Introdução à Julia

Oficina de Ferramentas Computacionais

Abel Soares Siqueira
Universidade Federal do Paraná

30 de Maio de 2015

Julia

- Alto nível;
- Alta performance;
- Sintaxe fácil;
- Interface com C/Fortran e Python.

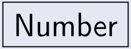
Mão na massa

```
$ julia
julia> 2+3
julia> 5*8
julia> 9/2
julia> 2^3
julia> exp(1)
```

Mão na massa

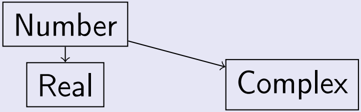
```
julia> round(exp(1))
julia> floor(exp(1))
julia> ceil(exp(1))
julia> div(13,4)
julia> rem(13,4)
julia> mod(13,4)
```

Hierarquia de tipo

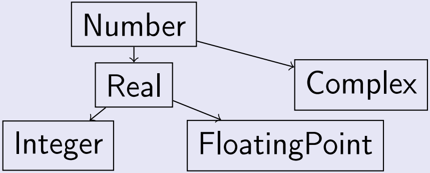


Number

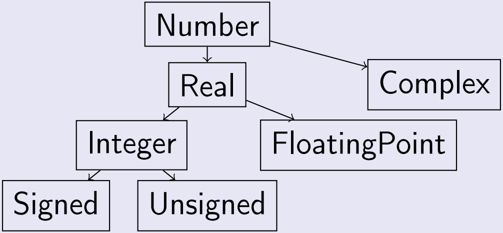
Hierarquia de tipo



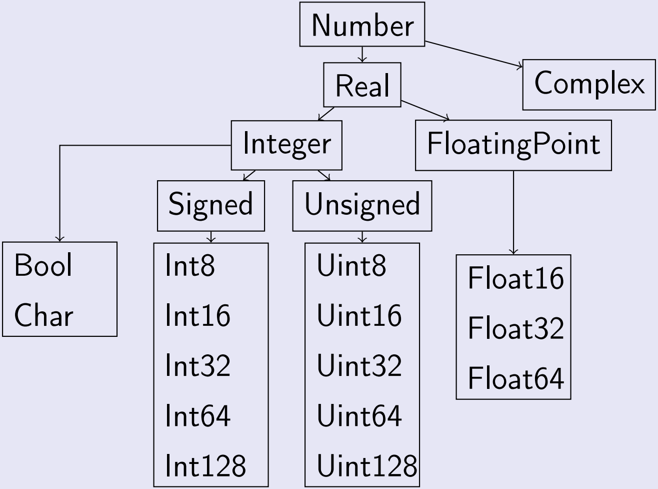
Hierarquia de tipo



Hierarquia de tipo



Hierarquia de tipo



```
julia> round(2.3)
julia> iround(2.3)
julia> # Na v0.4 iround não existe mais
julia> # Use round(Int, 2.3)
julia> typeof(2)
julia> typeof(2.0)
julia> 2 == 2.0
julia> isapprox(exp(1), 2.71828)
```

HELP

```
julia> help(exp)
julia> ?exp
```

test1.jl

```
2+3
println(3+4)
```

```
$ julia test1.jl
julia> include("test1.jl")
julia> rand(3)
julia> rand(3)

julia> A = rand(3,3)

julia> e = ones(3)
julia> rand(3)
julia> A = rand(3,3)
julia> e = ones(3)
julia> b = A*e
julia> rand(3)

julia> A = rand(3,3)

julia> e = ones(3)

julia> b = A*e

julia> x = A\b

julia> norm(x-e)

julia> x[1]
julia> A = rand(3,5)

julia> A[2,3] = 0.0

julia> A
julia> A = rand(3,5)

julia> A[2,3] = 0.0

julia> A

julia> (m,n) = size(A)  # Ou m,n = size(A)

julia> A'*A
julia> A = rand(3,5)
julia> A[2,3] = 0.0
julia> A
julia> (m,n) = size(A)  # Ou m,n = size(A)
julia> A’*A
julia> ones(3,5)
julia> zeros(3,5)
julia> eye(3,5)
julia> (L,U,P) = lu(A)

julia> norm(L*U-A[P,:])
julia> (L,U,P) = lu(A)

julia> norm(L*U-A[P,:])

julia> G = chol(A*A’)

julia> norm(G’*G-A*A’)
julia> (L,U,P) = lu(A)

julia> norm(L*U-A[P,:])

julia> G = chol(A*A’)

julia> norm(G’*G-A*A’)

julia> (U,S,V) = svd(A)

julia> norm(U*diagm(S)*V’-A)
julia> (L,U,P) = lu(A)
julia> norm(L*U-A[P,:])
julia> G = chol(A*A’)
julia> norm(G’*G-A*A’)
julia> (U,S,V) = svd(A)
julia> norm(U*diagm(S)*V’-A)
julia> (Q,R) = qr(A)
julia> norm(Q*R-A)
```

Cuidado com matrizes disfarçadas

```
julia> [1;2;3]  # Array
julia> [1,2,3]  # Array
julia> [1 2 3]  # Array 1x3
julia> [1 2 3]’  # Array 3x1
```

Cuidado com matrizes disfarçadas

```
julia> [1;2;3]  # Array
julia> [1,2,3]  # Array
julia> [1 2 3]  # Array 1x3
julia> [1 2 3]’  # Array 3x1
julia> [1;2;3] == [1,2,3]
julia> [1 2 3]’ == [1,2,3]
```

if, elseif, else

```
if ALGO
  CMDs
elseif OUTRO ALGO
  CMDs
else
  CMDs
end
```

ifelse.jl

```
if x > 0
  println("positivo")
elseif x == 0
  println("zero")
else
  println("negativo")
end
```

```
julia> x = 1

julia> include("ifelse.jl")

julia> x = 0

julia> include("ifelse.jl")

julia> x = -1

julia> include("ifelse.jl")
```

while

```
while ALGO
  CMDs
end
```

while.jl

```
while n != 1
  println("n = ", n)
  if n % 2 == 0
    n = n/2
  else
    n = 3*n+1
  end
end
```

```
julia> n = 3
julia> include("while.jl")
```

for

```
for VAR in RANGE
  CMDs
end
for VAR = RANGE
  CMDs
end
```

for.jl

```julia
for n in [1 10 100 200]
  println("log10($n) = ", log10(n))
end

for i = 1:10
  println("$i^2 = $(i^2)")
end
```

```
julia> include("for.jl")
julia> for i = 2:3:15
           println("$i")
       end
julia> typeof(1:10)
```

Comparações curtas

```
julia> 2 > 0
julia> 2 > 0 && println("ok")
julia> 2 < 0 && println("ok")
julia> error()
julia> 2 > 0 && error()
julia> 2 > 0 || println("ok")
julia> 2 < 0 || println("ok")
```

Comparações curtas

```
julia> 1 > 0 ?  println("ok") :  println("not")
julia> 1 < 0 ?  println("ok") :  println("not")
```

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

bhas.jl

```julia
function bhaskara(a, b, c)
  D = b^2 - 4*a*c;
  D = sqrt(D);
  return (-b+D)/(2*a), (-b-D)/(2*a)
end
```

```
julia> include("bhas.jl")
julia> bhaskara(1,5,6)
julia> include("bhas.jl")

julia> bhaskara(1,5,6)

julia> r = bhaskara(1,5,6)

julia> r[1]
julia> include("bhas.jl")

julia> bhaskara(1,5,6)

julia> r = bhaskara(1,5,6)

julia> r[1]

julia> r1,r2 = bhaskara(1,5,6)

julia> r1
julia> include("bhas.jl")

julia> bhaskara(1,5,6)

julia> r = bhaskara(1,5,6)

julia> r[1]

julia> r1,r2 = bhaskara(1,5,6)

julia> r1

julia> bhaskara(1,0,1)
```

bhas.jl

```julia
function bhaskara(a, b, c)
  D = b^2 - 4*a*c;
  if D >= 0
    D = sqrt(D);
  else
    D = im*sqrt(-D);
  end
  return (-b+D)/(2*a), (-b-D)/(2*a)
end
```

```
julia> include("bhas.jl")

julia> bhaskara(1,0,-1)

julia> bhaskara(1,0,1)
```

bhas.jl

```
function bhaskara(a, b, c)
  D = b^2 - 4*a*c;
  D = D >= 0 ? sqrt(D) : im*sqrt(-D)
  return (-b+D)/(2*a), (-b-D)/(2*a)
end
```

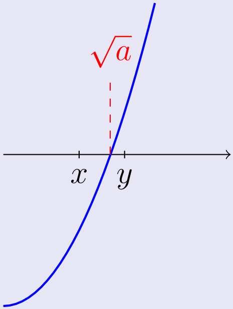
$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{i,j}^2}$$

normF

```
function normF(A::Matrix)
  (m,n) = size(A);
  s = 0.0
  for i = 1:m
    for j = 1:n
      s += A[i,j]^2
    end
  end
  return sqrt(s)
end
```

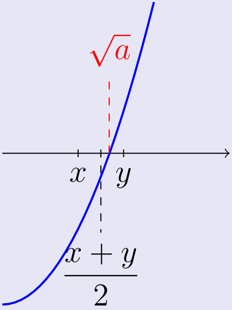
Exemplo: calculando a raiz de $a > 1$

$$f(x) = x^2 - a$$



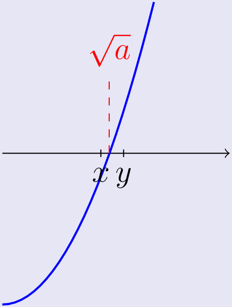
Exemplo: calculando a raiz de $a > 1$

$$f(x) = x^2 - a$$



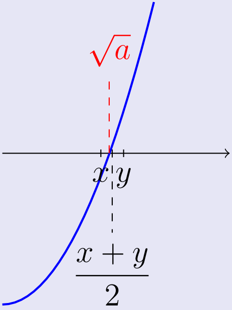
Exemplo: calculando a raiz de $a > 1$

$$f(x) = x^2 - a$$



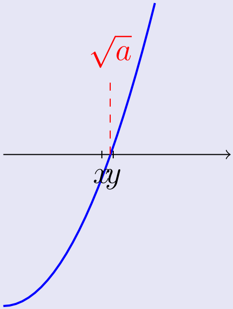
Exemplo: calculando a raiz de $a > 1$

$$f(x) = x^2 - a$$



Exemplo: calculando a raiz de $a > 1$

$$f(x) = x^2 - a$$



bissec.jl - v1

```
function bissec (a)
  x = 1
  y = a
  m = (x+y)/2
  while abs(m^2-a) > 1e-4
    if m^2 > a
      y = m
    else
      x = m
    end
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v2

```
function bissec (a)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  x, y = 1, a
  m = (x+y)/2
  while abs(m^2-a) > 1e-4
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v3

```
function bissec (a, tol, x, y)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  m = (x+y)/2
  while abs(m^2-a) > tol
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v4

```
function bissec (a, tol = 1e-6, x = 1, y = a)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  m = (x+y)/2
  while abs(m^2-a) > tol
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v5

```
function bissec (a; tol = 1e-6, x = 1, y = a)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  m = (x+y)/2
  while abs(m^2-a) > tol
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v6

```
function bissec (a::Float64; tol::Float64 = 1e-6, x::Float64 = 1,
    y::Float64 = a)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  m = (x+y)/2
  while abs(m^2-a) > tol
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

bissec.jl - v7

```
function bissec (a::Number; tol::Number = 1e-6, x::Number = 1,
    y::Number = a)
  a < 1 && error("Raiz de numero negativo nao existe nos reais")
  m = (x+y)/2
  while abs(m^2-a) > tol
    m^2 > a ? (y = m) : (x = m)
    m = (x+y)/2
  end
  return m
end
```

```
julia> f(x) = x^2

julia> f(1)

julia> f(2)

julia> f(-1)
julia> f(x) = x^2

julia> f(1)

julia> f(2)

julia> f(-1)

julia> g = x->x^3 - x

julia> g(1)

julia> g(2)

julia> g(-1)
julia> f(x) = x^2

julia> t = [1,2,3]

julia> f(t)  # Erro

julia> map(f, t)
julia> f(x) = x^2
julia> t = [1,2,3]
julia> f(t)  # Erro
julia> map(f, t)
julia> f(x) = x.^2
julia> f(t)  # Ok
julia> t.^3
julia> exp(t).*t
julia> 1./t
julia> h(x,y) = x^2-y^2

julia> h(2,1)
julia> h(x,y) = x^2-y^2
julia> h(2,1)
julia> f(x) = h(x,2)
julia> h(x,y) = x^2-y^2
julia> h(2,1)
julia> f(x) = h(x,2)
julia> q(x) = 0.5*dot(x,x)
julia> q(ones(10))
julia> h(x,y) = x^2-y^2
julia> h(2,1)
julia> f(x) = h(x,2)
julia> q(x) = 0.5*dot(x,x)
julia> q(ones(10))
julia> D(x,f,h) = (f(x+h)-f(x)/h)
julia> D(1,x->x^2,0.1)
julia> # f dependendo dela mesmo?
julia> f(x) = f(x-1)*x  # Loop infinito ou erro
julia> # f dependendo dela mesmo?
julia> f(x) = f(x-1)*x  # Loop infinito ou erro
julia> f(x) = x > 0 ?  x^2+1 :  2*x+1
julia> fat(x::Integer) = x > 1 ?  fat(x-1)*x :  1
julia> function foo(a::Float64, v::Array{Float64})
           a = 2
           v[1] = 0.0
           return a, v
       end
julia> a = 3
julia> v = rand(3)
julia> b, w = foo(a,v)
julia> function foo(a::Float64, v::Array{Float64})
           a = 2
           w = copy(v)
           w[1] = 0.0
           return a, w
       end
julia> a = 3
julia> v = rand(3)
julia> b, w = foo(a,v)
julia> function bar(x::Int)
           return x+1
       end
julia> function bar(x::FloatingPoint)
           return 1/x
       end
julia> bar(2)
julia> bar(2.0)
```

Interação com C e Fortran

ccall((FUNCAO, BIBLIOTECA), RETURN, (TIPOS, DE, ENTRADA), ENTRADAS)

- A biblioteca tem que ser dinâmica (shared);
- Os tipos estão numa tupla;
- Existem os tipos `Cint = Int32`, `Cfloat = Float32` e `Cdouble = Float64`.

Interação com C e Fortran

ccode/dot.c

```
double dotC (int n, double *x, double *y) {
  int i;
  double s = 0.0;

  for (i = 0; i < n; i++)
    s += x[i]*y[i];

  return s;
}
```

```
ccall( ("dotC", -), Cdouble, (Cint, Ptr{Cdouble}, Ptr{Cdouble}), n
    , x, y)
```

Interação com C e Fortran

```
$ gcc -c -o dot.o dot.c -fPIC
$ ld -shared -o libtestC.so dot.o
```

Interação com C e Fortran

testC.jl

```
n = 100
s = 0.0
for i = 1:100
  v = rand(n)
  w = rand(n)
  d = ccall(("dotC", "ccode/libtestC.so"), Cdouble,
    (Cint, Ptr{Cdouble}, Ptr{Cdouble}), n, v, w)
  s += abs(d-dot(v,w))
end
println("Erro = $s")
```

Interação com C e Fortran

fcode/dot.f

```
      subroutine dotF (N, X, Y, D)

        integer N
        double precision X(N), Y(N)
        double precision D

        integer i

        D = 0.0D0
        do i = 1,N
          D = D + X(i)*Y(i)
        end do

      end subroutine dotF
```

```
n = 100
d = [0.0] #Um vetor
ccall( ("dotf_", -), Void, (Ptr{Int32}, Ptr{Float64},
  Ptr{Float64}, Ptr{Float64}), Int32[n], x, y, d)
```

Interação com C e Fortran

```
$ gfortran -c -o dot.o dot.f -fPIC
$ ld -shared -o libtestF.so dot.o -gfortran
```

Logo

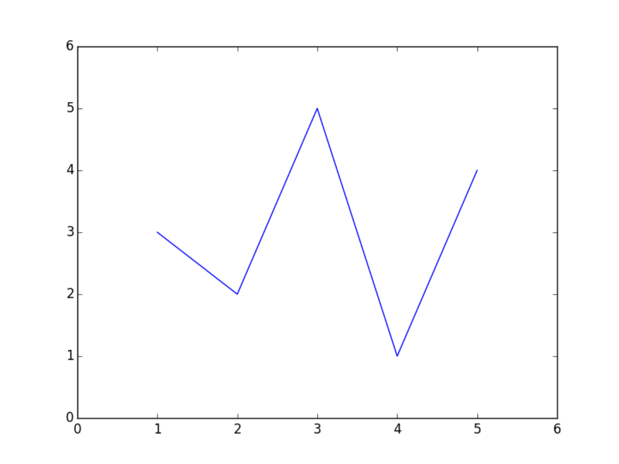
Interação com C e Fortran

testF.jl

```
n = 100
s = 0.0
for i = 1:100
  v = rand(n)
  w = rand(n)
  d = [0.0]
  ccall(("dotf_", "fcode/libtestF.so"), Void, (Ptr{Int32}, Ptr{
      Float64},
    Ptr{Float64}, Ptr{Float64}), Int32[n], v, w, d)
  s += abs(d[1]-dot(v,w))
end
println("Erro = $s")
```

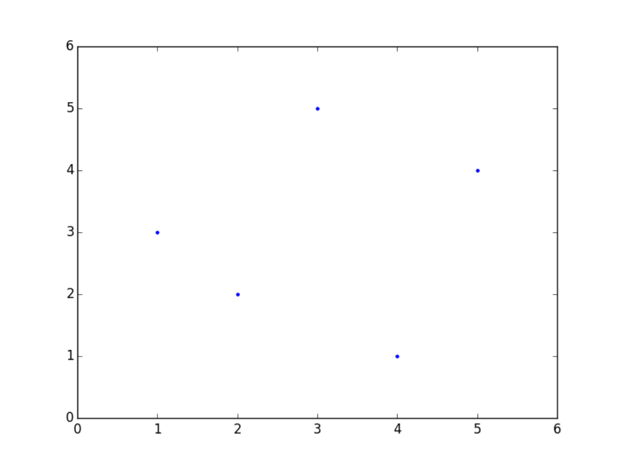
- Não existe pacote padrão;
- Algumas opções são PyPlot, Gadfly e Winston;
- Podemos instalar o pacote direto do terminal do Julia;
- Como é tudo novo, e existem muitas dependências, não há garantia que os pacotes estão funcionando.
- Normalmente esses problemas são reportados e alguma solução (às vezes temporária) é apresentada.

```
julia> Pkg.update()
julia> Pkg.add("PyPlot")
julia> using PyPlot
julia> plot([1,2,3,4,5],[3,2,5,1,4])
julia> plot([1,2,3,4,5],[3,2,5,1,4],".")
```

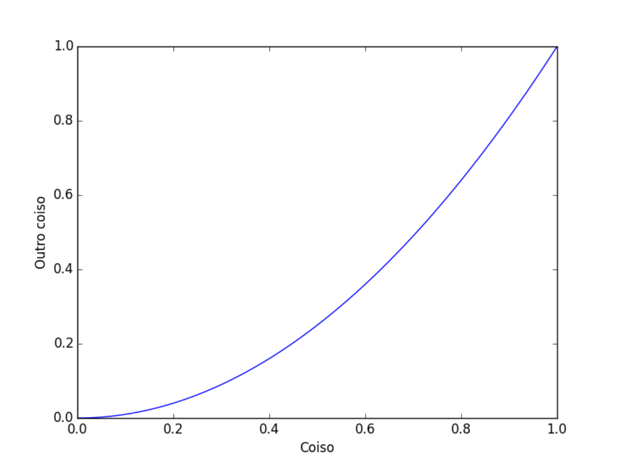


```
julia> plot([1,2,3,4,5],[3,2,5,1,4])
julia> axis([0,6,0,6])
```

Logo

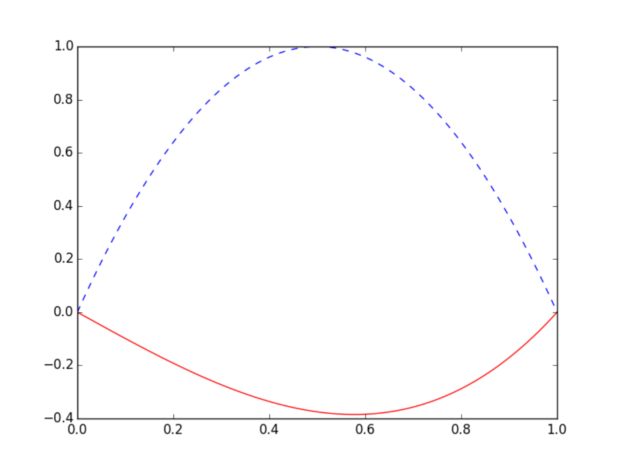


```
julia> plot([1,2,3,4,5],[3,2,5,1,4],".")
julia> axis([0,6,0,6])
```



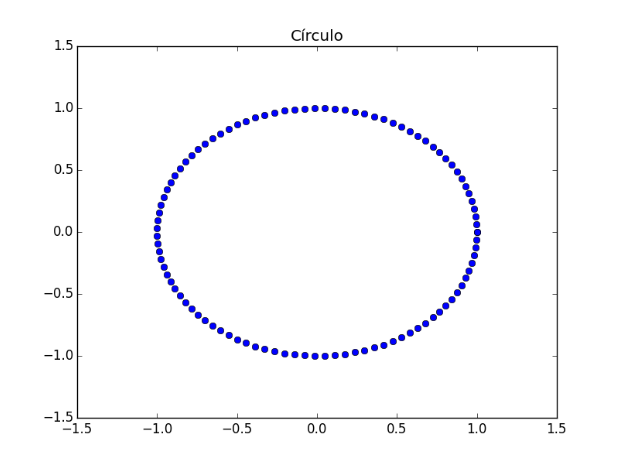
```
julia> x = linspace(0,1,100);
julia> plot(x,x.^2)
julia> xlabel("Coiso"); ylabel("Outro coiso")
```

Logo



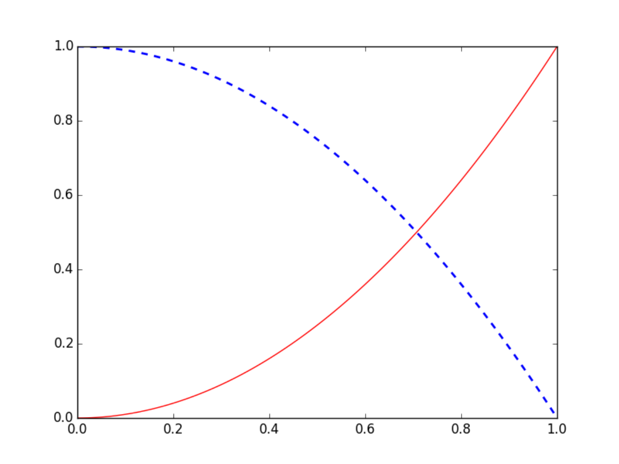
```
julia> x = linspace(0,1,100);
julia> plot(x, x.^3-x, "r", x, 4*x.*(1-x), "b--")
```

Logo



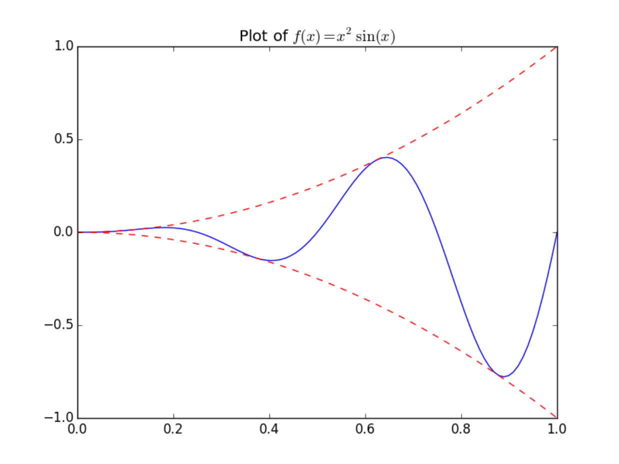
```
julia> x = linspace(0,1,100);
julia> plot(cos(2*pi*x), sin(2*pi*x), "o")
julia> title("Círculo")
```

Logo



```
julia> x = linspace(0,1,100);
julia> plot(x, x.^2, "r")
julia> plot(x, 1-x.^2, color="blue", linewidth=2.0, linestyle="--")
```

Logo



```
julia> x = linspace(0,1,100);
julia> plot(x, x.^2.*sin(4*pi*x))
julia> plot(x, x.^2, "r-", x, -x.^2, "r-")
julia> title(L"Plot of $f(x) = x^2\sin(x)$")
```

Obrigado

Esta apresentação está licenciada com uma Licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional.