Regressão logística multinomial com taxa de aprendizado dinâmica

Utilizando o método da bisecção

In [1]:

```
using Revise
using CSV, DataFrames, Plots, Distributions, LinearAlgebra
```

In [2]:

```
include("../src/reglog.jl")
```

Out[2]:

```
Main.RegLog
```

In [3]:

```
using .RegLog
```

Dados

In [4]:

```
X, Y = readiris("../data/raw/iris.m")
X = hcat(X, ones(size(X)[1]))
Yₘ = preparey(Y)
```

Out[4]:

```
150×3 Matrix{Float64}:
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 1.0  0.0  0.0
 ⋮
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
 0.0  0.0  1.0
```

In [5]:

```
n, 𝓓 = size(X)  # numero de instancias e features
k = size(Yₘ)[2]  # numero de classes
```

Out[5]:

```
3
```

Regressão multinomial sem e com bisseção

A implementação direta da softmax pode ser dada pelo algoritmo abaixo:

In [6]:

```
function cross_entropy(Yₘ, Ŷ)
    -sum(Yₘ .* log.(Ŷ))
end
```

Out[6]:

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

In [7]:

```
function simple_softmaxregression(X, Y)
    θ = rand(k, 𝒟)        # inicializa a matriz de coeficientes
    Ŷ = softmax(X * θ')  # calcula as probabilidades de cada classe
    ∇ = (Ŷ - Yₘ)' * X    # calcula o vetor gradiente
    ∇ₙ = ∇/norm(∇)        # normaliza o gradiente para um passo unitário
    itmax = 1000          # numero maximo de iterações
    η = 3e-1              # taxa de aprendizado fixa
    ϵ = 1e-2
    it = 0

    losses = Vector()

    while (norm(∇) > ϵ) & (it < itmax)
        it += 1

        Ŷ = softmax(X * θ')
        ∇ = (Ŷ - Yₘ)' * X
        ∇ₙ = ∇/norm(∇)
        θ = θ - η * ∇ₙ

        loss = cross_entropy(Yₘ, Ŷ)
        push!(losses, loss)
    end

    return θ, losses
end
```

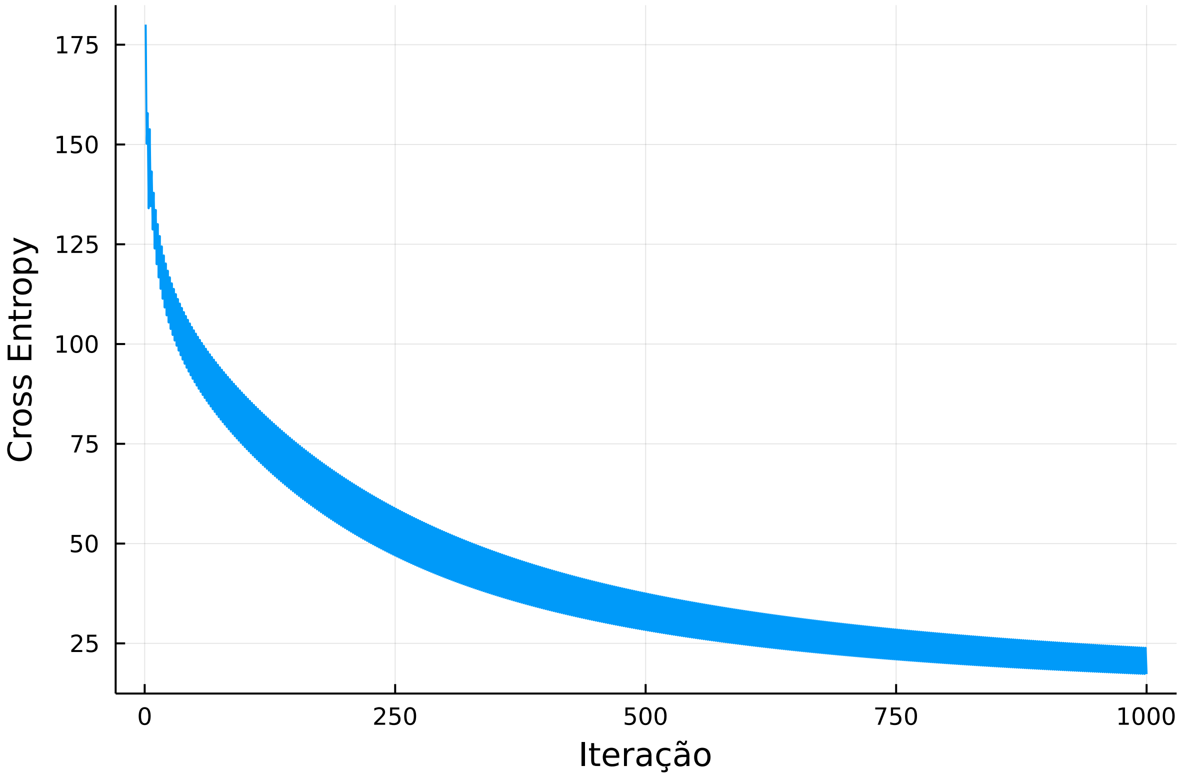
Out[7]:

