 0.1851851851851852
 0.05555555555555558
 0.3333333333333333
```

In [83]:
```
x = triu(A)\b   # aqui dizemos ao Julia que A é triangular
```

Out[83]:
```
3-element Vector{Float64}:
 0.1851851851851852
 0.05555555555555558
 0.3333333333333333
```

Instalação e execução do Jupyter-lab

Uma maneira de instalar o Jupyter-lab é a partir do próprio Julia:

- Abra o Julia
- Instale o pacote `IJulia` (`]add IJulia`)
  Isso instalará o Jupyter-lab e o núcleo do Julia para o Jupyter-lab, que permite a execução de comandos Julia direto no ambiente gráfico. A instalação é feita uma única vez.

Com o `IJulia` instalado, você poderá abrir o Jupyter-lab **a partir do menu de programas do sistema operacional**. Caso não seja possível, uma opção é abri-lo a partir do Julia:

- Abra o Julia
- Carregue o pacote `IJulia` (`using IJulia`)
- Execute `jupyterlab()`

Como obter ajuda para pacotes e comandos?

As principais fontes de ajuda são:

- Para ajuda com comandos de um pacote específico, consulte a página do pacote. Ela geralmente traz instruções de uso e a documentação completa

- Um atalho que geralmente resolve é o ambiente de ajuda do Julia: teclando `?COMANDO`, uma ajuda com exemplos é exibida
- Fóruns na internet
- Documentação oficial do Julia (https://docs.julialang.org/en/v1)

Usualmente, tudo começa com uma busca do tipo "como fazer tal coisa no julia?"

Exemplo:

1. Como computar a decomposição SVD de uma matriz?...
2. Um dos pacotes que faz isso é o `LinearAlgebra`, que traz o comando `svd` (pode não ser o único pacote)
3. Experimente executar no Julia:
   - Instale `LinearAlgebra` caso não o tenha
   - `using LinearAlgebra`
   - `?svd`

Dicas úteis

1. O terminal de comandos do Julia comporta-se como o GNU/Linux. Você pode começar a digitar um comando e teclar TAB --> TAB que verá as terminações possíveis. Isso dá agilidade e ajuda a lembrar dos comandos
2. Além do próprio Julia, os desenvolvedores de pacotes lançam novas versões com correções e melhorias. Assim, é bom de tempos em tempos atualizá-los executando `]up`. O *download* e instalação das novas versões é automático
3. Várias funções no Julia podem ser aplicadas a cada componente de uma lista (vetores, por exempo). Geralmente isso é indicado com um ponto depois do comando. Por exemplo:
   - `abs(-1.0)` retorna o **número** $|-1|$
   - `abs.([-1.0; 2.0; -8.5])` retorna o **vetor** dos valores absolutos de cada entrada, isto é, a função `abs` é aplicada **a cada componente** do vetor `[-1.0; 2.0; -8.5]`

In [84]:
```julia
abs(-1.0)
```

Out[84]:
```
1.0
```

In [85]:
```julia
abs.([-1.0; 2.0; -8.5])
```

Out[85]:
```
3-element Vector{Float64}:
 1.0
 2.0
 8.5
```

Mensagem importante

Como em qualquer linguagem, você aprenderá com a prática quais pacotes/comandos são adequados para as tarefas que costuma realizar!

Com o tempo, você também começará a buscar melhores práticas de programação (visando eficiência, por exemplo) e pacotes mais adequados.

Isso só você pode fazer!!!

Exercícios

Exercício 1: Reproduza todos os comandos vistos até aqui. Familharize-se com a manipulação de vetores e matrizes, condicionais se...então e laços. Recorra à ajuda contida no Julia ou em outra fonte caso necessário.

Exercício 2: Instale os seguintes pacotes no seu Julia:

`LinearAlgebra`, `SparseArrays`, `Plots`, `DataFrames`, `JuMP`, `NLPModels`, `NLPModelsJuMP`, `MatrixDepot`, `Printf`, `DelimitedFiles`, `GLPK`, `NLPModelsIpopt`, `NLPModelsAlgencan`, `CUTEst`

Exercício 3: Calcule a decomposição a valores singulares (SVD) de matrizes construídas randomicamente. Use o comando `svd` do pacote `LinearAlgebra`. Estude os exemplos contidos na ajuda do comando. Compare o produto $U\Sigma V^t$ das matrizes obtidas na decomposição com a matriz original.

Exercício 4: Procure um comando que compute a decomposição QR de uma matriz. Aplique em matrizes geradas randomicamente.

Exercício 5: Crie um arquivo `.jl` de código com 2 funções que recebam uma matriz e retornem as fatorações SVD e QR. Carregue-o no Julia e aplique as funções à diferentes matrizes.

Escrevendo modelos de otimização - pacote `JuMP`

Vamos considerar o modelo geral de otimização

$$
\begin{aligned}
&\min_x f(x) \\
&\text{s.a. } h_i(x) = 0, \quad i = 1, \ldots, m \\
&\qquad g_j(x) \leq 0, \quad j = 1, \ldots, p \\
&\qquad l_i \leq x_i \leq u_i, \quad i = 1, \ldots, n
\end{aligned}
$$

Vamos supor que todas as funções sejam pelo menos de classe $\mathcal{C}^1$.

O pacote `JuMP` permite escrever modelos de otimização linear, não linear, restritos, irrestritos, com variáveis contínuas e/ou discretas de forma natural.

Exemplo 1

$$\min_x x_1^2 + x_2^2$$

In [86]:
```
# Carrega o pacote. ATENÇÃO: Julia faz distinção entre
# maiúsculas e minúsculas, portanto jump, Jump etc não funcionará
using JuMP
```

In [87]:
```
# cria o modelo em branco
P1 = Model();
```

In [88]:
```
# variáveis x1 e x2
@variable(P1, x[1:2])
```

Out[88]:
```
2-element Vector{VariableRef}:
 x[1]
 x[2]
```

In [89]:
```
# função objetivo não linear, sentido de "minimização"
@objective(P1, Min, x[1]^2 + x[2]^2)
```

Out[89]: $x_1^2 + x_2^2$

In [90]:
```
# imprime o modelo para verificação
print(P1)
```

$$\min \quad x_1^2 + x_2^2$$

Exemplo 2

$$\begin{aligned} &\min_x \sum_{i=1}^{m} (x_i - 5)^2 + \sum_{i=1}^{m-1} (x_{i+1} - x_i)^3 \\ &\text{s.a. } 1 \le x_i \le 4, \quad i = 1, \dots, m \end{aligned}$$

In [91]:
```
using JuMP

m = 3
P2 = Model()

# variáveis e seus limitantes
@variable(P2, 1 <= x[1:m] <= 4)

# função objetivo
@objective(P2, Min, sum((x[i]-5)^2 for i in 1:m) + sum((x[i+1]-x[i])^3 fo
```

Out[91]: $\left(x_1^2 + x_2^2 + x_3^2 - 10x_1 - 10x_2 - 10x_3 + 75\right) + \left((x_2 - x_1)^3\right) + \left((x_3 - x_2)^3\right)$

Observe que a escrita das somas são feitas "como se escreve no papel" com `sum`

Internamente, as somas são expandidas.

In [92]:
```
print(P2)
```

$$
\begin{array}{rl}
\min & \left(x_1^2 + x_2^2 + x_3^2 - 10x_1 - 10x_2 - 10x_3 + 75\right) + \left((x_2 - x_1)^{3.0}\right) + \left((x_3 - x_2)^{3.0}\right) \\
\text{Subject to} & x_1 \geq 1 \\
 & x_2 \geq 1 \\
 & x_3 \geq 1 \\
 & x_1 \leq 4 \\
 & x_2 \leq 4 \\
 & x_3 \leq 4
\end{array}
$$

Exemplo 3

$$
\begin{aligned}
\min_x \, & (x_1 - 2)^2 + (x_2 - 1)^2 \\
\text{s.a. } & x_1 + x_2 - 2 \leq 0 \\
& x_1^2 - x_2 \leq 0
\end{aligned}
$$

In [93]:
```
using JuMP

P3 = Model()
@variable(P3, x[1:2])

# FO não linear
@objective(P3, Min, (x[1]-2)^2 + (x[2]-1)^2)

# Restrição linear
@constraint(P3, x[1] + x[2] - 2 <= 0)

# Restrição não linear
@constraint(P3, x[1]^2 - x[2] <= 0);
```

Exemplo 4

Um investidor quer aplicar seu capital em produtos financeiros (ações, renda fixa etc) selecionados de um *portifólio* de opções.

O investidor espera ter um retorno mínimo predefinido, e sua intenção é minimizar o risco considerando o histórico de retornos de cada opção de investimento.

Dados do problema:

- n: número de produtos no portifólio;
- $\sigma \in \mathbb{R}^n$: vetor dos retornos esperados de cada produto;
- $Q \in \mathbb{R}^{n \times n}$: matriz de covariância. A entrada q_{ij} mede a interdependência entre as opções i e j. Q será simétrica e semidefinida positiva;
- $R > 0$: retorno esperado pelo investidor;
- $u \in [0, 1]^n$: vetor com o máximo percentual a ser investido em cada produto;
- $N \in \mathbb{Z}_+$: número máximo de produtos selecionados. Evidentemente, $N < n$.

Variáveis:

- $x_i \geq 0$: fração do montante total investido no produto i, $i = 1, \ldots, n$;
- $y_i \in [0, 1]$: indica se o produto i foi selecionado ($y_i = 0$) ou não ($y_i \neq 0$), para $i = 1, \ldots, n$.

Modelo:

$$
\begin{align}
\min_{x,y} \ & x^t Q x \tag{1}\\
\text{sujeito a } & \sum_{i=1}^{n} \sigma_i x_i \geq R \tag{2}\\
& \sum_{i=1}^{n} x_i = 1 \tag{3}\\
& x_i y_i = 0, \qquad\qquad\qquad\quad i = 1, \ldots, n \tag{4}\\
& \sum_{i=1}^{n} y_i \geq n - N \tag{5}\\
& 0 \leq x_i \leq u_i, \quad 0 \leq y_i \leq 1, \qquad i = 1, \ldots, n \tag{6}
\end{align}
$$

- Função objetivo: diversifica os produtos (diminuição do risco);
- 1a restrição: o retorno total esperado é pelo menos R;
- 2a restrição : diz que as frações do investimento somam o montante total a ser investido;
- 3o bloco de restrições: diz que se não investimos no produto i ($x_i = 0$), então é permitdo $y_i > 0$;
- 4a restrição: busca contar os produtos não selecionados/investidos, dizendo que eles devem ser, no mínimo, $n - N$ (assim, não selecionamos mais que N produtos);
- 5o bloco de restrições: limitantes das variáveis.

In [94]:

```
# Dados de entrada: n, sigma, Q, R, u, N
function modelo(n,sigma,Q,R,u,N)
    P = Model()

    # Variáveis
    @variable(P, x[1:n] >= 0)    # frações do montante investido
    @variable(P, y[1:n], Bin)    # investe no ativo i?  (variável binária

    @objective(P, Min, x'*Q*x)                                # Função objet
    @constraint(P, sum(sigma[i]*x[i] for i in 1:n) >= R)   # 1a restrição
    @constraint(P, sum(x[i] for i in 1:n) == 1)            # 2a restrição
    for i in 1:n                                              # 3o bloco de
        @constraint(P, x[i]*y[i] == 0)
    end
    @constraint(P, sum(y[i] for i in 1:n) >= n - N)        # 4a restrição
    for i in 1:n                                              # 5a restrição
        set_upper_bound(x[i], u[i])
    end

    return P
end
```

Out[94]: modelo (generic function with 1 method)

In [95]:
```
# Exemplo
n = 3
sigma = ones(n)
Q = Symmetric(rand(n,n)) + 2.0*I(n)
R = 10
u = rand(n)
N = 2

P4 = modelo(n, sigma, Q, R, u, N);
```

In [96]:
```
print(P4)
```

$$
\begin{aligned}
\min \quad & 2.9757156329367507x_1^2 + 0.9207888365916077x_2 \times x_1 + 0.9837944928768199x_3 \times x \\
& + 2.4884373906992776x_3^2 \\
\text{Subject to} \quad & x_1 + x_2 + x_3 = 1 \\
& x_1 + x_2 + x_3 \geq 10 \\
& y_1 + y_2 + y_3 \geq 1 \\
& x_1 \times y_1 = 0 \\
& x_2 \times y_2 = 0 \\
& x_3 \times y_3 = 0 \\
& x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_3 \geq 0 \\
& x_1 \leq 0.8504052023330184 \\
& x_2 \leq 0.13514100565812326 \\
& x_3 \leq 0.9552081205717031 \\
& y_1 \in \{0, 1\} \\
& y_2 \in \{0, 1\} \\
& y_3 \in \{0, 1\}
\end{aligned}
$$

Acessando dados do problema e suas derivadas - pacotes `NLPModels` e `NLPModelsJuMP`

Uma coisa muito conveniente é que não precisamos calcular derivadas de primeira e segunda ordens da função objetivo e restrições à mão, o Julia faz isso por nós. Para tanto, convertemos o modelo Julia para a estrutura `NLPModels` através do pacote `NLPModelsJuMP`.

Isso é extremamente conveniente pois métodos costumam exigir cálculo de derivadas!

Exemplo:

$$
\begin{aligned}
& \min_x f(x) = (x_1 - 2)^2 + (x_2 - 1)^2 \\
& \text{s.a. } g_1(x) = x_1 + x_2 \leq 2 \\
& \qquad g_2(x) = x_1^2 - x_2 \leq 0
\end{aligned}
$$

In [97]:
```julia
using JuMP, NLPModels, NLPModelsJuMP

P3 = Model()
@variable(P3, x[1:2])
@objective(P3, Min, (x[1]-2)^2 + (x[2]-1)^2)
@constraint(P3, x[1] + x[2] <= 2)
@constraint(P3, x[1]^2 - x[2] <= 0);
```

$$
\begin{aligned}
\min_x \ & f(x) = (x_1 - 2)^2 + (x_2 - 1)^2 \\
\text{s.a. } & g_1(x) = x_1 + x_2 \leq 2 \\
& g_2(x) = x_1^2 - x_2 \leq 0
\end{aligned}
$$

In [98]:
```julia
nlp = MathOptNLPModel(P3);    # CONVERTEMOS O MODELO JuMP PARA NLPModels
```

In [99]:
```julia
x = [1.0; 1.0];
```

In [100…
```julia
obj(nlp, x)     # função objetivo em x
```

Out[100…
```
1.0
```

In [101…
```julia
grad(nlp, x)     # gradiente da função objetivo em x
```

Out[101…
```
2-element Vector{Float64}:
 -2.0
  0.0
```

In [102…
```julia
hess(nlp, x)     # hessiana da função objetivo em x
```

Out[102…
```
2×2 Symmetric{Float64, SparseArrays.SparseMatrixCSC{Float64, Int64}}:
 2.0   ⋅ 
  ⋅   2.0
```

$$
\begin{aligned}
\min_x \ & f(x) = (x_1 - 2)^2 + (x_2 - 1)^2 \\
\text{s.a. } & g_1(x) = x_1 + x_2 \leq 2 \\
& g_2(x) = x_1^2 - x_2 \leq 0
\end{aligned}
$$

In [103…
```julia
cons(nlp, x)     # restrições em x (sem o termo livre)
```

Out[103…
```
2-element Vector{Float64}:
 2.0
 0.0
```

ATENÇÃO: `cons` não leva em conta os limitantes das restrições!

O vetor de limitantes superiores é gravado em `nlp.meta.ucon`, enquanto o vetor de limitantes inferiores para as restrições em `nlp.meta.lcon`.

No exemplo anterior, podemos fazer

In [104…
```julia
cons(nlp, x) .- nlp.meta.ucon    # avalia g(x) - u
```

Out[104…
```
2-element Vector{Float64}:
 0.0
 0.0
```

Por que o Julia não considera os limitantes ao avaliar as restrições com `cons` ?
Resposta: para evitar ambiguidade.

Exemplo: suponha que uma restrição tenha limitante superior E inferior:

$L \leq c(x) \leq U.$

Então "avaliar essa restrição" é ambiguo porque, na verdade, são duas restrições ($L - c(x) \leq 0$ e $c(x) - U \leq 0$)

- `cons(nlp, x)` avalia $c(x)$ sem limitantes.
- `nlp.meta.lcon .- cons(nlp, x)` avalia $L - c(x)$
- `cons(nlp, x) .- nlp.meta.ucon` avalia $c(x) - U$

Jacobiano: $\quad J(x) = \begin{bmatrix} \nabla g_1^t(x) \\ \vdots \\ \nabla g_p^t(x) \end{bmatrix}$

In [105…
```
jac(nlp, x)    # Jacobiano em x
```

Out[105…
```
2×2 SparseArrays.SparseMatrixCSC{Float64, Int64} with 4 stored entries:
 1.0   1.0
 2.0  -1.0
```

$$\begin{aligned} &\min_x f(x) \\ &\text{s.a. } h_i(x) = 0,\ i = 1, \ldots, m \\ &\qquad g_j(x) \leq 0,\ j = 1, \ldots, p \end{aligned}$$

Lagrangiana:

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i^h h_i(x) + \sum_{j=1}^{p} \lambda_j^g g_j(x)$$

Hessiana da lagrangiana:

$$\nabla_{xx}^2 L(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i^h \nabla^2 h_i(x) + \sum_{j=1}^{p} \lambda_j^g \nabla^2 g_j(x)$$

In [106…
```
# Hessiana da lagrangeana do exemplo em x = [1.0;1.0] com multiplicador λ
hess(nlp, x, [0.5;1.5])
```

Out[106…
```
2×2 Symmetric{Float64, SparseArrays.SparseMatrixCSC{Float64, Int64}}:
 5.0   ⋅
  ⋅   2.0
```

As vezes precisamos apenas produto "hessiana x vetor" ou "jacobiana x vetor". É a opção preferível por questões de eficiência (calcular produto matriz-vetor não requer armazenar a matriz na memória).

In [107…
```
# ∇^2 L(x,λ) * v
λ = [0.5;1.5]
v = [1.0;-1.0]
hprod(nlp, x, λ, v)
```

Out[107…
```
2-element Vector{Float64}:
  5.0
 -2.0
```

In [108…
```
jprod(nlp, x, v)    # J(x) * v
```

Out[108…
```
2-element Vector{Float64}:
 0.0
 3.0
```

In [109…
```
u = [5.0;4.0]
jtprod(nlp, x, u)    # J(x)^t * u
```

Out[109…
```
2-element Vector{Float64}:
 13.0
  1.0
```

Mais detalhes: https://github.com/JuliaSmoothOptimizers/NLPModels.jl

Exercícios

Exercício 6: Considere a função de Rosenbrock $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Escreva o modelo `JuMP` e converta para a estrutura `NLPModels`. Estude a otimalidede do ponto $x = (1,1)$ calculando no Julia gradiente e hessiana.

Dica: você pode calcular todos os autovalores de uma matriz A executando `using LinearAlgebra; eigvals(Matrix(A))`

Exercício 7: Repita o exercício anterior com a função de Rosenbrock de n variáveis dada por

$$f(x) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$$

Escolha $n \geq 3$ (por exemplo, $n = 10$). **Não expanda a soma à mão!**

Exercício 8: Considere a função $f(x) = x_1^2 - x_1 x_2 + 2x_2^2 - 2x_1 + e^{x_1 + x_2}$. Encontre uma direção $d \in \mathbb{R}^2$ tal que $\nabla f(0,0)^t d < 0$ (use o Julia para fazer a conta).

Exercício 9: Considere os pontos

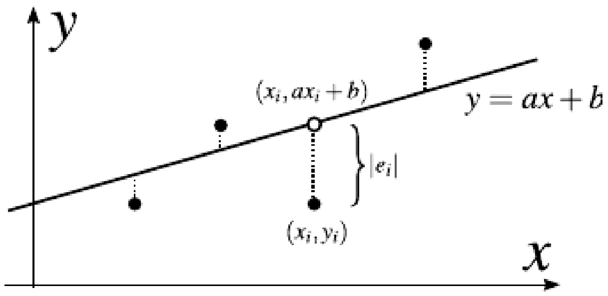
$$(x_1, y_1) = (1,1), \quad (x_2, y_2) = (2,2), \quad (x_3, y_3) = (3,1) \quad \text{e} \quad (x_4, y_4) = (4,3)$$

do plano xy. A reta $y = ax + b$ que melhor aproxima esses pontos é a reta cujos coeficientes minimizam a função de erro

$$f(a,b) = \sum_{i=1}^{4} e_i^2,$$

onde e_i é a diferença entre as alturas do ponto (x_i, y_i) e o ponto $(x_i, ax_i + b)$ da reta, isto é,

$$e_i = y_i - (ax_i + b).$$



Monte o modelo `JuMP` que represente o problema.

Exercício 10: Escreva o modelo `JuMP` de cada um dos problemas abaixo.

- $$\begin{aligned} &\min\ x^2 + y^2 - 6x - 2y + 10 \\ &\quad \text{s.a}\ 2x + y - 2 \leq 0 \\ &\qquad y - 1 \leq 0 \end{aligned}$$
- $$\begin{aligned} &\min\ \ -2x + y \\ &\quad \text{s.a}\ x^2 - y \leq 0 \\ &\qquad y - 4 \leq 0 \end{aligned}$$
- $$\begin{aligned} &\min\ x_1 + x_2 + \cdots + x_n \\ &\quad \text{s.a}\ x_1 x_2 \cdots x_n = 1 \\ &\qquad x_i \geq 0,\ i = 1, \ldots, n. \end{aligned}$$

Exercício 11: Escreva o modelo do GAP (*generalized assignment problem*) em `JuMP`:

$$\begin{aligned} \max_x & \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \\ \text{s.a} & \sum_{j=1}^{n} w_{ij} x_{ij} \leq t_i \qquad i = 1, \ldots, m \\ & \sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n \end{aligned}$$

em que os p_{ij}'s, w_{ij}'s e t_i's são parâmetros positivos dados pelo usuário. Observe que p, w e as variáveis x são **matrizes** $m \times n$, e que o sentido de otimização é **maximizar**.

Consulte https://en.wikipedia.org/wiki/Generalized_assignment_problem para uma descrição detalhada do problema.

Exercício 12: Transforme o segundo modelo do Exercício 10 na estrutura `NLPModels` executando algo como `nlp = MathOptNLPModel(P)`. Estude as propriedades do problema explorando a estrutura `nlp.meta`. Identifique número de variáveis, número de restrições totais, lineares, não lineares, limitantes superiores e inferiores de variáveis e restrições.

Veja a seção "Attributes" em https://github.com/JuliaSmoothOptimizers/NLPModels.jl para uma lista completa.

Métodos de otimização: exemplos

Método de Newton

O método de Newton é um método clássico para resolução de sistemas não lineares. Pode ser usado para minimizar funções gerais pois "resolver" o problema

$$\min_x f(x),$$

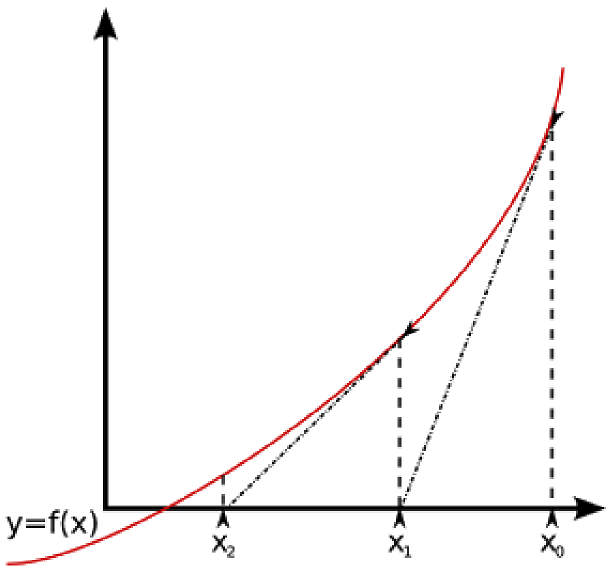
f de classe $\mathcal{C}^2$, consiste em resolver o sistema não linear

$$\nabla f(x) = 0.$$

Considere um sistema não linear com 1 variável e 1 equação $F(x) = 0$.

Ideia Newtoniana:

1. Encontrar um zero de F é difícil... Então trocamos o problema de resolver $F(x) = 0$ por uma sequência de problemas mais fáceis.
2. Dado x^k, o próximo iterando x^{k+1} será zero da aproximação linear de F em x^k.
3. "Se tudo ocorrer bem", x^k tenderá à um zero de F.



Fonte da imagem: https://procesosnumericos2015.weebly.com/meacutetodo-de-newton-raphson.html

Aproximação linear de F em x^k:

$$L(x) = F(x^k) + F'(x^k)(x - x^k).$$

Portanto, x^{k+1} é tal que $L(x^{k+1}) = 0$, isto é,

$$F'(x^k)(x^{k+1} - x^k) = -F(x^k).$$

Chamando $d^k = x^{k+1} - x^k$ (direção Newtoniana), d^k é solução do **sistema Newtoniano**

$$F'(x^k)d = -F(x^k),$$

e o **passo de Newton** é

$$x^{k+1} = x^k + d^k.$$

Isso pode ser feito para sistemas com n variáveis e m equações ($F : \mathbb{R}^n \to \mathbb{R}^m$)...

No caso de interesse, $F = \nabla f : \mathbb{R}^n \to \mathbb{R}^n$, e a iteração Newtoniana fica

$$\nabla^2 f(x^k)d^k = -\nabla f(x^k), \qquad x^{k+1} = x^k + d^k.$$

Critério de parada com a norma do infinito: $\quad \|\nabla f(x)\|_\infty \leq \varepsilon$

Método de Newton

Entrada: ponto inicial $x^0 \in \mathbb{R}^n$ e tolerância para convergência $\varepsilon > 0$

1. enquanto $\|\nabla f(x^k)\|_\infty > \varepsilon$
2. Calcule a direção de Newton d^k resolvendo o sistema Newtoniano $\nabla^2 f(x^k)d = -\nabla f(x^k)$
3. $x^{k+1} = x^k + d^k$
4. $k \leftarrow k + 1$
5. fim

Para cálculo de derivadas, usaremos `JuMP` com `NLPModels`.

In [110…

```
using LinearAlgebra, Printf

# Método de Newton
# Entrada: modelo nlp na estrutura NLPModels, x0 ponto inicial (opcional)

function newton(nlp; x0 = nothing, eps = 1e-6, maxiters = 100, saidas = t
    # lê o número de variáveis da estrutura NLPModels
    n = nlp.meta.nvar

    # Testa se usuário forneceu o ponto inicial. Se não forneceu, inicia
    if isnothing(x0)
        x = zeros(n)
    else
        # aloca vetor solução, copiando x0
        x = deepcopy(x0)
```

```julia
    end

    # contador de iterações
    k = 0

    # computa gradiente e sua norma do infinito
    g = grad(nlp, x)
    norma_g = norm(g, Inf)

    # cabeçalho
    if saidas
        println("Iter  |     norma ∇f |     norma dN")
    end

    while (k <= maxiters)
        # Solução do sistema Newtoniano
        d = hess(nlp, x) \ (-g)

        x .= x + d

        k += 1

        # Atualiza gradiente e norma no novo iterando
        g = grad(nlp, x)
        norma_g = norm(g, Inf)

        # Imprime dados da iteração
        if saidas
            @printf("%5d | %.6e | %.6e\n", k, norma_g, norm(d,Inf))
        end

        # Parar?
        if (norma_g <= eps)
            if saidas
                println()
                println("Problema resolvido com sucesso!")
            end
            # encerra laço while
            break
        end
    end

    # Retorna solução, |∇f|_∞ e número de iterações
    return x, norma_g, k
end
```

Out[110…

```
newton (generic function with 1 method)
```

Exemplo

In [111…

```julia
using JuMP, NLPModels, NLPModelsJuMP

P = Model()
@variable(P, x[1:2])
@objective(P, Min, sin(x[1] - pi/4) + (1-x[2])^3);
```

In [112…

```julia
print(P)
```

$$\min \quad \sin\left(x_1 - 0.7853981633974483\right) + \left(\left(-x_2 + 1\right)^{3.0}\right)$$

In [113…

```
nlp = MathOptNLPModel(P)

newton(nlp);
#newton(nlp, x0=[10.0;-0.5]);
```

```
Iter  |      norma ∇f |      norma dN
    1 | 7.500000e-01 | 1.000000e+00
    2 | 1.875000e-01 | 2.500000e-01
    3 | 4.687500e-02 | 1.250000e-01
    4 | 1.171875e-02 | 6.250000e-02
    5 | 2.929688e-03 | 3.125000e-02
    6 | 7.324219e-04 | 1.562500e-02
    7 | 1.831055e-04 | 7.812500e-03
    8 | 4.577637e-05 | 3.906250e-03
    9 | 1.144409e-05 | 1.953125e-03
   10 | 2.861023e-06 | 9.765625e-04
   11 | 7.152557e-07 | 4.882813e-04

Problema resolvido com sucesso!
```

Método do gradiente

Basicamente, métodos tipo gradiente para minimização irrestrita consistem na iteração

$$x^{k+1} = x^k - t_k \nabla f(x^k)$$

onde $t_k \in (0,1]$ é o **tamanho de passo**.

Sabemos que $d^k = -\nabla f(x^k)$ decresce f **localmente** a partir de x^k caso $\nabla f(x^k)^t d^k < 0$ (direção de descida).

Qual tamanho do passo t_k garante que $f(x^{k+1}) = f(x^k + t_k d^k) \ll f(x^k)$?

Condição de Armijo: Compute $t_k \in (0,1]$ de modo a satisfazer

$$f(x^k + t_k d^k) \leq f(x^k) + \eta t_k \nabla f(x^k)^t d^k$$

onde $\eta \in (0,1)$ é um parâmetro.

Teorema:

Se $\nabla f(x^k)^t d^k < 0$ então existe $\bar{t} \in (0,1]$ tal que a condição de Armijo é satisfeita para todo $t \in (0,\bar{t}]$.

Busca linear inexata com *backtracking*

1. Inicie $t_k \leftarrow 1$
2. Se a condição de Armijo for satisfeita, pare: t_k **é um passo válido**. Caso constrário, atualize $t_k \leftarrow t_k/2$ e repita este passo.

O Teorema anterior garante que o procedimento acima é finito.

Isto é, tentamos primeiro o passo 1 e o dividimos por 2 até que o decréscimo de f segundo Armijo ocorra.

Método do gradiente com backtracking

Entrada: $x^0 \in \mathbb{R}^n$, parâmetro de Armijo $\eta \in (0,1)$, tolerância para convergência $\varepsilon > 0$

1. enquanto $\|\nabla f(x^k)\|_\infty > \varepsilon$
2. $\quad d^k = -\nabla f(x^k)$
3. $\quad t_k \leftarrow 1$
4. $\quad$ enquanto $f(x^k + t_k d^k) > f(x^k) + \eta t_k \nabla f(x^k)^t d^k$
5. $\qquad t_k \leftarrow t_k/2$
6. $\quad$ fim
7. $\quad x^{k+1} = x^k + t_k d^k$
8. $\quad k \leftarrow k+1$
9. fim

In [114…

```
using LinearAlgebra, Printf

# Método do gradiente com backtracking
# Entrada: modelo nlp na estrutura NLPModels, x0 (opcional), eta (opciona

function gradiente(nlp; x0 = nothing, eps = 1e-6, eta = 1e-4, maxiters =
    # lê o número de variáveis da estrutura NLPModels
    n = nlp.meta.nvar

    # Testa se usuário forneceu o ponto inicial. Se não forneceu, inicia
    if isnothing(x0)
        x = zeros(n)
    else
        # aloca vetor solução, copiando x0
        x = deepcopy(x0)
    end

    # aloca x^{k+1}
    xnew = similar(x)

    # contador de iterações
    k = 0

    # cabeçalho
    if saidas
        println("Iter   |     norma ∇f |             t")
    end

    # inicializa variáveis fora do laço while
    norma_g = Inf
    t = 1

    while (k <= maxiters)
        # Direção
        d = -grad(nlp, x)

        norma_g = norm(d, Inf)
```

```
        # Imprime dados da iteração
        if saidas
            @printf("%5d | %.6e | %.6e\n", k, norma_g, t)
        end

        # Parar?
        if (norma_g <= eps)
            if saidas
                println()
                println("Problema resolvido com sucesso!")
            end
            break
        end

        f = obj(nlp, x)
        xnew .= x + t*d
        fnew = obj(nlp, xnew)

        # Busca linear
        t = 1
        gtd = -d'*d
        while fnew > f + eta * t * gtd
            t = t/2
            xnew .= x + t*d
            fnew = obj(nlp, xnew)
        end

        k += 1

        # Atualiza x para próxima iteração
        x .= xnew
    end

    # Retorna solução, |∇f|_∞ e número de iterações
    return x, norma_g, k
end
```

Out[114…

```
gradiente (generic function with 1 method)
```

In [115…

```
using JuMP, NLPModels, NLPModelsJuMP

P = Model()
@variable(P, x[1:2])
@objective(P, Min, x[1]^2 + 10*x[2]^2)

print(P)
```

$$\min \quad x_1^2 + 10x_2^2$$

In [116…

```
nlp = MathOptNLPModel(P)
gradiente(nlp, x0 = [-5.0;3.0]);
```

```
Iter  |     norma ∇f |            t
    0 | 6.000000e+01 | 1.000000e+00
    1 | 1.500000e+01 | 6.250000e-02
    2 | 7.656250e+00 | 1.000000e+00
    3 | 1.500000e+01 | 2.500000e-01
    4 | 3.750000e+00 | 6.250000e-02
    5 | 2.930908e+00 | 1.000000e+00
    6 | 8.437500e+00 | 5.000000e-01
    7 | 2.109375e+00 | 6.250000e-02
    8 | 5.273438e-01 | 1.000000e+00
    9 | 1.318359e-01 | 6.250000e-02
   10 | 3.295898e-02 | 1.000000e+00
   11 | 8.239746e-03 | 6.250000e-02
   12 | 2.059937e-03 | 1.000000e+00
   13 | 5.149841e-04 | 6.250000e-02
   14 | 1.287460e-04 | 1.000000e+00
   15 | 3.218651e-05 | 6.250000e-02
   16 | 8.046627e-06 | 1.000000e+00
   17 | 2.011657e-06 | 6.250000e-02
   18 | 5.029142e-07 | 1.000000e+00

Problema resolvido com sucesso!
```

Método de Newton globalizado

Sabe-se que o método de Newton como apresentado aqui pode não convergir, ou até mesmo falhar.

Exemplo:

In [117…

```
# método de Newton implementado
include("newton.jl")
```

Out[117…

```
newton (generic function with 1 method)
```

In [118…

```
using JuMP, NLPModels, NLPModelsJuMP

P = Model()
@variable(P, x[1:2])
@objective(P, Min, sin(x[1] - pi/4) + (1-x[2])^3)

nlp = MathOptNLPModel(P);
```

In [119…

```
# Executando Newton a partir do ponto (1,1)
newton(nlp, x0 = [1.0;1.0])
```

```
Iter  |     norma ∇f |     norma dN
```

```
SingularException(0)

Stacktrace:
 [1] #lu#7
   @ ~/.julia/juliaup/julia-1.11.3+0.x64.linux.gnu/share/julia/stdlib/v1.11/SparseArrays/src/solvers/umfpack.jl:389 [inlined]
 [2] lu(S::SparseArrays.SparseMatrixCSC{Float64, Int64})
   @ SparseArrays.UMFPACK ~/.julia/juliaup/julia-1.11.3+0.x64.linux.gnu/share/julia/stdlib/v1.11/SparseArrays/src/solvers/umfpack.jl:384
 [3] \(A::Symmetric{Float64, SparseArrays.SparseMatrixCSC{Float64, Int64}}, B::Vector{Float64})
   @ SparseArrays.CHOLMOD ~/.julia/juliaup/julia-1.11.3+0.x64.linux.gnu/share/julia/stdlib/v1.11/SparseArrays/src/solvers/cholmod.jl:1907
 [4] newton(nlp::MathOptNLPModel; x0::Vector{Float64}, eps::Float64, maxiters::Int64, saidas::Bool)
   @ Main ~/Dropbox/Minicurso_Julia/newton.jl:32
 [5] top-level scope
   @ In[119]:2
```

Deu erro!!! Por quê?

Lendo as informações na tela, o erro ocorreu ao tentar resolver o sistema Newtoniano. De fato, Julia tenta resolver o sistema $\nabla^2 f(1,1)d = \nabla f(1,1)$ fazendo a fatoração LU da hessiana (isso está na saida do erro).

Porém, $\nabla^2 f(1,1)$ é singular, e logo não possui fatoração LU:

In [120…

```julia
hess(nlp, [1.0;1.0])
```

Out[120…

```
2×2 Symmetric{Float64, SparseArrays.SparseMatrixCSC{Float64, Int64}}:
 -0.212958   ⋅
   ⋅        0.0
```

Uma solução é descartar a direção de Newton e tomar a direção $-\nabla f(x^k)$.

In [121…

```julia
using LinearAlgebra, Printf

# Método de Newton
# Entrada: modelo nlp na estrutura NLPModels, x0 ponto inicial (opcional)

function newton2(nlp; x0 = nothing, eps = 1e-6, maxiters = 100, saidas =
    # lê o número de variáveis da estrutura NLPModels
    n = nlp.meta.nvar

    # Testa se usuário forneceu o ponto inicial. Se não forneceu, inicia
    if isnothing(x0)
        x = zeros(n)
    else
        # aloca vetor solução, copiando x0
        x = deepcopy(x0)
    end

    # contador de iterações
    k = 0

    # computa gradiente e sua norma do infinito
    g = grad(nlp, x)
    norma_g = norm(g, Inf)
```

```julia
    # cabeçalho
    if saidas
        println("Iter  |     norma ∇f |     norma dN")
    end

    while (k <= maxiters)
        # inicia direção da iteração em branco
        d = []

        # Tenta calcular uma solução do sistema Newtoniano
        try
            d = hess(nlp, x) \ (-g)
        catch
            # caso deu errado, toma a direção -∇f
            d = -grad(nlp, x)
        end

        x .= x + d

        k += 1

        # Atualiza gradiente e norma no novo iterando
        g = grad(nlp, x)
        norma_g = norm(g, Inf)

        # Imprime dados da iteração
        if saidas
            @printf("%5d | %.6e | %.6e\n", k, norma_g, norm(d,Inf))
        end

        # Parar?
        if (norma_g <= eps)
            if saidas
                println()
                println("Problema resolvido com sucesso!")
            end
            # encerra laço while
            break
        end
    end

    # Retorna solução, |∇f|_∞ e número de iterações
    return x, norma_g, k
end
```

Out[121… newton2 (generic function with 1 method)

In [122…
```julia
newton2(nlp, x0=[1.0;1.0])
```

```
Iter  |     norma ∇f |     norma dN
    1 | 7.231395e-01 | 9.770613e-01
    2 | 8.509440e-02 | 7.231395e-01
    3 | 1.030316e-04 | 8.509440e-02
    4 | 1.821378e-13 | 1.030316e-04

Problema resolvido com sucesso!
```

Out[122… ([-0.7853981633972661, 1.0], 1.8213780837848305e-13, 4)

Mas ainda há dois problemas:

1. A direção de Newton pode não ser de descida, isto é, pode ocorrer $\nabla f(x^k)^t d^k > 0$. Neste caso, caminhar na direção de Newton **aumentará** f localmente a partir de x^k, o que não queremos.
2. Mesmo se a direção de Newton for de descida, não controlamos o tamanho do passo ao fazer simplesmente $x + d$... Assim, devemos realizar busca linear (Armijo + *backtracking*) para garantir decréscimo de f

Quando a direção de Newton é de descida?

Uma matriz quadrada A de ordem n é **definida positiva** se $z^t A z > 0$ para todo $0 \neq z \in \mathbb{R}^n$.

Teorema: Se $\nabla^2 f(x^k)$ for definida positiva, então a direção de Newton

$$d^k = -(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$$

é de descida para f a partir de x^k.

Uma maneira de verificar se uma matriz simétrica é definida positiva é através da fatoração de Cholesky.

Fatoração de Cholesky

Dada uma matriz simétrica A, a fatoração de Cholesky consiste em escrever

$$A = GG^t$$

onde G é matriz triangular inferior com diagonal toda positiva.

Teorema: Uma matriz simétrica A é definida positiva se, e somente se, admite fatoração de Cholesky.

Assim, a estratégia é tentar calcular a fatoração de Cholesky de $\nabla^2 f(x^k)$. Se der certo, a direção de Newton está bem definida e é de descida. Caso contrário, ou a direção de Newton não está bem definida ou ela não é de descida, casos em que temos que descartá-la.

Calculando a fatoração de Cholesky no Julia

In [123…
```
using LinearAlgebra
A = Symmetric([4.0 2.0 0.0; 0.0 2.0 1.0; 0.0 0.0 3.0], :U)
```

Out[123…
```
3×3 Symmetric{Float64, Matrix{Float64}}:
 4.0  2.0  0.0
 2.0  2.0  1.0
 0.0  1.0  3.0
```

In [124…
```
F = cholesky(A)
```

Out[124…
```
Cholesky{Float64, Matrix{Float64}}
U factor:
3×3 UpperTriangular{Float64, Matrix{Float64}}:
 2.0  1.0  0.0
  ⋅   1.0  1.0
  ⋅    ⋅   1.41421
```

In [125…
```julia
# F é uma estrutura que guarda a fatoração.
# ELA PODE SER USADA PARA RESOLVER O SISTEMA Ax = b NO LUGAR DE A
b = ones(3)
x = F\b
```

Out[125…
```
3-element Vector{Float64}:
 0.12499999999999997
 0.25000000000000006
 0.24999999999999994
```

In [126…
```julia
norm(A*x-b)
```

Out[126…
```
1.1102230246251565e-16
```

Se temos a fatoração de Cholesky de uma matriz A, é importante usar a fatoração para resolver sistemas $Ax = b$, pois isso será feito resolvendo dois sistemas lineares triangulares, que é mais rápido:

1. $A = GG^t$ (fatoração de Cholesky de A)
2. $y = G^t x$ (sistema triangular superior)
3. $Gy = b$ (sistema triangular inferior)

(verifique que os passos acima levam à $Ax = b$)

Ao fazer `x = F\b`, o Julia internamente faz isso!

Gradientes conjugados

O método dos gradientes conjugados[1,2] é um dos métodos iterativos mais utilizados para minimizar quadráticas com hessiana simétrica e definida positiva:

$$f(x) = \frac{1}{2} x^t A x - b^t x$$

A matriz $n \times n$ simétrica e definida positiva. Minimizar f é equivalente à resolver

$$\nabla f(x) = Ax - b = 0.$$

---

1. Hestenes, Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. Journal of Research of the National Bureau of Standards, 49(6), 1952
2. Nocedal, Wright. Numerical Optimization. 2ed. Springer, New York, 2006

Gradiente Conjugados

Entrada: matriz A, vetor b, $x_0 \in \mathbb{R}^n$ e tolerância para convergência $\varepsilon > 0$

1. (inicialização) $r_0 = Ax_0 - b, \quad p_0 = -r_0, \quad k \leftarrow 0$
2. enquanto $\|r_k\| > \varepsilon$
3. $\quad w_k = Ap_k$
4. $\quad \alpha_k = \dfrac{r_k^t r_k}{p_k^t w_k}$
5. $\quad x_{k+1} = x_k + \alpha_k p_k$
6. $\quad r_{k+1} = r_k + \alpha_k w_k$
7. $\quad \beta_{k+1} = \dfrac{r_{k+1}^t r_{k+1}}{r_k^t r_k}$
8. $\quad p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$
9. $\quad k \leftarrow k + 1$
10. fim

In [127…

```
using LinearAlgebra

# Gradiente conjugados
# A, b e x0 são parâmetros obrigatórios
# eps é parâmetro opcional, cujo valor padrão é 10^{-8}
# maxiters é parâmetro opcional, cujo valor padrão é 100
function cg(A, b, x0; eps = 1e-8, maxiters = 100, saidas = true)
    # captura dimensão de A
    n = size(A,1)

    # aloca vetores na memória
    x = zeros(n)
    r = similar(x)
    p = similar(x)
    w = similar(x)

    # inicialização
    r .= A*x - b
    p .= -r
    k = 0
    norma_r = norm(r)

    # laço principal
    while (norma_r > eps) && (k <= maxiters)
        w .= A*p
        alpha = (r'*r) / (p'*w)
        x .= x + alpha*p
        rtr_old = r'*r
        r .= r + alpha*w
        beta = (r'*r) / rtr_old
        p .= -r + beta*p
        norma_r = norm(r)
        k += 1
        if saidas
            println("Iteração $(k): norma resíduo = $(norma_r)")
        end
    end

    # retorna solução, norma de r e número de iterações
    return x, norma_r, k
end
```

Out[127… cg (generic function with 1 method)

In [128…
```
# construindo uma matriz simétrica e definida positiva de ordem 10
n = 10
A = rand(n,n)
A = A + A' + I(n)
A = Symmetric(A)

b = A*ones(n)

x0 = rand(n)    # ponto inicial

# aplicando gradiente conjugados
x, r, iter = cg(A, b, x0);
```

```
Iteração 1: norma resíduo = 0.9986167745775487
Iteração 2: norma resíduo = 0.1330260767513322
Iteração 3: norma resíduo = 0.15008684844411538
Iteração 4: norma resíduo = 0.1531722077290098
Iteração 5: norma resíduo = 0.05933660676502804
Iteração 6: norma resíduo = 0.15213240513491047
Iteração 7: norma resíduo = 0.29963208708381206
Iteração 8: norma resíduo = 0.0416842458368714
Iteração 9: norma resíduo = 0.134519783554693
Iteração 10: norma resíduo = 0.00025238170972716l2
Iteração 11: norma resíduo = 4.795018912392672e-13
```

In [129…
```
# conferindo a solução
norm(A*x-b)
```

Out[129… 4.773345503698588e-13

Um exemplo mais interessante

O método dos gradientes conjugados é adequado à sistemas com muitas variáveis, e onde A é esparsa.

Vamos utilizar uma matriz da coletânea *Suite Sparse Matrix Collection* (sparse.tamu.edu/)

In [130…
```
using LinearAlgebra
# pacote para baixar matrizes da Suite Sparse Matrix Collection
using MatrixDepot
# pacote para lidar com matrizes esparsas
using SparseArrays

# baixa/lê a matriz HB/1138_bus
A = matrixdepot("HB/1138_bus");

# garantindo tomar a versão esparsa de A
A = sparse(A);
```

```
[ Info: verify download of index files...
[ Info: reading database
EOFError()
```

```
┌ Warning: recreating database file
└ @ MatrixDepot ~/.julia/packages/MatrixDepot/4S7Oa/src/download.jl:59
[ Info: reading index files
[ Info: adding metadata...
[ Info: adding svd data...
[ Info: writing database
┌ Warning: exception during initialization: 'KeyError(MatrixDepot)'
└ @ MatrixDepot ~/.julia/packages/MatrixDepot/4S7Oa/src/MatrixDepot.jl:125
```

Propriedades de A: (https://sparse.tamu.edu/HB/1138_bus)

In [131…

```
n = size(A,1)
# número de não zeros / dimensão A
nnz(A), n
```

Out[131… (4054, 1138)

In [132…

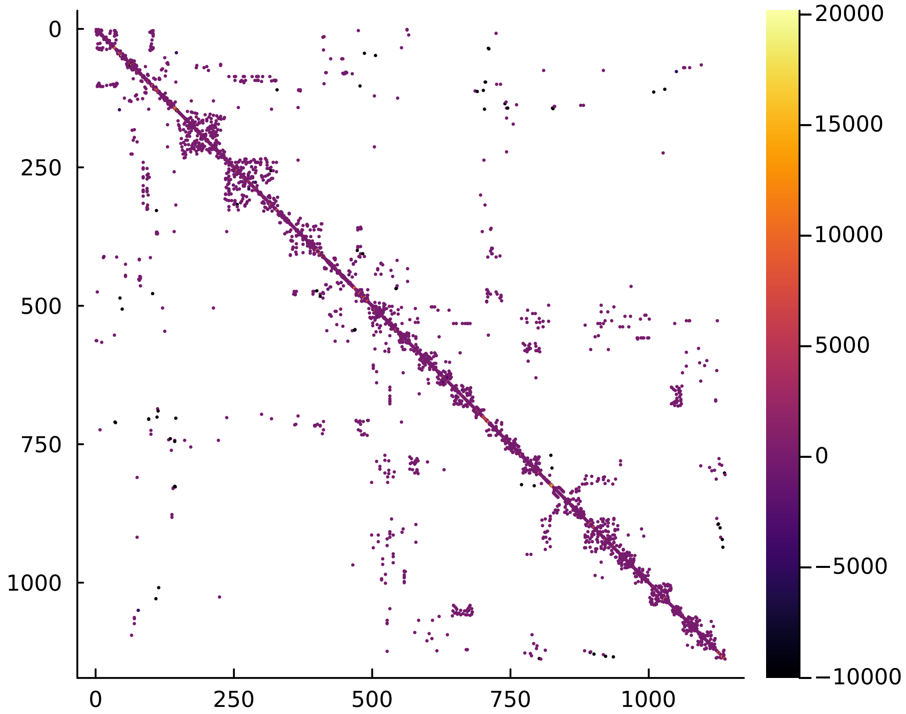
```
# % não zeros
n = size(A,1)
100*nnz(A)/n^2
```

Out[132… 0.31303955695713814

In [133…

```
# estrutura de esparsidade de A
using Plots
spy(A)
```

Out[133…



In [134…

```
# Testando gradientes conjugados
b = A*ones(n)
x0 = rand(n)
x, r, iter = cg(A,b,x0, maxiters=5*n, saidas=false);

# resíduo final, iterações
r, iter
```

Out[134… (7.789662782950453e-9, 2930)

In [135…
```
# contabilizando tempo de resolução
@time cg(A,b,x0, maxiters=5*n, saidas=false);
```

```
0.060184 seconds (70.34 k allocations: 206.077 MiB, 44.28% gc time)
```

Exercícios

Exercício 13: Repita os testes com os códigos fornecidos. Invente problemas e aplique os métodos para visualizar o comportamento e possíveis erros.

Exercício 14: Implemente o método de Newton que faça o seguinte:

1. Tente calcular a fatoração de Cholesky de $\nabla^2 f(x^k)$ para decidir qual direção tomar (Newtoniana ou gradiente)
2. Independentemente da direção escolhida, realize busca linear com Armijo e *backtracking*
3. Pare a execução declarando "função possivelmente ilimitada inferiormente" se $f(x^k) \leq M$ para algum parâmetro $M < 0$ dado pelo usuário. Use como padrão $M = -10^{20}$

Teste sua implementação nos problemas irrestritos dos Exemplos 1, 2, 3 e 4. Também, invente problemas e inicie o método de diferentes pontos.

Exercício 15: Faça uma versão do método do exercício anterior usando gradientes conjugados para resolver o sistema Newtoniano ao invés do operador `\`. Caso gradientes conjugados não convirja ou dê erro, adote a direção de gradiente.

Se inicializarmos um vetor "vazio" no Julia, digamos do tipo numérico para números reais de precisão dupla (`Float64`),

In [136…
```
gs = Float64[]
```

Out[136…
```
Float64[]
```

podemos agregar valores `gs` com o comando `push!` . Por exemplo,

In [137…
```
push!(gs, 10.3)
```

Out[137…
```
1-element Vector{Float64}:
 10.3
```

In [138…
```
push!(gs, 1e-5)
push!(gs, 5.67)
```

Out[138…
```
3-element Vector{Float64}:
 10.3
  1.0e-5
  5.67
```

Isso é útil quando queremos guardar informações das iterações de um algoritmo para plotar gráficos, por exemplo.

Exercício 16: Implemente uma modificação no código do método do gradiente que incialize o vetor `gs` fora do `while` principal, agregue os valor de $\|\nabla f\|_\infty$ em cada iteração, e retorne `gs` no final.

Use o vetor `gs` para plotar um gráfico $\|\nabla f\|_\infty$ em função das iterações.

Para fazer o gráfico, você deve carregar o pacote `Plots` e executar simplesmene

`fig = plot(gs, label="norma grad f")`

Caso queira salvar a figura, execute `savefig(fig, "figura.png")` (extensão pdf também é aceita).

Acesso a bancos de problemas-teste

Um ponto importante quando se faz pesquisa que demande testes computacionais é o uso de problemas-teste (instâncias) consolidados na literatura.

A comunidade de otimização e pesquisa operacional disponibiliza vários bancos de problemas. Vários deles têm interfaces para Julia que facilitam o uso.

Alguns bancos de problemas

Otimização contínua

- CUTEst (*Constrained and Unconstrained Testing Environment with safe threads*): problemas quadráticos e não lineares gerais. Inclui problemas lineares da coletânea NETLIB.
  - Interface: pacote `CUTEst`
  - Página do pacote
- Problemas de quadrados mínimos não lineares de Moré, Garbow e Hillstrom (1981): pacote `NLSModels`
- Problemas irrestritos: pacote `OptimizationProblems`

Programação linear inteira mista e otimização combinatória

- Problema do caixeiro viajante (*Traveling Salesman Problem*): `TSPLIB`
- Problemas de localização de facilidades (*Facility Location Problems*): `FacilityLocationProblems`
- Problema de alocação generalizado (GAP - *Generalized Assignmnent Problem*): `AssignmentProblems`
- Problema de empacotamento (*Bin Packing Problem*): `BPPLib`
- *Capacitated Lot Sizing Problem*: `LotSizingProblems`
- *Multi-Depot Vechile Scheduling Problem*: `MDVSP`
- Problema de roteamento de veículos capacitado (*Capacitated Vehicle Routing Problem*): `CVRPLIB`

- *Inventory Routing Problem*: `InventoryRoutingProblems`
- *Capacitated Arc Routing Problem*: `CARPData`

Banco de matrizes

- Matrizes esparsas da *Suite Sparse Matrix Collection*: inclui inúmeras matrizes de porte pequeno à grande, provenientes de aplicações
  - pacote `MatrixDepot` (usado no exemplo de gradientes conjugados)
  - Página do pacote

Conjuntos de dados para aprendizado de máquina

- Pacote `MLDatasets`

Formatos específicos

- Leitura de arquivos AMPL (`.nl`): pacote `AmplNLReader`
- Leitura de arquivos `MPS` e `QPS` (programação linear e quadrática): pacote `QPSReader`

Fontes sem interface para Julia, mas que podem ser lidas implementando funções adequadas

- Página com links para várias bibliotecas
- Problemas com restrições de equilíbrio/complementaridade

Vamos destacar dois bancos de testes:

- **CUTEst** (***Constrained and Unconstrained Testing Environment with safe threads***)
  Este é um dos principais banco de problemas para testes de algoritmos para programação não linear geral. Contém problemas quadráticos, não lineares gerais e lineares. É amplamente aceita na comunidade como certificado do bom funcionamento de um algoritmo.

- **Matrizes esparsas da *Suite Sparse Matrix Collection***
  É o principal banco de matrizes. É mantida pela Universidade da Flórida. Contém variadas matrizes de diferentes tamanhos, tipos (simétricas ou não, definidas positivas ou não etc). As matrizes provém de aplicações, assim são uma ótima base para testes "verdadeiros".

CUTEst

`CUTEst` é o pacote que lê problemas do banco de problemas. Ao instalar este pacote, todos os problemas da CUTEst são automaticamente baixados, logo não é necessário baixá-los manualmente. Os problemas já vêm no formato `NLPModels`, cujas derivadas podem ser calculadas automaticamente.

In [139…

```
using CUTEst
```

```julia
# carrega o problema BYRDSPHR
nlp = CUTEstModel("BYRDSPHR");
```

In [140…
```julia
display(nlp)
```

```
 Problem name: BYRDSPHR
  All variables: ████████████████████ 3      All constraints: ███████████
█████████ 2
           free: ████████████████████ 3                 free: ···········
········· 0
          lower: ···················· 0                lower: ···········
········· 0
          upper: ···················· 0                upper: ···········
········· 0
        low/upp: ···················· 0              low/upp: ···········
········· 0
          fixed: ···················· 0                fixed: ███████████
█████████ 2
         infeas: ···················· 0               infeas: ···········
········· 0
           nnzh: ( 50.00% sparsity)   3               linear: ···········
········· 0
                                                   nonlinear: ███████████
█████████ 2
                                                        nnzj: (  0.00% sp
arsity)   6
```

Como sei quais problemas estão na CUTEst?

Executando `list_sif_problems()` você verá a lista completa dos problemas.

No momento da escrita desses slides, a coletânea possui 1539 problemas... Então, na prática, é comum escolhermos os problemas por categorias.

In [141…
```julia
# Problemas irrestritos com mínimo de 10 variáveis, máximo de 20,
# somente variáveis livres (sem limitantes)
probs = select_sif_problems(min_var=10, max_var=20, contype="unc", only_f
```

Out[141…
```
 9-element Vector{String}:
  "TRIGON1"
  "STRATEC"
  "PARKCH"
  "HILBERTB"
  "WATSON"
  "TESTQUAD"
  "OSBORNEB"
  "STRTCHDV"
  "TRIGON2"
```

In [142…
```julia
# descarregando problema previamente carregado
finalize(nlp)
# carregando o 9o problema da lista "probs"
nlp = CUTEstModel(probs[9]);
```

Neste ponto, `nlp` já está pronto para ser usado. Por exemplo, podemos aplicar nosso método de gradiente:

In [143…

```
include("gradiente.jl")
gradiente(nlp, maxiters=1000);
```

```
Iter  |      norma ∇f |             t
    0 | 3.914385e+02 | 1.000000e+00
    1 | 7.418185e+01 | 7.812500e-03
    2 | 8.605208e+01 | 1.220703e-04
    3 | 5.790627e+02 | 6.250000e-02
    4 | 1.100824e+03 | 3.906250e-03
    5 | 8.634119e+00 | 3.906250e-03
    6 | 5.177649e+00 | 1.000000e+00
    7 | 2.067650e+00 | 1.953125e-03
    8 | 9.612982e-01 | 1.000000e+00
    9 | 4.353371e-01 | 1.953125e-03
   10 | 2.005628e-01 | 1.000000e+00
   11 | 9.178642e-02 | 1.953125e-03
   12 | 4.214421e-02 | 1.000000e+00
   13 | 1.932249e-02 | 1.953125e-03
   14 | 8.865104e-03 | 1.000000e+00
   15 | 4.066024e-03 | 1.953125e-03
   16 | 1.865168e-03 | 1.000000e+00
   17 | 8.555350e-04 | 1.953125e-03
   18 | 3.924375e-04 | 1.000000e+00
   19 | 1.800103e-04 | 1.953125e-03
   20 | 8.257085e-05 | 1.000000e+00
   21 | 3.787521e-05 | 1.953125e-03
   22 | 1.737336e-05 | 1.000000e+00
   23 | 7.969157e-06 | 1.953125e-03
   24 | 3.655452e-06 | 1.000000e+00
   25 | 1.676755e-06 | 1.953125e-03
   26 | 7.691275e-07 | 1.000000e+00

Problema resolvido com sucesso!
```

Observação: ao carregar um problema da CUTEst, o pacote pode ficar "ocupado" com aquele problema gerando erro ao tentar ler outro problema. Então, para carregar outro modelo, primeiro descarregue o anterior:

In [144…

```
finalize(nlp)     # garantindo que nlp estará liberado
nlp = CUTEstModel("3PK")
```

Out[144…

```
  Problem name: 3PK
   All variables: ████████████████████ 30    All constraints: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
            free: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0               free: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           lower: ████████████████████ 30             lower: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           upper: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0              upper: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
         low/upp: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0            low/upp: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           fixed: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0              fixed: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
          infeas: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0             infeas: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
            nnzh: ( 50.54% sparsity)   230           linear: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
                                                  nonlinear: ⋅⋅⋅⋅⋅⋅⋅⋅⋅
⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
                                                       nnzj: (------%
sparsity)
```

Lista de atributos para filtragem de problemas da CUTEst

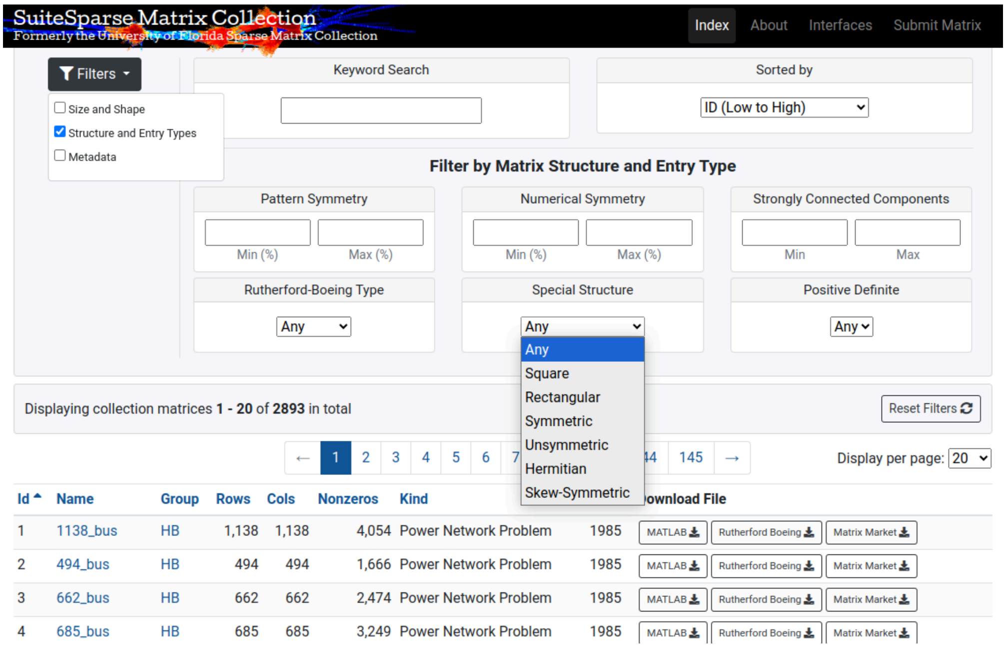
- **max_var=[número]**: número máximo de variáveis
- **min_var=[número]**: número mínimo de variáveis
- **max_con=[número]**: número máximo de restrições ordinárias ($h(x) = 0$ e $g(x) \leq 0$)
- **min_con=[número]**: número mínimo de restrições ordinárias ($h(x) = 0$ e $g(x) \leq 0$)
- **only_free_var=true**: somente problemas com todas as variáveis livres
- **only_bnd_var=true**: somente problemas com variáveis limitadas
- **only_equ_con=true**: somente problemas com restrições ordinárias de igualdade
- **only_ineq_con=true**: somente problemas com restrições ordinárias de desigualdade
- **only_linear_con=true**: somente problemas com restrições ordinárias lineares
- **only_nonlinear_con=true**: somente problemas com restrições ordinárias não lineares

- **objtype=T**: tipo da função objetivo, onde **T** pode assumir
  - **"none"**: sem função objetivo (problema de viabilidade)
  - **"constant"**: função objetivo constante
  - **"linear"**: função objetivo linear
  - **"quadratic"**: função objetivo quadrática
  - **"sum_of_squares"**: função objetivo igual à uma soma de quadrados
  - **"other"**: outro tipo não especificado acima
  - Obs: `CUTEst.objtypes` lista os tipos de função objetivo acima.
- **contype=C**: categoria das restrições, onde **C** pode assumir
  - **"unc"**: sem restrições (problema irrestrito)
  - **"fixed_vars"**: restrições somente fixando variáveis

- **"bounds"**: somente limitantes em variáveis
- **"network"**: restrições representam a matriz de adjacência de uma rede
- **"linear"**: apenas restrições lineares
- **"quadratic"**: apenas restrições quadráticas (inclue as lineares)
- **"general"**: restrições mais gerais que as categorias acima
- Obs: `CUTEst.contypes` lista os tipos de restrições acima.

Suite Sparse Matrix Collection

No site https://sparse.tamu.edu/ você pode consultar as matrizes categorizadas.



Um pacote que lê o banco de matrizes é o `MatrixDepot`

In [145…
```julia
using MatrixDepot
```

In [146…
```julia
# Carregando a matriz 1138_bus do grupo HB
A = matrixdepot("HB/1138_bus");
```

Observação: no primeiro carregamento, a matriz é baixada da internet.

In [147…
```julia
# tamanho de A
size(A)
```

Out[147… (1138, 1138)

In [148…
```julia
# A é simétrica?
issymmetric(A)
```

Out[148… true

In [149…
```julia
# A é deifnida positiva?
try
F = cholesky(A)
println("SIM, pois Cholesky deu certo! :)")
```

```
catch
println("Não :(")
end
```

```
SIM, pois Cholesky deu certo! :)
```

Algumas entradas da coletânea contêm mais do que uma matriz.

Por exemplo, `LPnetlib/lp_25fv47` (https://sparse.tamu.edu/LPnetlib/lp_25fv47) são dados de um problema de programação linear

$$\min_x \ c^t x \quad \text{s.a} \quad Ax = b, \quad l \leq x \leq u$$

In [150…
```
# Carregando TODOS os dados disponíveis em LPnetlib/lp_25fv47
P = mdopen("LPnetlib/lp_25fv47");
```

In [151…
```
P.A
```

Out[151…
```
821×1876 SparseMatrixCSC{Float64, Int64} with 10705 stored entries:
```

In [152…
```
P.b'
```

Out[152…
```
1×821 adjoint(::Matrix{Float64}) with eltype Float64:
 0.0  29.0  60.0  73.0  77.0  27.0  44.0  …  200.0  79.0  148.0  146.0
247.0
```

In [153…
```
P.c'
```

Out[153…
```
1×1876 adjoint(::Matrix{Float64}) with eltype Float64:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  1.5
313
```

Tecle `P.` TAB -> TAB para visualizar todos os elementos carregados em P

Exercícios

Exercício 17: Crie a lista dos problemas da CUTEst que possuem no máximo 1000 variáveis, mínimo de 10 restrições, somente restrições de igualdade com pelo menos uma não linear e função objetivo do tipo "quadrática" ou "soma de quadrados".

Dica: para capturar problemas com função objetivo quadrática OU soma de quadrados, passe o parâmetro

`objtype=["quadratic";"sum_of_squares"]`

Exercício 18: Crie a lista dos problemas irrestritos da CUTEst.

Acesso a solvers de terceiros

A comunidade de otimização / pesquisa operacional implementou e disponibiliza uma quantidade grande de métodos. São códigos feitos puramente em Julia ou interfaces em Julia para pacotes em outras linguagens.

Vamos exemplificar com o uso de alguns pacotes livres.

Ipopt - Interior Point Optimizer

Implementação de um método de pontos interiores para problemas gerais com restrições, descrito em

A. Wächter and L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, Mathematical Programming 106(1), pp. 25-57, 2006

Este pacote é mantido pelo consórcio COIN-OR e é amplamente utilizado pela comunidade acadêmica e industrial. É considerado um dos melhores *solvers* para problemas de programação não linear com restrições.

1. Página do desenvolvedor: https://github.com/coin-or/Ipopt
2. Interface para Julia para modelos na estrutura `NLPModels` : pacote `NLPModelsIpopt`

Exemplo de uso do Ipopt

In [154…

```
using NLPModelsIpopt
```

In [155…

```
using CUTEst

# descarrega nlp caso algum problema foi carregado anteriormente
if @isdefined(nlp) finalize(nlp) end

# carrega o problema BYRDSPHR da CUTEst
nlp = CUTEstModel("BYRDSPHR")
```

Out[155…

```
  Problem name: BYRDSPHR
   All variables: ████████████████████ 3      All constraints: ████████████████████ 2
            free: ████████████████████ 3                 free: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           lower: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0                lower: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           upper: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0                upper: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
         low/upp: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0              low/upp: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
           fixed: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0                fixed: ████████████████████ 2
          infeas: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0               infeas: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
            nnzh: ( 50.00% sparsity)   3               linear: ⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 0
                                                    nonlinear: ████████████████████ 2
                                                         nnzj: (  0.00% sparsity)   6
```

In [156…

```
# Resolve problema com Ipopt
saida = ipopt(nlp)
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

This is Ipopt version 3.14.17, running with linear solver MUMPS 5.7.3.

Number of nonzeros in equality constraint Jacobian...:        6
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        3
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        2
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -5.0000000e+00 1.60e+01 1.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -2.5101106e+02 3.03e+04 1.00e+00  -1.0 1.32e+11    -  1.00e+00 9.31e-10f 31
   2r-2.5101106e+02 3.03e+04 9.99e+02   4.5 0.00e+00    -  0.00e+00 4.77e-07R 22
   3r-1.2914279e+02 7.72e+03 2.03e+00   4.5 3.02e+07    -  1.00e+00 9.92e-04f  1
   4 -6.2806600e+01 1.93e+03 4.97e-01  -1.0 3.09e+01  -4.0 1.00e+00 1.00e+00h  1
   5 -3.1793756e+01 4.81e+02 3.73e-01  -1.0 1.55e+01    -  1.00e+00 1.00e+00h  1
   6 -1.6426563e+01 1.18e+02 3.36e-01  -1.0 7.72e+00    -  1.00e+00 1.00e+00h  1
   7 -9.0128409e+00 2.75e+01 3.03e-01  -1.0 3.75e+00    -  1.00e+00 1.00e+00h  1
   8 -5.7845334e+00 5.21e+00 2.19e-01  -1.0 1.65e+00    -  1.00e+00 1.00e+00h  1
   9 -4.7982378e+00 4.88e-01 7.07e-02  -1.0 5.17e-01    -  1.00e+00 1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10 -4.6848686e+00 6.56e-03 2.78e-03  -1.7 6.50e-02    -  1.00e+00 1.00e+00h  1
  11 -4.6833007e+00 2.34e-06 2.05e-06  -3.8 1.53e-03    -  1.00e+00 1.00e+00h  1
  12 -4.6833001e+00 8.88e-12 1.34e-12  -8.6 2.37e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 12

                                   (scaled)                 (unscaled)
```

```
Objective...............:  -4.6833001326724997e+00   -4.6833001326724997e+00
Dual infeasibility......:   1.3389289676979388e-12    1.3389289676979388e-12
Constraint violation....:   8.8764551264830516e-12    8.8764551264830516e-12
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e+00
Overall NLP error.......:   8.8764551264830516e-12    8.8764551264830516e-12


Number of objective function evaluations             = 23
Number of objective gradient evaluations             = 13
Number of equality constraint evaluations            = 67
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations    = 14
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 12
Total seconds in IPOPT                               = 0.251

EXIT: Optimal Solution Found.
```

Out[156…

```
"Execution stats: first-order stationary"
```

In [157…

```julia
saida.solution   # Solução obtida
```

Out[157…

```
3-element Vector{Float64}:
 0.5000000000000001
 2.0916500663387194
 2.0916500663337803
```

In [158…

```julia
saida.iter   # Número de iterações
```

Out[158…

```
12
```

In [159…

```julia
saida.multipliers   # Multiplicadores de Lagrange
```

Out[159…

```
2-element Vector{Float64}:
  0.6195228609333598
 -0.3804771390666402
```

In [160…

```julia
saida.objective   # Valor FO final
```

Out[160…

```
-4.6833001326725
```

In [161…

```julia
saida.primal_feas   # Violação das restrições
```

Out[161…

```
8.876455126483052e-12
```

Lista dos atributos:

```
julia> saida.
bounds_multipliers_reliable  dual_feas
dual_residual_reliable
elapsed_time                 iter
iter_reliable
```

```
multipliers                multipliers_L
multipliers_U
multipliers_reliable       objective
objective_reliable
primal_feas                primal_residual_reliable
solution
solution_reliable          solver_specific
solver_specific_reliable
status                     status_reliable
time_reliable
```

Algencan - Lagrangeano aumentado

Implementação do método de lagrangeano aumentado com salvaguardas descrito em

R. Andreani, E. G. Birgin, J. M. Martínez, M. L. Schuverdt. On Augmented Lagrangian Methods with General Lower-Level Constraints. SIAM Journal on OptimizationVol. 18(4):1286-1309, 2008

É um pacote desenvolvido por pesquisadores do Brasil. É bem consolidado e muito estudado na literatura. Também resolve problemas de programação não linear com restrições.

1. Página do desenvolvedor: https://www.ime.usp.br/~egbirgin/tango/codes.php
2. Interface Julia para modelos na estrutura `NLPModels` : pacote `NLPModelsAlgencan` (escrito por Paulo J. S. Silva)

A execução de Algencan segue a mesma lógica de Ipopt.

In [162...

```
using NLPModelsAlgencan
using CUTEst

if @isdefined(nlp) finalize(nlp) end
nlp = CUTEstModel("BYRDSPHR");
```

Observação: para instalar Algencan, é necessário alguma implementação da biblioteca de álgebra linear BLAS instalada no sistema. Em sistemas GNU/Linux, será necessário o pacote de sistema `libopenblas-dev` ou similar.

In [163...

```
saida = algencan(nlp)
```

```
 ==============================================================================
=====
 This is ALGENCAN 3.1.1.
 ALGENCAN, an Augmented Lagrangian method for nonlinear programming, is pa
rt of
 the TANGO Project: Trustable Algorithms for Nonlinear General Optimizatio
n.
 See http://www.ime.usp.br/~egbirgin/tango/ for details.
 ==============================================================================
=====

 There are no strings to be processed in the array of parameters.

 The specification file is not being used.

 Available HSL subroutines = NONE

 ALGENCAN PARAMETERS:

 firstde                =                    T
 seconde                =                    T
 truehpr                =                    T
 hptype in TN           =               TRUEHP
 lsslvr in TR           =            NONE/NONE
 lsslvr in NW           =            NONE/NONE
 lsslvr in ACCPROC      =            NONE/NONE
 innslvr                =                   TN
 accproc                =                    F
 rmfixv                 =                    T
 slacks                 =                    F
 scale                  =                    T
 epsfeas                =           1.0000D-08
 epsopt                 =           1.0000D-08
 efstain                =           1.0000D-04
 eostain                =           1.0000D-12
 efacc                  =           1.0000D-04
 eoacc                  =           1.0000D-04
 iprint                 =                   10
 ncomp                  =                    6

 Specification filename =                   ''
 Output filename        =                   ''
 Solution filename      =                   ''

 Number of variables               :       3
 Number of equality constraints    :       2
 Number of inequality constraints  :       0
 Number of bound constraints       :       0
 Number of fixed variables         :       0

 There are no fixed variables to be removed.

 Objective function scale factor   : 1.0D+00
 Smallest constraints scale factor : 1.0D-01

 Entry to ALGENCAN.
 Number of variables  :       3
 Number of constraints:       2

out penalt  objective infeas  scaled    scaled infeas norm   |Grad| inner
```

```
Newton
ite         function  ibilty  obj-funct infeas +compl graLag infeas totit
forKKT
  0        -5.000D+00 2.D+01 -5.000D+00 2.D+00 2.D+00 1.D+00 2.D+00    0
0   0
  1 3.D+01 -5.490D+00 1.D+00 -5.490D+00 1.D-01 1.D-01 2.D-03 3.D-02   10C
0   0
  2 5.D+01 -4.912D+00 3.D-01 -4.912D+00 3.D-02 3.D-02 2.D-05 6.D-03   14C
0   0
  3 5.D+01 -4.746D+00 7.D-02 -4.746D+00 7.D-03 7.D-03 1.D-08 1.D-03   17C
0   0
  4 5.D+01 -4.700D+00 2.D-02 -4.700D+00 2.D-03 2.D-03 2.D-06 4.D-04   19C
0   0
  5 5.D+01 -4.688D+00 5.D-03 -4.688D+00 5.D-04 5.D-04 1.D-08 1.D-04   21C
0   0
  6 5.D+01 -4.684D+00 1.D-03 -4.684D+00 1.D-04 1.D-04 6.D-05 3.D-05   22C
0   0
  7 5.D+01 -4.684D+00 3.D-04 -4.684D+00 3.D-05 3.D-05 4.D-06 7.D-06   23C
0   0
  8 5.D+01 -4.683D+00 9.D-05 -4.683D+00 9.D-06 9.D-06 3.D-07 2.D-06   24C
0   0
  9 5.D+01 -4.683D+00 2.D-05 -4.683D+00 2.D-06 2.D-06 2.D-08 5.D-07   25C
0   0

out penalt  objective infeas  scaled    scaled infeas norm   |Grad| inner
Newton
ite         function  ibilty  obj-funct infeas +compl graLag infeas totit
forKKT
 10 5.D+01 -4.683D+00 7.D-06 -4.683D+00 7.D-07 7.D-07 4.D-09 1.D-07   26C
0   0
 11 5.D+01 -4.683D+00 2.D-06 -4.683D+00 2.D-07 2.D-07 3.D-09 3.D-08   27C
0   0
 12 5.D+01 -4.683D+00 5.D-07 -4.683D+00 5.D-08 5.D-08 3.D-09 9.D-09   28C
0   0
 13 5.D+01 -4.683D+00 1.D-07 -4.683D+00 1.D-08 1.D-08 3.D-09 2.D-09   29C
0   0
 14 5.D+01 -4.683D+00 3.D-08 -4.683D+00 3.D-09 3.D-09 3.D-09 6.D-10   30C
0   0
 15 5.D+01 -4.683D+00 8.D-09 -4.683D+00 8.D-10 8.D-10 3.D-09 2.D-10   31C
0   0

 Flag of ALGENCAN: Solution was found.

 User-provided subroutines calls counters:

 Subroutine fsub     (coded=F):        0
 Subroutine gsub     (coded=F):        0
 Subroutine hsub     (coded=F):        0
 Subroutine csub     (coded=F):        0 (       0 calls per constraint in
avg)
 Subroutine jacsub   (coded=F):        0 (       0 calls per constraint in
avg)
 Subroutine hcsub    (coded=F):        0 (       0 calls per constraint in
avg)
 Subroutine fcsub    (coded=T):       80
 Subroutine gjacsub  (coded=T):       78
 Subroutine gjacpsub (coded=F):        0
 Subroutine hlsub    (coded=T):       31
 Subroutine hlpsub   (coded=F):        0
```

```
Total CPU time in seconds =     0.08
```

Out[163…

```
"Execution stats: first-order stationary"
```

In [164…

```julia
saida.solution
```

Out[164…

```
3-element Vector{Float64}:
 0.5000000067183041
 2.091650072235853
 2.0916500607874013
```

In [165…

```julia
saida.multipliers
```

Out[165…

```
2-element Vector{Float64}:
  0.6195228592651569
 -0.3804771374970072
```

In [166…

```julia
saida.objective
```

Out[166…

```
-4.683300139741558
```

Alterando parâmetros em Ipopt e Algencan

E possível executar Ipopt, Algencan ou qualquer outro método alterando os parâmetros.

Os parâmetros dependem de cada método.

Por exemplo, para executar Ipopt com no máximo 5 iterações, fazemos:

In [167…

```julia
saida = ipopt(nlp, max_iter=5)
```

```
This is Ipopt version 3.14.17, running with linear solver MUMPS 5.7.3.

Number of nonzeros in equality constraint Jacobian...:        6
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        3
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        2
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
   0 -5.0000000e+00 1.60e+01 1.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+
00   0
   1 -2.5101106e+02 3.03e+04 1.00e+00  -1.0 1.32e+11    -  1.00e+00 9.31e-
10f 31
   2r-2.5101106e+02 3.03e+04 9.99e+02   4.5 0.00e+00    -  0.00e+00 4.77e-
07R 22
   3r-1.2914279e+02 7.72e+03 2.03e+00   4.5 3.02e+07    -  1.00e+00 9.92e-
04f  1
   4 -6.2806600e+01 1.93e+03 4.97e-01  -1.0 3.09e+01  -4.0 1.00e+00 1.00e+
00h  1
   5 -3.1793756e+01 4.81e+02 3.73e-01  -1.0 1.55e+01    -  1.00e+00 1.00e+
00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:  -3.1793756184499983e+01   -3.1793756184499983e+
01
Dual infeasibility......:   3.7266958384774895e-01    3.7266958384774895e-
01
Constraint violation....:   4.8090034143823772e+02    4.8090034143823772e+
02
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+
00
Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e+
00
Overall NLP error.......:   4.8090034143823772e+02    4.8090034143823772e+
02


Number of objective function evaluations             = 16
Number of objective gradient evaluations             = 6
Number of equality constraint evaluations            = 60
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 7
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 5
Total seconds in IPOPT                               = 0.002

EXIT: Maximum Number of Iterations Exceeded.
```

Out[167… "Execution stats: maximum iteration"

A lista de parâmetros deverá ser obtida na página de cada pacote.

Lista completa de parâmetros do Ipopt:

- https://coin-or.github.io/Ipopt/OPTIONS.html

Alguns parâmetros de Algencan:

- epsfeas: tolerância para viabilidade
- epsopt: tolerância para norma do infinito do gradiente da função lagrangiano

Consulte https://www.ime.usp.br/~egbirgin/tango/codes.php ou https://doi.org/10.1137/1.9781611973365 para maiores detalhes.

GLPK - GNU Linear Programming Kit

Pacote mantido pela comunidade de software livre. É uma implementação em C dos métodos simplex e variantes, pontos interiores para PL e métodos enumerativos para programação linear inteira mista.

1. Página do desenvolvedor: https://www.gnu.org/software/glpk/
2. Interface Julia: pacote `GLPK`

Ao contrário dos anteriores, GLPK não trabalha com a estrutura `NLPModels`, e sim com no modelo `JuMP` diretamente. De fato, em problemas lineares não faz sentido o cálculo de derivadas automaticamente, já que os dados são todos lineares.

Ao inicializar um modelo `JuMP` em branco, fazíamos `P = Model()`.

Aqui, vamos inciar o modelo já dizendo que o GLPK será utilizado.

In [168…
```
using JuMP
using GLPK
```

In [169…
```
P = Model(GLPK.Optimizer)
```

Out[169…
```
A JuMP Model
├ solver: GLPK
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none
```

Observe que GLPK foi atribuído à `P`, como indica a linha `solver: GLPK`

Escrevemos o modelo como anteriormente...

In [170…
```
# dados para um PL qualquer...
n = 5   # número de variáveis
m = 2   # número de restrições
A = rand(m,n)
```

```
b = A*ones(n)
c = rand(n);   # vetor de custos (FO)
```

In [171…
```
@variable(P, x[1:n]);
```

In [172…
```
# altera limitantes inferiores e superiores de x
for i in 1:n
    set_lower_bound(x[i], -10.0)
    set_upper_bound(x[i], 10.0)
end;
```

In [173…
```
@objective(P, Min, c'*x);
```

In [174…
```
@constraint(P, A*x == b);
```

A novidade aqui é a forma de executar o *solver*:

In [175…
```
optimize!(P)
```

Neste ponto, GLPK foi aplicado e toda saida foi gravada no próprio `P`.

In [176…
```
if termination_status(P) == OPTIMAL   # testando se o problema foi resolv
    println("O problema foi resolvido com sucesso!")
else
    println("O problema não foi resolvido.")
end
```

```
O problema foi resolvido com sucesso!
```

In [177…
```
value.(x)    # valor de cada variável no fim (solução)
```

Out[177…
```
5-element Vector{Float64}:
  10.0
 -10.0
  -7.391493323005019
   4.343160899215796
  10.0
```

In [178…
```
objective_value(P)   # objetivo na solução
```

Out[178…
```
-5.479528777465585
```

Softwares proprietários

Pacotes **proprietários** para programação linear, inteira mista, quadrática e outros:

1. **Gurobi**
   Página do desenvolvedor: https://www.gurobi.com/
   Interface Julia: https://github.com/jump-dev/Gurobi.jl
2. **IBM CPLEX**
   Página do desenvolvedor: https://www.ibm.com/products/ilog-cplex-optimization-studio
   Interface Julia: https://github.com/jump-dev/CPLEX.jl

3. **Xpress**
   Página do desenvolvedor: https://www.fico.com/en/products/fico-xpress-optimization
   Interface Julia: https://github.com/jump-dev/Xpress.jl
4. **Mosek**
   Página do desenvolvedor: https://www.mosek.com/
   Interface Julia: https://github.com/jump-dev/MosekTools.jl
5. **Knitro**
   Página do desenvolvedor: https://www.artelys.com/knitro
   Interface Julia: https://github.com/jump-dev/KNITRO.jl ou https://github.com/JuliaSmoothOptimizers/NLPModelsKnitro.jl

ATENÇÃO: *softwares* proprietários requerem prévia obtenção de licença de uso e instalação manual. A instalação da interface Julia **não** baixa os *solvers* automaticamente.

Outros pacotes selecionados

1. O pacote Julia `Krylov` possui implementações eficientes de métodos "tipo Krylov" para resolução de sistemas lineares, tais como o gradientes conjugados. Veja https://jso.dev/Krylov.jl/stable
2. O projeto `JuliaSmoothOptimizers` possui vários pacotes interessantes para otimização. Veja https://github.com/JuliaSmoothOptimizers
3. O pacote `Optim` possui implementações de vários métodos, como L-BFGS e Newton com regiões de confiança. Veja https://julianlsolvers.github.io/Optim.jl/stable/
4. O projeto JuliaNLSolvers reúne diferentes pacotes, como o `Optim`. Veja https://github.com/JuliaNLSolvers
5. `Tulip` é um método de pontos interiores para PL que implementa técnicas modernas, feito em Julia. Veja https://github.com/ds4dm/Tulip.jl
6. Pacotes para programação cônica (semidefinida e outros): SCS, ProxSDP, DSDP, SDPJSolver
7. `Flux`: pacote para aprendizado de máquina. Veja https://fluxml.ai/ e https://github.com/FluxML/Flux.jl

Exercícios

Exercício 17: Resolva todos os modelos com restrições apresentados usando Ipopt e Algencan. Para cada problema / método, leia a solução obtida. Se possível confira se o ponto calculado é solução ótima, pelo menos aproximadamente.

Exercício 18: Invente instâncias do GAP (Exercício 11) e resolva-as usando o GLPK. Adapte seu código feito no Exercício 11 se necessário.

Lendo arquivos texto estruturados - pacote `DelimitedFiles`

Eventualmente, queremos ler arquivos de texto estruturados de forma organizada. Uma típica situação é quando instâncias para um determinado problema são fornecidas em formato TXT.

Vamos considerar o GAP (generalized assignment problem - Exercício 6) como exemplo.

$$\begin{aligned}
\max_x \; & \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \\
\text{s.a} \; & \sum_{j=1}^{n} w_{ij} x_{ij} \leq t_i \qquad i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n
\end{aligned}$$

Instâncias usadas na literatura para este problema estão discponíveis em

https://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

O arquivo `gap1.txt` começa assim:

```
5
5 15
17 21 22 18 24 15 20 18 19 18 16 22 24 24 16
23 16 21 16 17 16 19 25 18 21 17 15 25 17 24
16 20 16 25 24 16 17 19 19 18 20 16 17 21 24
19 19 22 22 20 16 19 17 21 19 25 23 25 25 25
18 19 15 15 21 25 16 16 23 15 22 17 19 22 24
8 15 14 23 8 16 8 25 9 17 25 15 10 8 24
15 7 23 22 11 11 12 10 17 16 7 16 10 18 22
21 20 6 22 24 10 24 9 21 14 11 14 11 19 16
20 11 8 14 9 5 6 19 19 7 6 6 13 9 18
8 13 13 13 10 20 25 16 16 17 10 10 5 12 23
36 34 38 27 33
```

A primeira linha contém o número de instâncias. Os blocos seguintes seguem o seguinte formato, descrevendo cada instância:

```
m n              (linha 2)
[matrix p]
[matrix w]
[vetor t]
```

O pacote `DelimitedFiles` lê textos estruturados como este e organiza tudo numa tabela (como se fosse uma planilha do Excel)

A partir daí podemos extrair dados de forma simples.

In [179…
```
using DelimitedFiles
dados = readdlm("GAP/gap1.txt")
```

Out[179…
```
61×15 Matrix{Any}:
  5    ""    ""    ""    ""    ""   …    ""    ""    ""    ""    ""    ""
  5  15      ""    ""    ""    ""        ""    ""    ""    ""    ""    ""
 17  21    22    18    24    15          18    16    22    24    24    16
 23  16    21    16    17    16          21    17    15    25    17    24
 16  20    16    25    24    16          18    20    16    17    21    24
 19  19    22    22    20    16       …  19    25    23    25    25    25
 18  19    15    15    21    25          15    22    17    19    22    24
  8  15    14    23     8    16          17    25    15    10     8    24
 15   7    23    22    11    11          16     7    16    10    18    22
 21  20     6    22    24    10          14    11    14    11    19    16
 20  11     8    14     9     5       …   7     6     6    13     9    18
  8  13    13    13    10    20          17    10    10     5    12    23
 36  34    38    27    33      ""        ""    ""    ""    ""    ""    ""
  ⋮                               ⋮    ⋱        ⋮
  5  15      ""    ""    ""    ""        ""    ""    ""    ""    ""    ""
 25  25    18    24    20    19       …  15    18    18    25    15    22
 25  18    17    22    21    23          19    15    18    16    23    16
 18  16    19    15    15    18          24    22    20    25    16    21
 18  21    16    18    17    24          16    17    22    22    18    16
 17  18    15    21    23    21          22    19    15    22    22    25
 16  20     9    22    17    19       …  13     6    20    23    19     7
 12  22    18    18     6    13          14    20    12    17    14    22
  5  19    19    14    24    16          22    13    23    24    15    20
 20   8     6     9     5    17          12    14    17    15    23    21
  6   6    24    24     8     7          18    12    20    20     7    12
 40  38    38    35    34      ""   …    ""    ""    ""    ""    ""    ""
```

In [180…
```
# quantidade de instâncias no arquivo
dados[1,1]
```

Out[180…
```
5
```

In [181…
```
# m, n estão na linha 2, colunas 1, 2
# para garantir que o tipo numérico é inteiro, convertemos as entradas (I
m, n = Int.(dados[2,1:2])
```

Out[181…
```
2-element Vector{Int64}:
  5
 15
```

In [182…
```
# p é o próximo bloco m x n
p = Int.(dados[ 3:(3+m-1) , 1:n])
```

Out[182…
```
5×15 Matrix{Int64}:
 17  21  22  18  24  15  20  18  19  18  16  22  24  24  16
 23  16  21  16  17  16  19  25  18  21  17  15  25  17  24
 16  20  16  25  24  16  17  19  19  18  20  16  17  21  24
 19  19  22  22  20  16  19  17  21  19  25  23  25  25  25
 18  19  15  15  21  25  16  16  23  15  22  17  19  22  24
```

In [183…
```
# w é o próximo bloco m x n
w = Int.(dados[ (3+m):(3+2*m-1) , 1:n])
```

Out[183…
```
5×15 Matrix{Int64}:
  8  15  14  23   8  16   8  25   9  17  25  15  10   8  24
 15   7  23  22  11  11  12  10  17  16   7  16  10  18  22
 21  20   6  22  24  10  24   9  21  14  11  14  11  19  16
 20  11   8  14   9   5   6  19  19   7   6   6  13   9  18
  8  13  13  13  10  20  25  16  16  17  10  10   5  12  23
```

In [184…
```
# t é a próxima linha 1 x m
t = Int.(dados[ 3+2*m , 1:m ])
```

Out[184…
```
5-element Vector{Int64}:
 36
 34
 38
 27
 33
```

Com isso podemos fazer uma função que lê a primeira instância de um arquivo TXT passado como parâmetro:

In [185…
```
function readGAP(arquivo)
    dados = readdlm(arquivo)

    m, n = Int.(dados[2,1:2])

    p = Int.(dados[ 3:(3+m-1) , 1:n])
    w = Int.(dados[ (3+m):(3+2*m-1) , 1:n])
    t = Int.(dados[ 3+2*m , 1:m ])

    return m, n, p, w, t
end
```

Out[185…
```
readGAP (generic function with 1 method)
```

Exercícios

Exercício 19: Baixe as instâncias do GAP na página do minicurso e resolva-as com sua implementação do Exercício 17. Leia os dados das instâncias utilizando a função `readGAP`.

Tabelando resultados - pacote `DataFrames`

Um *dataframe* é basicamente uma tabela com recursos de adição e exclusão de linhas e colunas, ordenamento e filtragem de linhas. É ideal para armazenar resultados de testes.

É possível exportar um dataframe para arquivos de planilha CSV, TXT ou mesmo salvar em arquivos binários.

Primeiro carregamos o pacote `DataFrames`:

In [186…
```
using DataFrames
```

Inicializamos o dataframe dizendo quais as colunas. Por exemplo, o dataframe `resultados` a seguir terá as colunas "Problemas", "iter" e "f":

In [187…
```
resultados = DataFrame(Problema=[],iter=[],f=[])
```

Out[187…
0×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any

O comando `push!` adiciona linhas da seguinte forma:

`push!(dataframe, (DADOS SEPARADOS POR VÍRGULA))`

Não esqueça da exclamação (no Julia uma exclamação indica que o comando atualiza o objeto)

In [188…
```
# adiciona uma linha com dados "prob1", 13, 1e-5
push!(resultados, ("prob1", 13, 1.0e-5) )
```

Out[188…
1×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	prob1	13	1.0e-5

In [189…
```
texto = "prob2"
it = 213
f_val = 4687.4
push!(resultados, (texto, it, f_val) )
push!(resultados, ("outra linha", it*2, f_val/2) )
```

Out[189…
3×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	prob1	13	1.0e-5
2	prob2	213	4687.4
3	outra linha	426	2343.7

Outras funções

In [190…
```
# Deleta a segunda linha
delete!(resultados, 2)
```

Out[190… 2×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	prob1	13	1.0e-5
2	outra linha	426	2343.7

In [191…
```
# Ordena pela primeira coluna
sort!(resultados, 1)
```

Out[191… 2×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	outra linha	426	2343.7
2	prob1	13	1.0e-5

Filtros

Podemos exibir um dataframe resultante de uma busca. O trecho a seguir exibe o dataframe cujas linhas são aquelas de resultados com coluna "iter" menor que 400:

In [192…
```
filter(row -> row[:iter] < 400, resultados)
```

Out[192… 1×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	prob1	13	1.0e-5

O comando a seguir dá o mesmo efeito. Nele interpretamos `resultados` como uma matriz com 3 colunas. Assim, podemos filtrar as linhas e tomar todas as colunas (indicado por `:`)

In [193…
```
resultados[(resultados.iter .< 400),:]
```

Out[193… 1×3 DataFrame

Row	Problema	iter	f
	Any	Any	Any
1	prob1	13	1.0e-5

Sendo uma matriz, podemos filtrar colunas também:

In [194…
```
# somente duas primeiras colunas
resultados[:,1:2]
```

Out[194… 2×2 DataFrame

Row	Problema	iter
	Any	Any
1	outra linha	426
2	prob1	13

Observação: Um dataframe guarda qualquer tipo numérico (inteiros, números reais, vetores, textos etc). No exemplo acima, a primeira coluna guarda texto, a segunda coluna inteiros e a terceira números reais.

Usando `Dataframes` para tabelar resultados de testes

Supõe que temos que executar gradientes conjugados sobre uma lista de problemas e que queiramos guardar os resultados em uma tabela com colunas "Matriz", "n", "iter" e "residuo".

In [195…
```
using DataFrames
using MatrixDepot   # para usar matrizes da Suita Sparse Matrix Collectio
using SparseArrays  # matrizes esparsas

# inclue nossa implementação de gradientes conjugados
include("cg.jl")
```

Out[195…
```
cg (generic function with 1 method)
```

In [196…
```
# Matrizes selecionadas, organizadas em um vetor de textos
matrizes = ["bcsstm21"; "bcsstm02"; "bcsstk27"; "bcsstk06"; "662_bus"]
```

Out[196…
```
5-element Vector{String}:
 "bcsstm21"
 "bcsstm02"
 "bcsstk27"
 "bcsstk06"
 "662_bus"
```

In [197…
```
# Inicializa o dataframe
resultados = DataFrame(Matriz=[],n=[],iter=[],residuo=[]);
```

Temos que executar `cg` em nos problemas construídos com cada matriz do vetor `matrizes`

In [198…
```
for matriz in matrizes
    A = matrixdepot("*/$(matriz)") # carrega a matriz
    n = size(A,1)                  # dimensão (todas as matrizes são quad
    b = A*ones(n)                  # lado direito

    # aplica cg em um bloco protegido de erros
    try
        # a função cg retorna x, o resíduo e o número de iterações (nessa
```

```
        # como x não interessa, colocamos ~
        ~, residuo, iter = cg(A, b, zeros(n), maxiters = 5*n, saidas = fa

        # se chegou até aqui, não deu erro... grava resultados no datafra
        push!(resultados, (matriz, n, iter, residuo))
    catch
        println("ERRO (matriz $(matriz))!")
    end
end

# ordena resultados pela primeira coluna
sort!(resultados, 1);
```

In [199…

```
resultados
```

Out[199…

5×4 DataFrame

Row	Matriz	n	iter	residuo
	Any	Any	Any	Any
1	662_bus	662	746	8.14402e-9
2	bcsstk06	420	2101	24086.9
3	bcsstk27	1224	1792	9.68412e-9
4	bcsstm02	66	12	9.95503e-17
5	bcsstm21	3600	3	1.9068e-15

Salvando resultados em arquivos

Podemos salvar o dataframe `resultados` em diferentes formatos. Alguns deles:

- **Arquivo TXT:** é o formato mais simples. Geralmente códigos em C trabalham com TXT.
- **Planilha CSV:** CSV é um formato livre para planilhas, compatíveis com editores como Excel e OpenOffice. É um formato mais completo, que permite tratamento, filtragem dos dados e muito mais.
  - Página do pacote `CSV`
- **Arquivo binário `JLD2`:** tipo de arquivo que só pode ser lido dentro do Julia. Podemos gravar qualquer objeto definido no Julia (dataframes, funções, imagens etc). A vantagem é que os dados são gravados sem perda: por exemplo, se um número real tem 16 casas decimais de precisão, nunca ocorrerá truncamentos. É útil quando queremos guardar objetos (não códigos) para carregar no Julia posteriormente, por exemplo, pontos iniciais gerados por alguma estratégia que não se pode repetir rapidamente.
  - Página do pacote `JLD2`

Com o dataframe `resultados` em mãos, é fácil gravar os arquivos.

Arquivos TXT

In [200…
```
using Printf    # para impressão estilo C

# Cria o arquivo "saida.txt", SUBSTITUINDO CASO JÁ EXISTA
txt = open("resultados.txt", "w")

write(txt, @sprintf("%s", resultados))   # Escreve os dados no arquivo
close(txt);    # Fecha o arquivo
```

Planilhas CSV

In [201…
```
using CSV
CSV.write("resultados.csv", resultados);
```

Arquivos binários

Uso:

- Gravação: `jldsave("arquivo.jld2"; [OBJETOS])`
- Leitura: `dados = jldopen("arquivo.jld2", "r")`

In [202…
```
using JLD2
jldsave("resultados.jld2"; resultados)
```

In [203…
```
# Lendo o conteúdo do arquivo "resultados.jld2" para o objeto "dados"
dados = jldopen("resultados.jld2", "r")
```

Out[203…
```
JLDFile /home/secchin/Dropbox/Minicurso_Julia/resultados.jld2 (read-onl
y)
 └─🔢 resultados
```

In [204…
```
dados["resultados"]
```

Out[204… 5×4 DataFrame

Row	Matriz	n	iter	residuo
	Any	Any	Any	Any
1	662_bus	662	746	8.14402e-9
2	bcsstk06	420	2101	24086.9
3	bcsstk27	1224	1792	9.68412e-9
4	bcsstm02	66	12	9.95503e-17
5	bcsstm21	3600	3	1.9068e-15

Exercícios

Exercício 20: Repita os testes com gradientes conjugados. Utilize outras matrizes da Suite *Sparse Matrix Collection* à sua escolha (cuidado para não exagerar no tamanho das matrizes).

Exercício 21: Tabele os resultados do Exercício 19. Salve-os e arquivos TXT e CSV. Abra o último no editor de planilhas de sua preferência.