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

In [8]:

```
simple_θ, simple_loss = simple_softmaxregression(X, Yₘ)
plot(simple_loss, xlabel="Iteração", ylabel="Cross Entropy", legend=false)
```

Out[8]:



No entanto, para otimizar a convergência do algorítmo, podemos utilizar o método de bissecção baseado em algumas premissas e proposições. \

Primeiro inicializaremos novamente os parâmetros da softmax:

In [9]:

```
θ = rand(k, 𝒟)        # inicializa a matriz de coeficientes
Ŷ = softmax(X * θ')  # calcula as probabilidades de cada classe
∇ = (Ŷ - Yₘ)' * X    # calcula o vetor gradiente
∇ₙ = ∇/norm(∇)        # normaliza o gradiente para um passo unitário
itmax = 1000          # numero maximo de iterações
η = 3e-1              # taxa de aprendizado fixa
ϵ = 1e-2
it = 0
```

Out[9]:

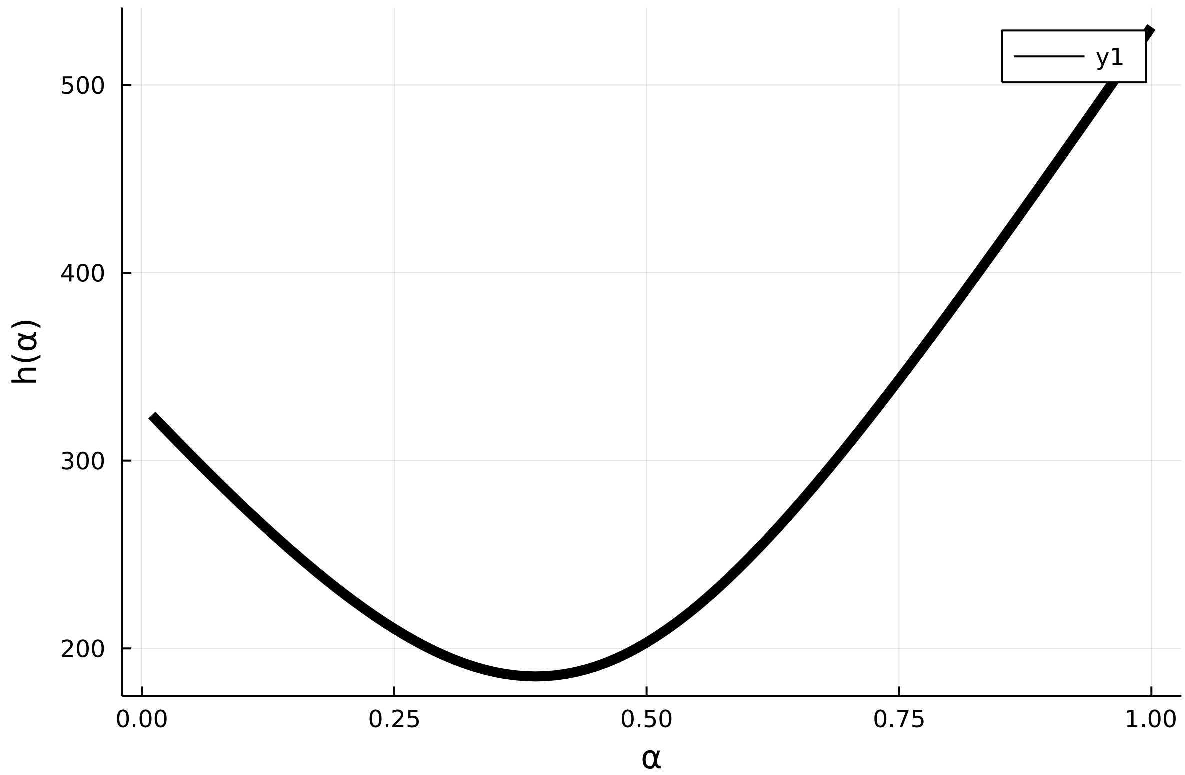
0

Podemos notar após esta primeira iteração que h(a) é uma função convexa:

In [10]:

```
alphas = cumsum(zeros(100) .+ 0.01)
hs = Vector()
for a in alphas
    push!(hs, h(a, θ, ∇ₙ, X, Yₘ))
end
plot(alphas, hs, linewidth=5, color=:black, xlabel="α", ylabel="h(α)")
```

Out[10]:

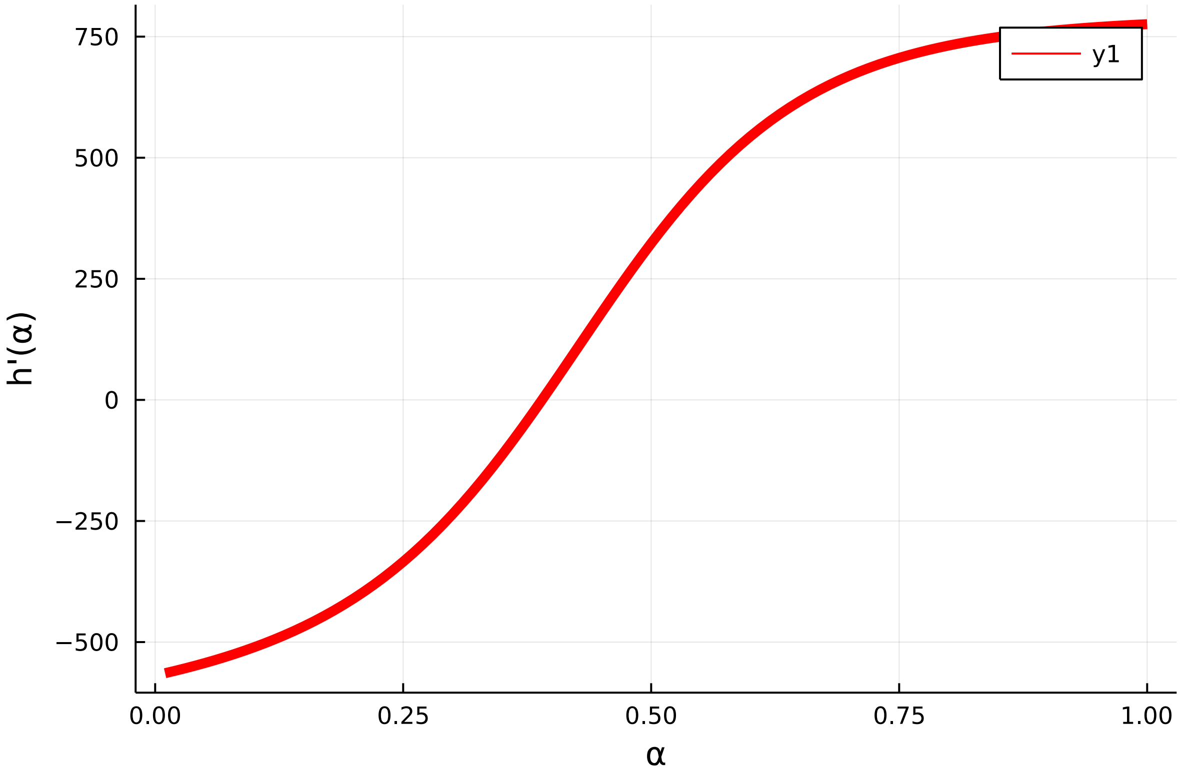


E sendo convexa, sabemos que a proposição h'(0) < 0 é verdadeira, como demonstrado:

In [11]:

```
hls = Vector()
for a in alphas
    push!(hls, ĥ(a, θ, ∇ₙ, X, Yₘ))
end
plot(alphas, hls, linewidth=5, color=:red, xlabel="α", ylabel="h'(α)")
```

Out[11]:



Sabendo disso, podemos aplicar o método da bisecção escolhendo:

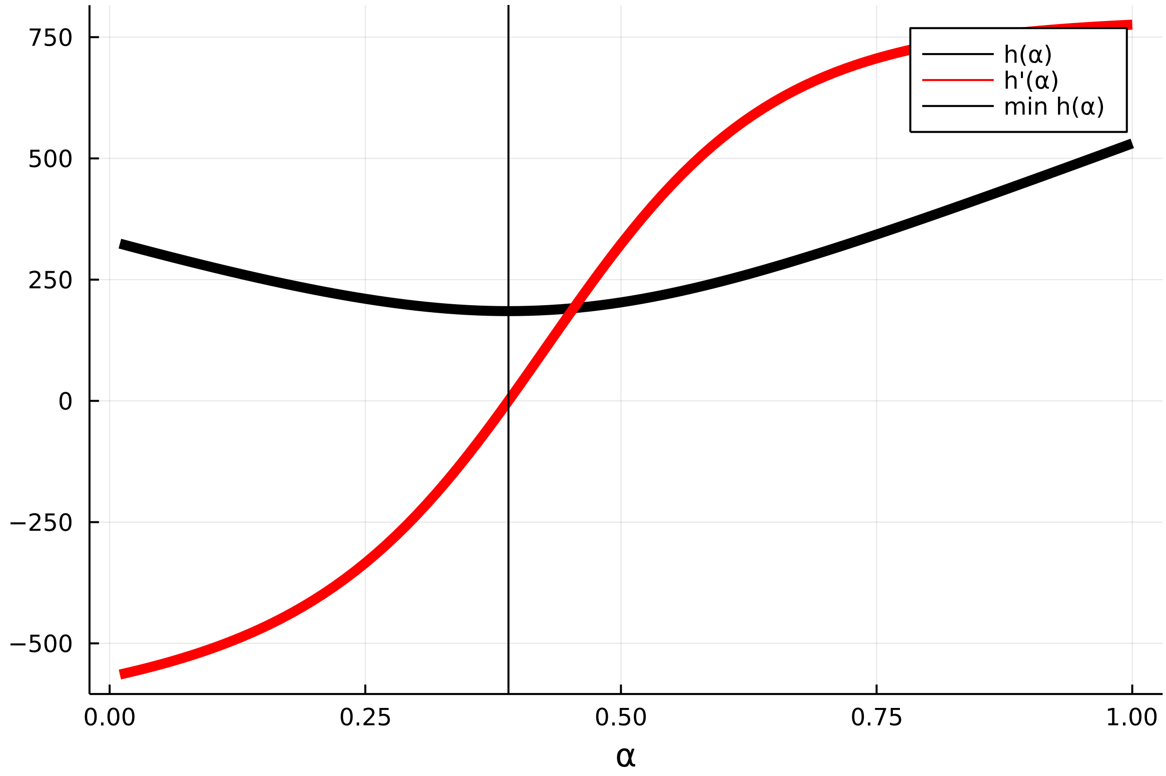
- αl = 0
- αu = o primeiro α aleatório cuja h'(α) > 0

Podemos ainda verificar que quando h'(a) tende a 0 o valor de h(a) tende ao mínimo global

In [12]:

```
plot(alphas, hs, linewidth=5, color=:black, xlabel="α", label="h(α)")
plot!(alphas, hls, linewidth=5, color=:red, xlabel="α", label="h'(α)")
vline!([alphas[argmin(hs)]], color=:black, label="min h(α)")
```

Out[12]:



Logo, basta adicionarmos o algoritmo de bissecção para encontrar um α, que minimize a loss function na iteração atual.

In [13]:

```
function bisection_softmaxregression(X, Y)
    θ = rand(k, 𝒟)        # inicializa a matriz de coeficientes
    Ŷ = softmax(X * θ')  # calcula as probabilidades de cada classe
    ∇ = (Ŷ - Yₘ)' * X    # calcula o vetor gradiente
    ∇ₙ = ∇/norm(∇)        # normaliza o gradiente para um passo unitário
    itmax = 1000         # numero maximo de iterações
    ϵ = 1e-2
    it = 0

    losses = Vector()

    while (norm(∇) > ϵ) & (it < itmax)
        it += 1

        Ŷ = softmax(X * θ')
        ∇ = (Ŷ - Yₘ)' * X
        ∇ₙ = ∇/norm(∇)
        η = bisection(θ, ∇ₙ, X, Yₘ)
        θ = θ - η * ∇ₙ

        loss = cross_entropy(Yₘ, Ŷ)
        push!(losses, loss)
    end

    return θ, losses
end
```

Out[13]:

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

In [14]:

```
bisec_θ, bisec_loss = bisection_softmaxregression(X, Yₘ)
```

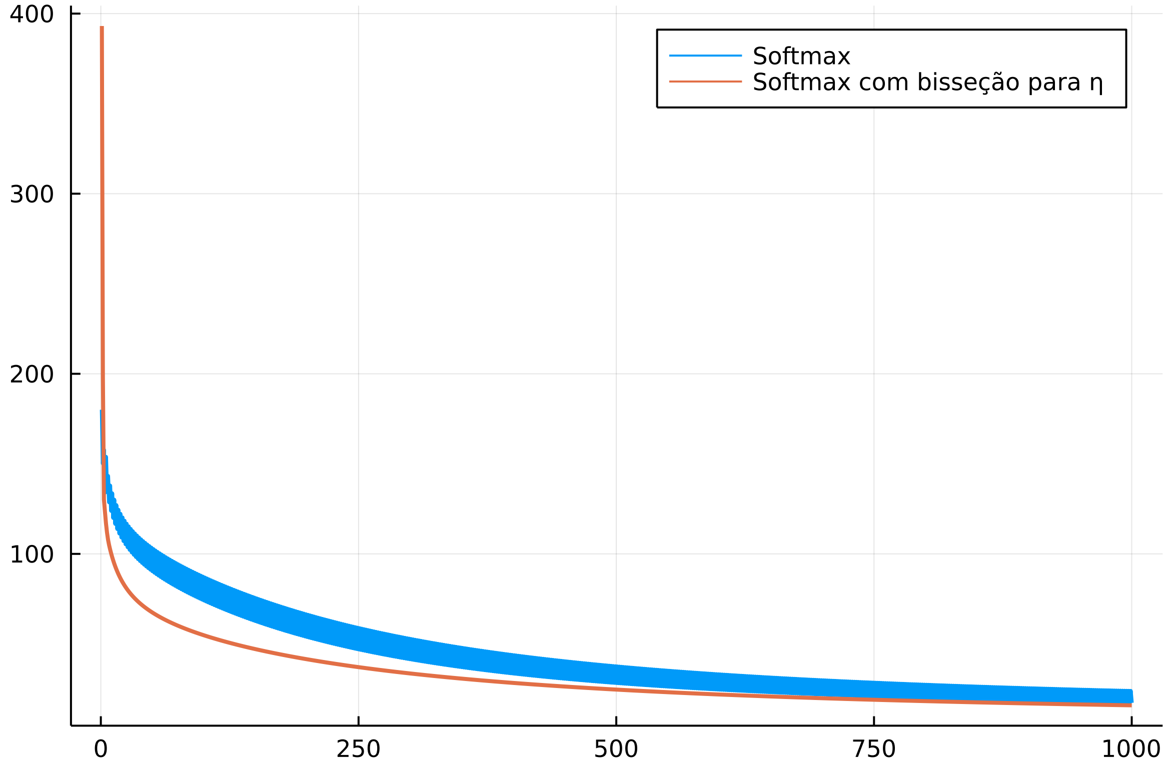
Out[14]:

```
([1.6160018149258872 2.645038273990014 … -1.4048683797617203 0.4632478
9008518613; 1.3341211127833377 0.22346302154821485 … -0.93763063702626
16 0.9923870839370718; -1.3912273071574415 -2.0347878181311763 … 3.677
487515982601 -0.5129117332595153], Any[393.10443907250794, 198.3104590
0514727, 130.15746556847995, 123.40042253663779, 116.85279098645037, 1
11.7802343354486, 107.84685706820117, 104.90332816924126, 102.46104670
787798, 100.26012823329481  …  16.00113189717527, 15.991547496416546,
15.981961417315883, 15.972413888045336, 15.962864683252977, 15.9533538
44738924, 15.943841324820493, 15.934366958230852, 15.924890921431297,
15.915452855310672])
```

In [17]:

```
plot(simple_loss, linewidth=2, label="Softmax")
plot!(bisec_loss, linewidth=2, label="Softmax com bisseção para η")
```

Out[17]:



In [ ]: