

# Regressões Não Lineares

Danilo A C Souto



## Programação Não Linear

Diferentemente dos sistemas lineares, a programação não linear é um sistema equações sobre um conjunto de variáveis cujos valores são desconhecidos e uma *Função Objetivo* a ser maximizada, ou minimizada, sendo esta não Linear.

Modelos lineares, refletem apenas aproximações dos modelos reais. Os fenômenos naturais são melhor representados por modelos não lineares.

- As equações precisam ser resolvidas em todas as áreas

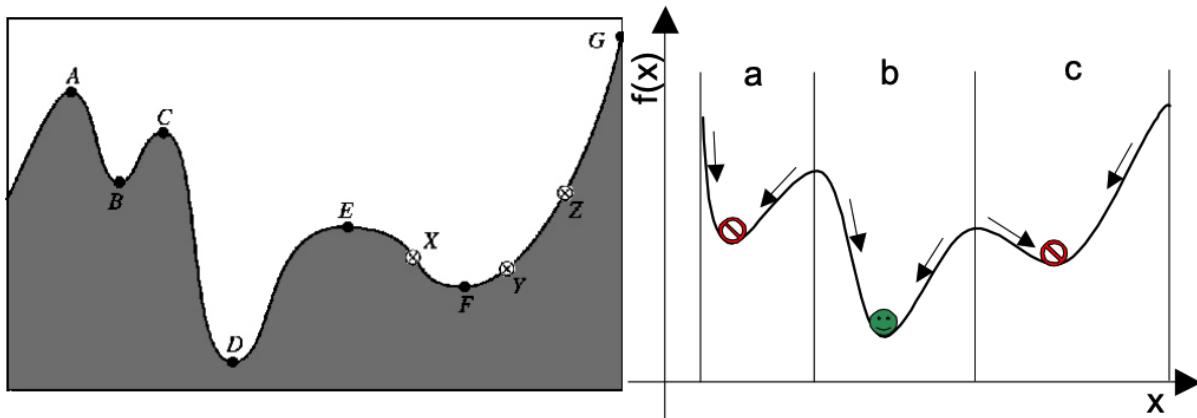
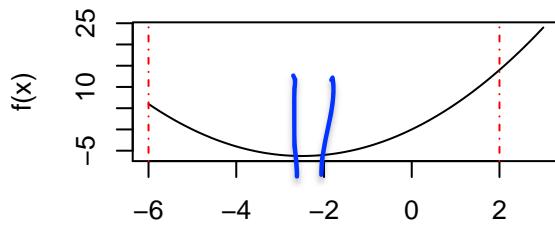
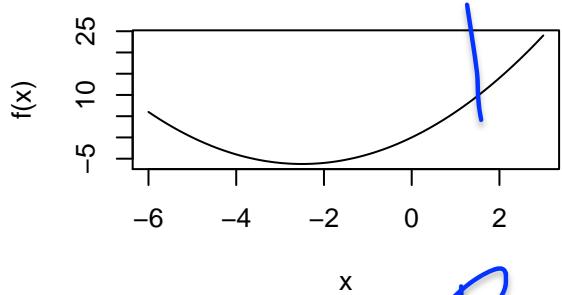


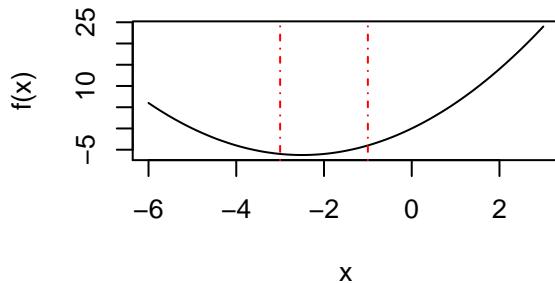
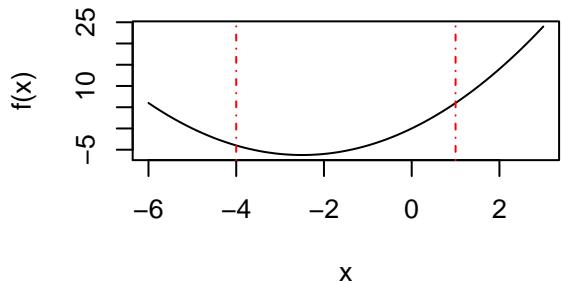
Figure 1: <https://www.ufjf.br/epd015/files/2010/06/ProgramacaoNaoLinear.pdf>

## Métodos de confinamento

Uma das formas é utilizar um método de confinamento. Ao utilizar um método de confinamento, definimos uma área para busca e esse intervalo é reduzido sucessivamente até uma precisão desejada.



*Poder computacional*



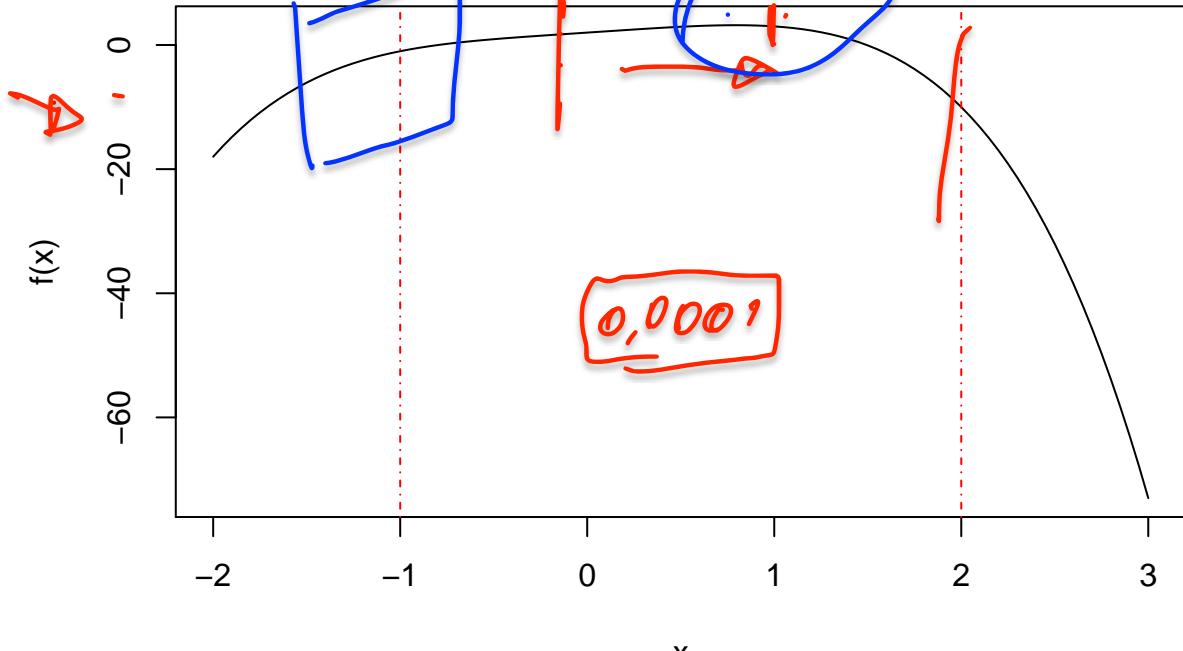
Outro exemplo é a função abaixo

*resolvendo*

$$f(x) = -x^4 + 2x + 2$$

Quando plotamos a equação temos a seguinte imagem:

```
f <- function(x){-x^4+2*x+2}
curve(expr = f, from = -2, to = 3)
abline(v = -1, col="red", lty=4)
abline(v = 2, col="red", lty=4)
```



Como ela possui apenas uma dimensão, neste caso dada por x, podemos utilizar a função do pacote **stats**, padrão no R, optimize(funcao, intervalo, tol, maximum)

Portanto para definirmos uma função para otimização temos que definir 4 itens:

1. Função a ser otimizada
2. Intervalo de pesquisa
3. Tolerância ou precisão desejada
4. Se a função deve ser maximizada ou minimizada

Para a otimização, ou melhor, maximização da função descrita acima  $f(x) = -x^4 + 2x + 2$  temos o seguinte código:

```
opt <- optimize(f, interval = c(-2, 2), tol = 0.0001, maximum = T)
```

```
## [1] -0.472136
## [1] 0.472136
## [1] 1.055728
## [1] 1.121517
## [1] 0.8634195
## [1] 0.7139625
## [1] 0.8983338
## [1] 0.9058533
## [1] 0.9087909
## [1] 0.9085694
## [1] 0.9085361
## [1] 0.9086028
## [1] 0.9085694
```

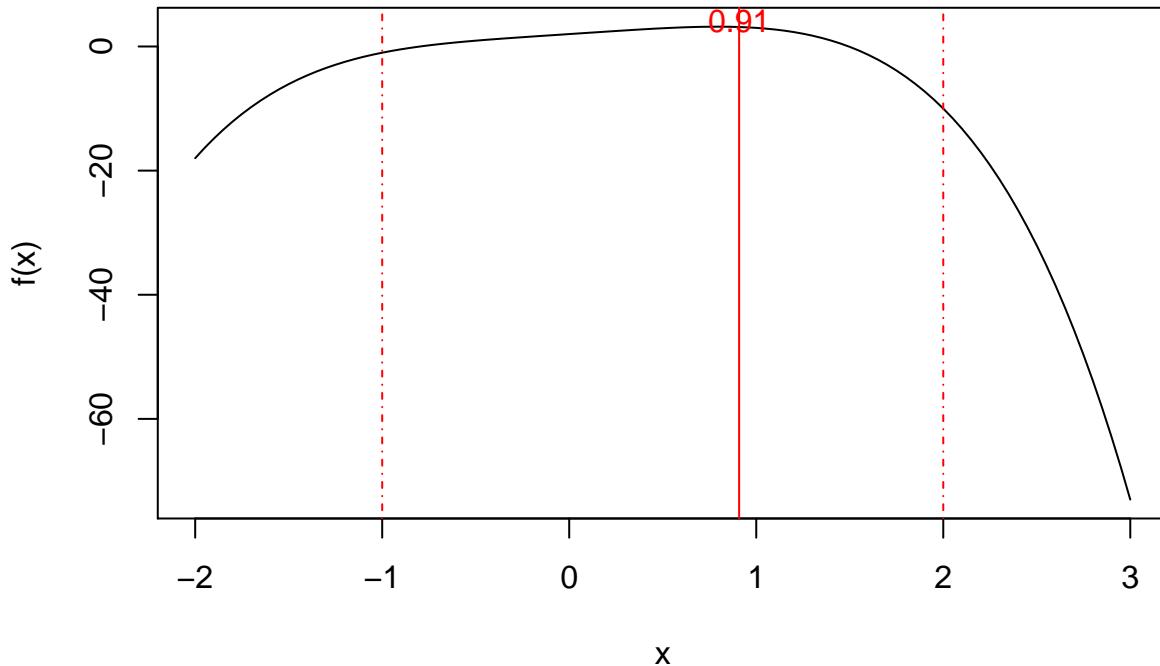
Note que a função faz busca no intervalo alterando os valores até alcançar a precisão desejada. Como saímos temos

```
opt

## $maximum
## [1] 0.9085694
##
## $objective
## [1] 4.044261
```

Caso o problema seja de minimização, no lugar do *maximum* teremos *minimum*

```
curve(expr = f, from = -2, to = 3)
abline(v = -1, col="red", lty=4)
abline(v = 2, col="red", lty=4)
abline(v = opt$maximum, col="red")
text(round(opt$maximum,2), col= "red", x = opt$maximum, y = opt$objective)
```



## Funções não lineares

Mas para as funções descritas acima é muito fácil encontrar o máximo. Basta fazer a derivada da função e igualar a zero!!!

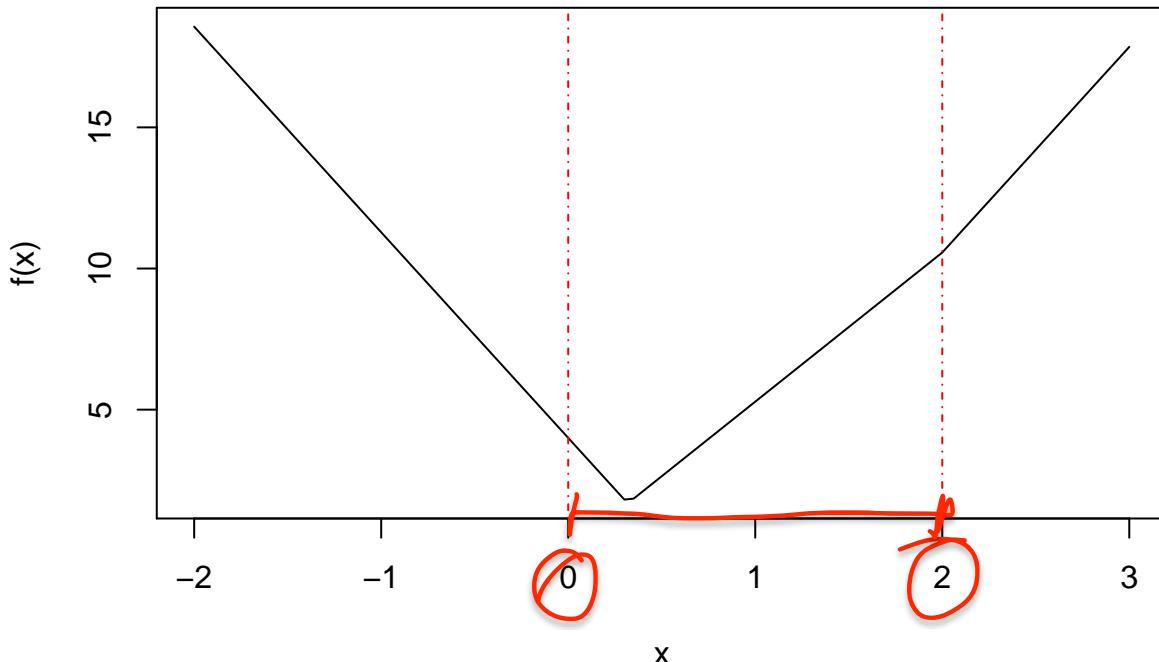
Correto?

Mas para funções não deriváveis como a função a seguir

$$f(x) = |x - 2| + 2|\pi \cdot x - 1|$$

Segue a imagem da função. Como pode ser observado não fica claro qual seria o mínimo dela. E perceba que neste caso temos um problema de minimização. Ou seja temos que encontrar o menor valor.

```
1. f <- function(x){abs(x-2) + 2*abs(pi*x-1)} .
curve(expr = f, from = -2, to =3)
abline(v = -0, col="red", lty=4)
abline(v = 2, col="red", lty=4)
```



Neste caso a computação nos ajuda a resolver o problema de otimização. Mas lembre-se de que devemos seguir os mesmos passos já descritos

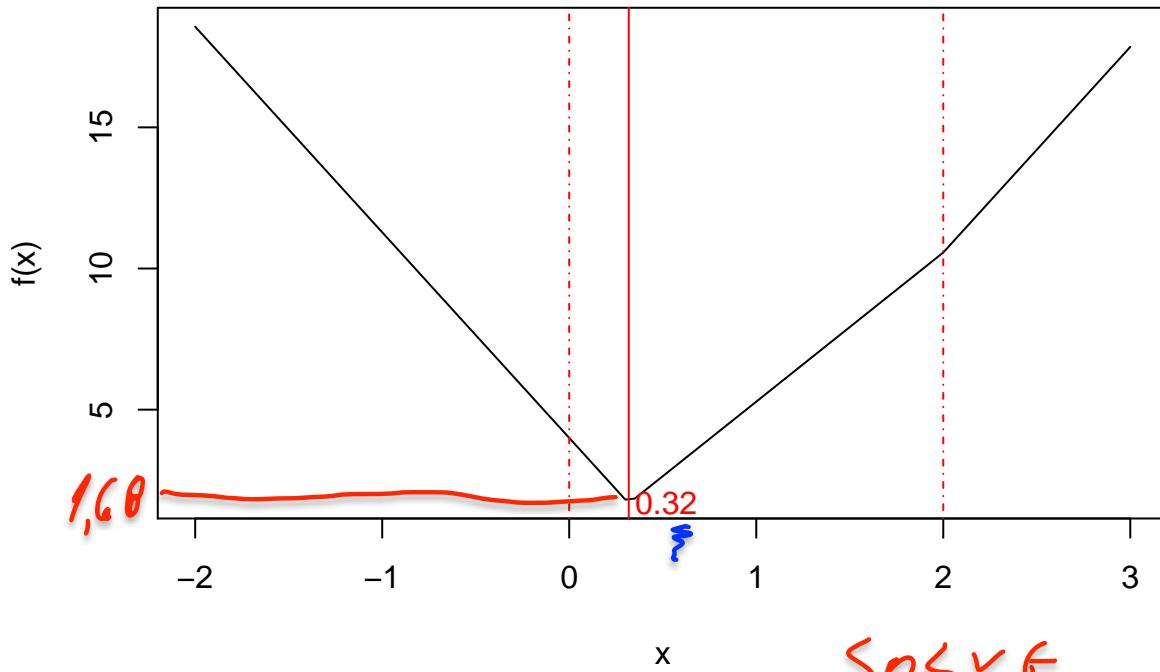
1. Definir a função a ser otimizada 1
2. Definir o intervalo de busca 2
3. Definir a precisão desejada 3
4. Definir se é um critério de maximização ou minimização 4

O código pode ser visto a seguir:

```
opt <- optimize(f, interval = c(0, 2), tol = 0.0001, maximum = FALSE)
curve(expr = f, from = -2, to = 3)
abline(v = 0, col="red", lty=4)
abline(v = 2, col="red", lty=4)
abline(v = opt$minimum, col="red")
text(round(opt$minimum, 2), col= "red", x = opt$minimum+0.2, y = opt$objective)
```

*OPT \$ minimum*

*OPT \$ objective*



Para este problema temos  $x = 0.3183126$  e  $y = 1.6817044$

## Multi Dimensões

As funções podem possuir muitas dimensões ou muitas variáveis. Assim os sistemas simplificados, como vistos anteriormente, não funcionam.

Para isso temos alguns pacotes específicos para este fim, dentre tantos temos o `optimx()`

Um conjunto de métodos não lineares estão no pacote `optimx`

```
require(optimx)
optimx(par, fn, gr=NULL, Hess=NULL, lower=inf,
upper=inf, method='', itnmax=NULL, ...)
```

- Métodos baseados em Gradiente Descendente
- Métodos baseados em Hessiana
- Nelder-Mead

### Hessiana

*Vantagem*

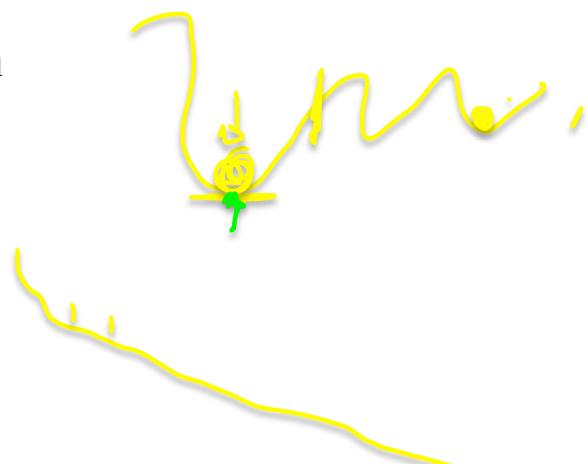
- Vantagem
  - Converge rapidamente
- Desvantagem
  - As derivadas parciais devem ser determinadas
  - A cada passo envolve a inversão de uma matriz
  - A estimativa inicial deve estar próxima da solução ideal

### Gradiente Descendente

- Vantagem
  - Não precisa calcular a matriz inversa
  - Cada iteração é barata computacionalmente
  - O tamanho do passo pode ser adaptativo

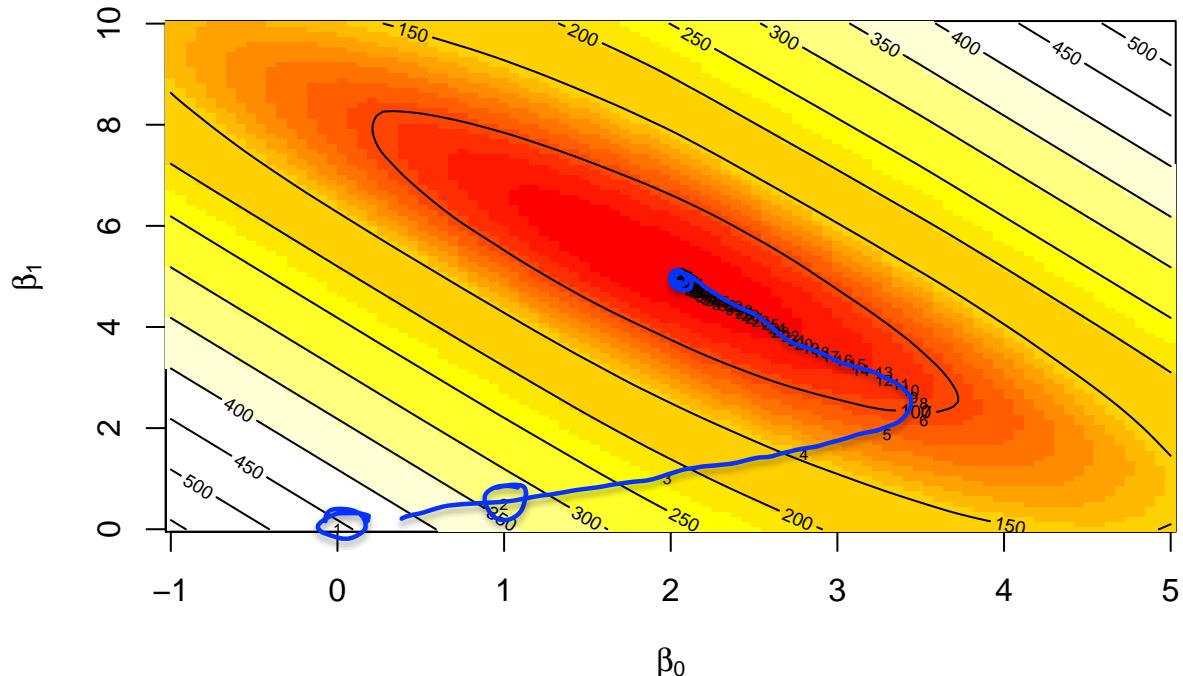
*Não Convergiu*

100

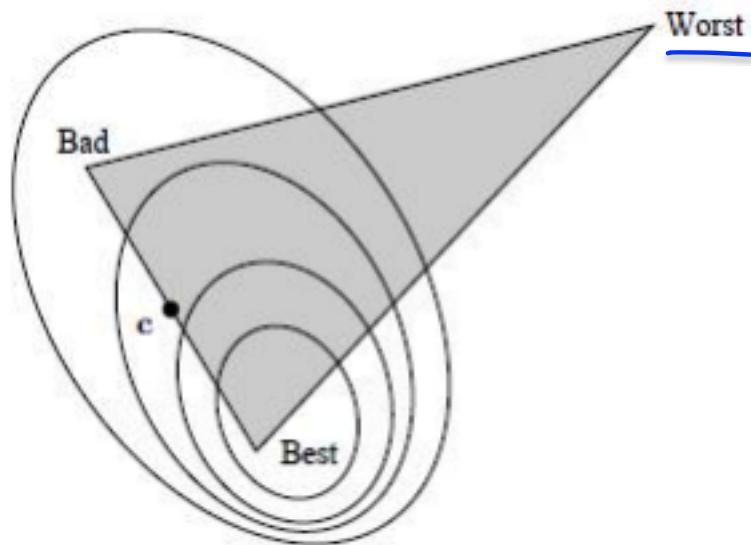


- Desvantagem
  - Precisa de *tunning*
  - Mais iterações que o método de Newton

## [1] 0.0000000 -0.2626263

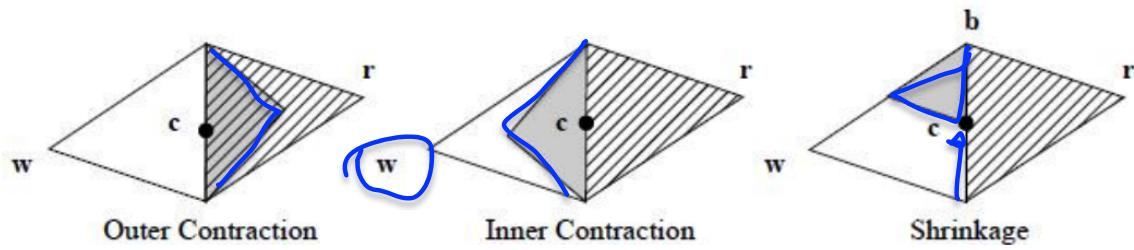
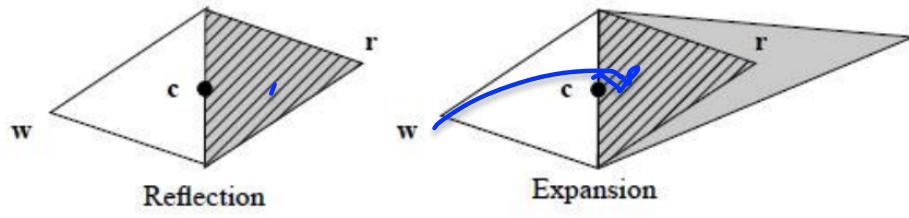


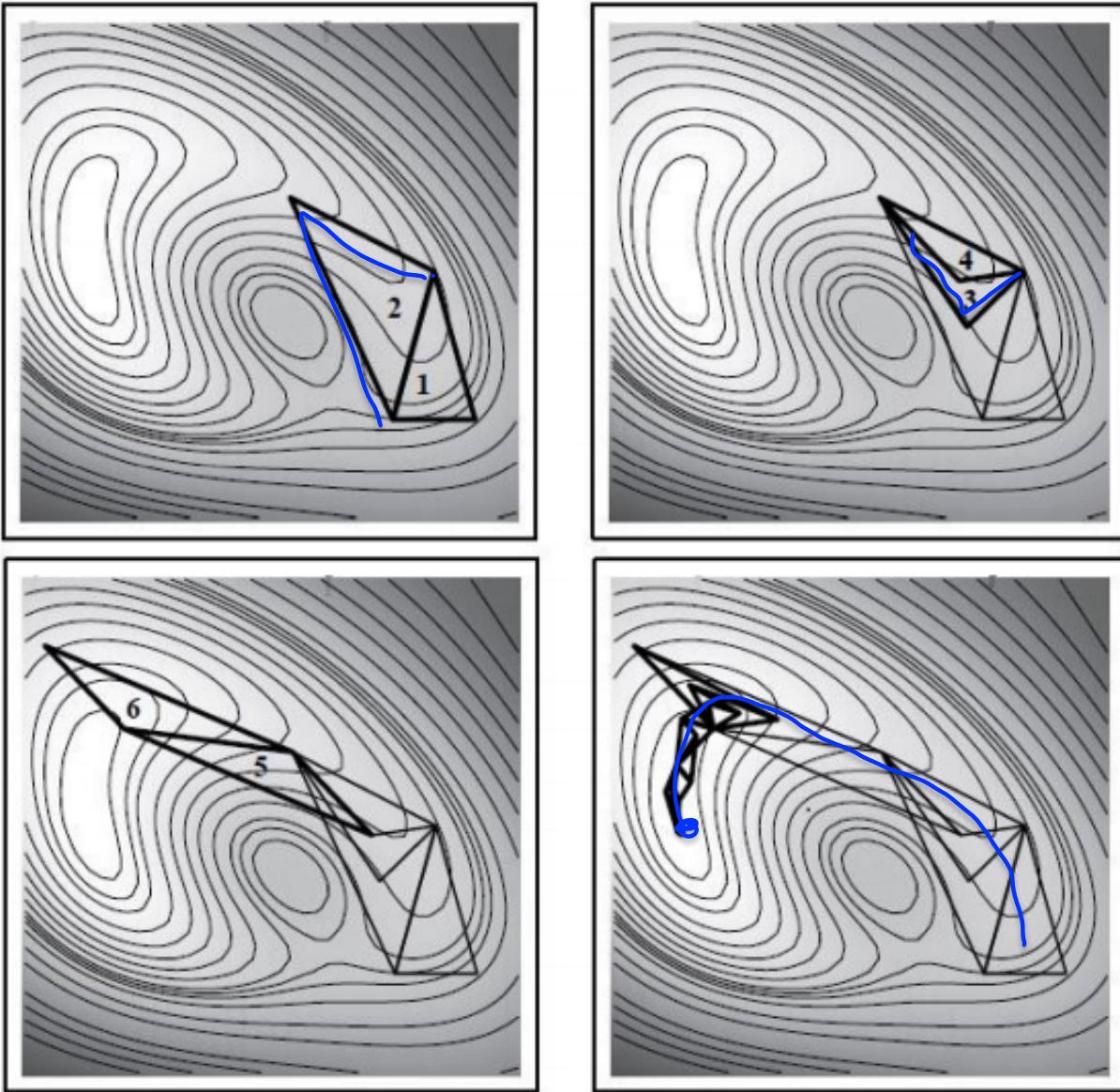
Nelder Mead



*Simplex*

↗ Aqui são os 5 possíveis caminhos que a função pode assumir a cada passo





- Vantagens
  - Não precisa de derivadas ↗
  - Eficiente para um número baixo de dimensões ↗
  - Pode ser utilizado para funções não contínuas
  - Resistente ao ruído ↘
- Desvantagens
  - Depende da natureza do problema
  - Pode convergir para soluções intermediárias que não são nem máximos nem mínimos
  - Baixa velocidade ↘

Givens, G. H., Hoeting, J. A. (2013). Computational Statistics. 2 ed. Wiley

## Exemplo de função multidimensional

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Note que para este caso temos mais de um parâmetro. Na definição da função temos uma entrada de um vetor como parâmetros.

```
→ install.packages("optimx")
library(optimx)

#definição da função
fn <- function(para){
  #transforma o vetor de parametros para em matriz
  matrix.A <- matrix(para, ncol=2)
  x <- matrix.A[,1]
  y <- matrix.A[,2]

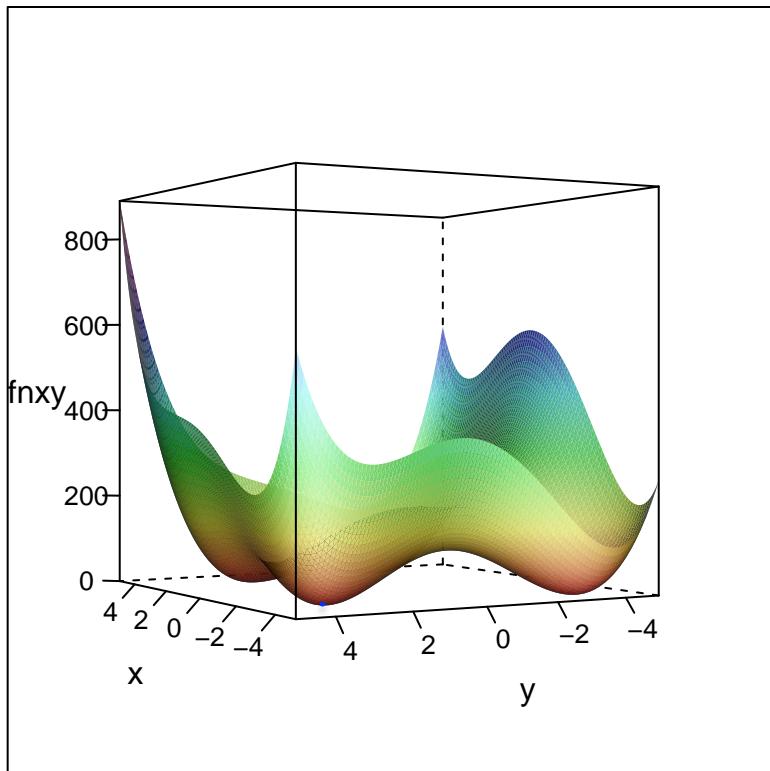
  #função a ser otimizada
  f.x <- (x^2+y-11)^2+(x+y^2-7)^2

  return(f.x)
}
```

Aqui a visualização da função. Como pode ser visto é uma função com vários pontos de mínimos locais.

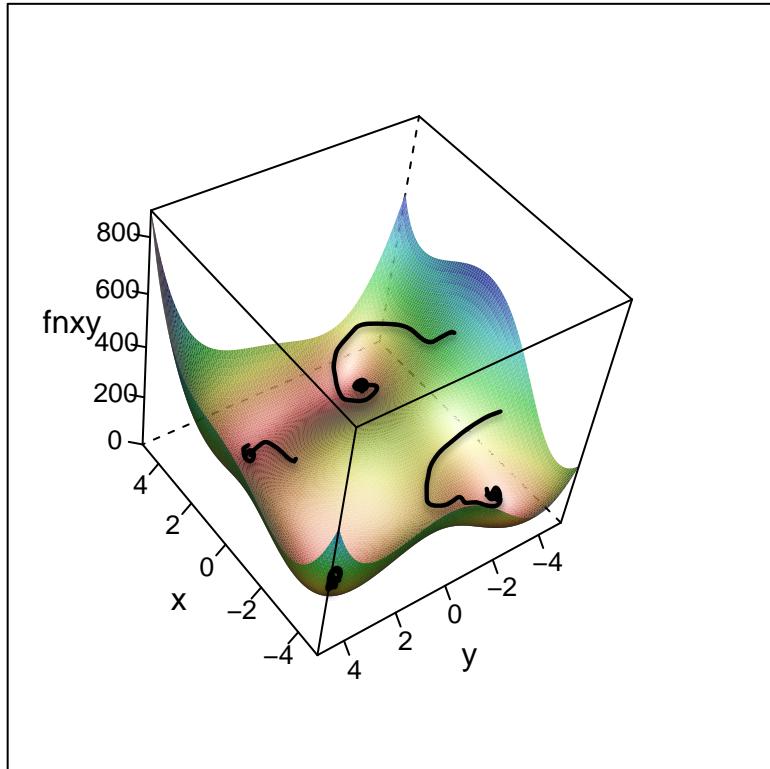
Portanto precisamos descobrir como encontrar o mínimo global

```
wireframe(fnxy ~ x*y, data = df, shade = TRUE, drape=FALSE,
scales = list(arrows = FALSE),
screen = list(z=-240, x=-90))
```



De outro ângulo

```
wireframe(fnxy ~ x*y, data = df, shade = TRUE, drape=FALSE,
scales = list(arrows = FALSE),
screen = list(z=-240, x=-45, y=10))
```



Nelder Mead

#parametros iniciais  
par <- c(1,1)

→ optimx(par, fn, method = "Nelder-Mead")

2 sente varridas  
3 metodos de busca

## p1 p2 value fevals gevals niter convcode kkt1  
## Nelder-Mead 2.999995 2.000183 5.56163e-07 67 NA NA 0 FALSE  
## kkt2 xtime  
## Nelder-Mead TRUE 0.001

## Gradientes

```
#parametros iniciais  
par <- c(1,1)  
  
optimx(par, fn, method = "CG")  
  
##      p1 p2      value fevals gevals niter convcode kkt1 kkt2 xtime  
## CG  3  2  1.081231e-12    119     31     NA          0 TRUE TRUE 0.001
```

BFGS

```
#parametros iniciais  
par <- c(1,1)
```

```

optimx(par, fn, method = "BFGS")
##      p1 p2      value fevals gevals niter convcode kkt1 kkt2 xtime
## BFGS  3  2 1.354193e-12    32       11     NA          0 TRUE TRUE     0

```

Vimos aqui os valores encontrados para os parâmetros **p1** e **p2** como o número de vezes que a função foi avaliada em **fevals**. Outro resultado muito importante a ser avaliado é **convcode** pois ele indica os possíveis casos de sucesso ou falha na otimização. (0 para sucesso).

Em resumo, quanto aos métodos de otimização, temos

Dimensionality	One-di-dimensional	Multi-dimensional		
Category	Non-gradient based	CG Gradient based	Hessian based	Non-gradient based
Algorithms	Golden Section Search	Gradient descent methods	Newton and quasi-Newton methods	Golden Section Search, Nelder-Mead
Package	stats		optimx	
Functions	optimize()	CG	BFGS L-BFGS-B	Nelder-Mead

## Régressões Não Lineares

"Modelos não lineares (MNL) usualmente são sustentados por alguma informação sobre a relação entre Y e x. Tal informação está vinculada à diferentes graus de conhecimento como

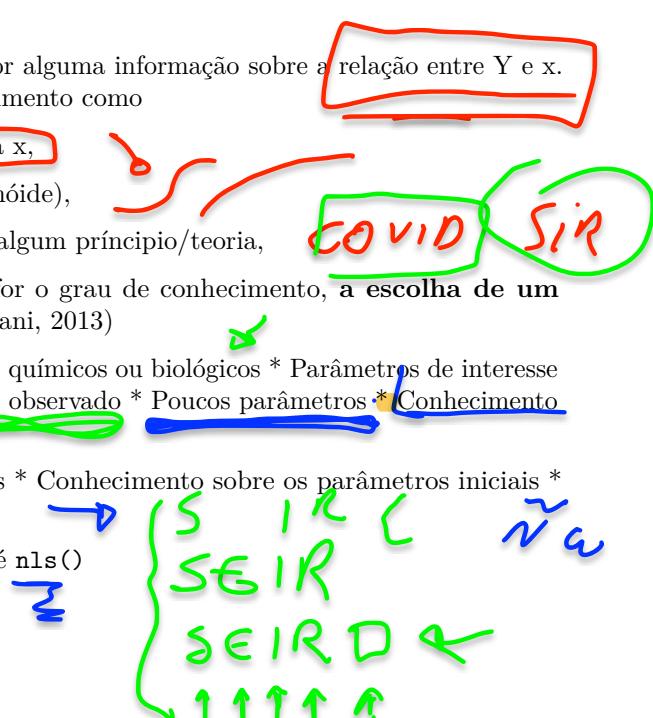
- 1) uma análise de um diagrama de dispersão de y contra x,
- 2) restrições de forma da função (ser monótona, ser sigmóide),
- 3) a solução de uma equação diferencial sustentada por algum princípio/teoria,
- 4) ou a interpretação dos seus parâmetros. Seja qual for o grau de conhecimento, a escolha de um modelo não linear raramente é empírica." (Zeviani, 2013)

Possui algumas vantagens: \* sustentação em princípios físicos, químicos ou biológicos \* Parâmetros de interesse do pesquisador \* Podem ser feitas previsões fora do domínio observado \* Poucos parâmetros \* Conhecimento do pesquisador

Mas também tem desvantagens \* Procedimentos iterativos \* Conhecimento sobre os parâmetros iniciais \* Exigem conhecimento do pesquisador

A função no R básica para trabalhar modelos não lineares é **nls()**

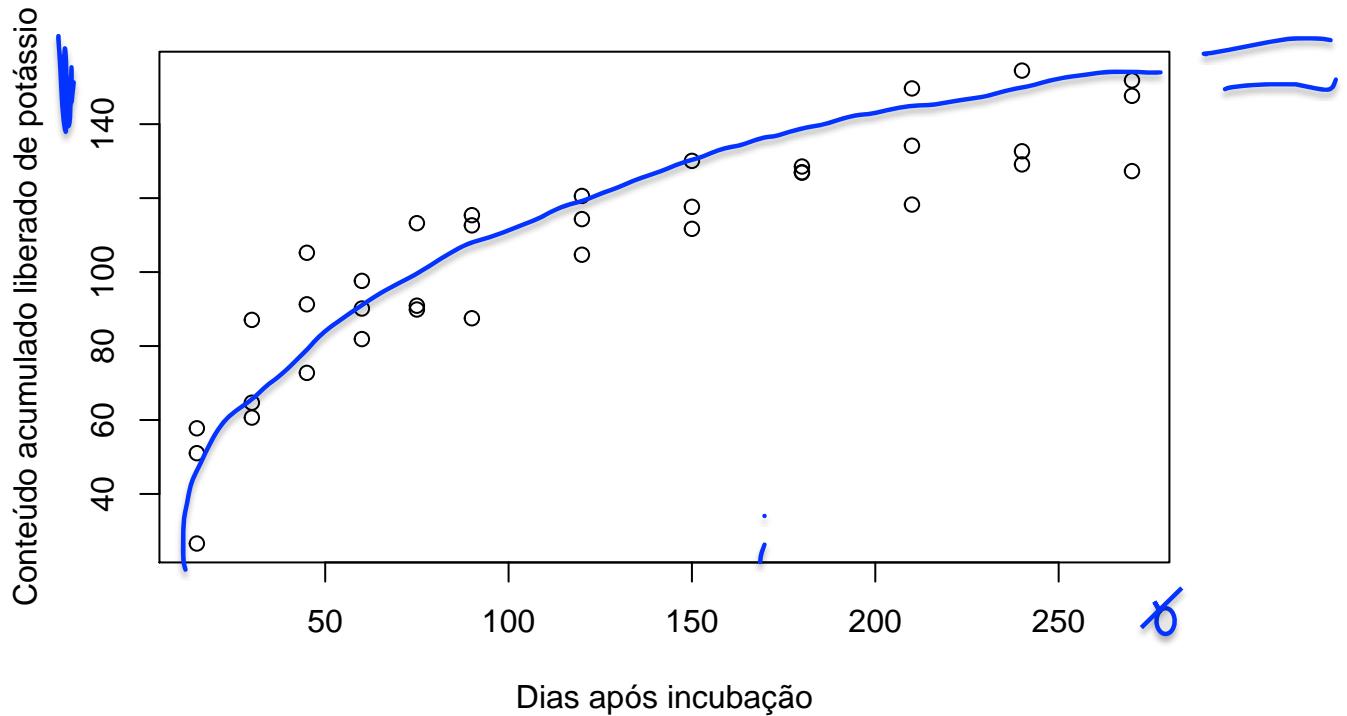
1. Análise



Exemplo: Quantidade de potássio liberado por esterco Zeviani, 2013)

```
da <- data.frame(y=c(51.03, 57.76, 26.60, 60.65, 87.07, 64.67, 91.28, 105.22,
72.74, 81.88, 97.62, 90.14, 89.88, 113.22, 90.91, 115.39,
112.63, 87.51, 104.69, 120.58, 114.32, 130.07, 117.65,
111.69, 128.54, 126.88, 127.00, 134.17, 149.66, 118.25,
132.67, 154.48, 129.11, 151.83, 147.66, 127.30),
x=rep(c(15, 30, 45, 60, 75, 90, 120, 150, 180, 210, 240, 270), each=3),
r=rep(1:3, 12))
```

```
plot(y~x, data=da, xlab="Dias após incubação", ylab="Conteúdo acumulado liberado de potássio")
```



Deve-se considerar uma função crescente a partir da origem (Michalis-Menten)

$$f(x, \theta) = \frac{\theta_a x}{\theta_v + x}$$

Como a fórmula é de conhecimento do pesquisador, temos que  $\theta_a$  é a assíntota e  $\theta_v$  é a meia vida. Desta maneira temos que a fórmula a ser ajustada é  $\text{formula}=y~A*x/(V+x)$  e podemos escolher o intervalo de busca entre 160 e 40.

```
n0 <- nls(formula=y~A*x/(V+x), data=da, start=list(A=160, V=40), trace=T)
```

```
## 4698.534 : 160 40
## 4547.358 : 157.59100 40.52048
## 4547.345 : 157.62960 40.57067
## 4547.345 : 157.63309 40.57431
summary(n0)

##
## Formula: y ~ A * x/(V + x)
##
```

```

## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## A    157.633     6.110  25.800 < 2e-16 ***
## V     40.574     5.658   7.171 2.71e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.56 on 34 degrees of freedom
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 7.907e-06

```

Aqui temos o intervalo de confiança

```

intervalo <- confint(n0)

## Waiting for profiling to be done...
intervalo

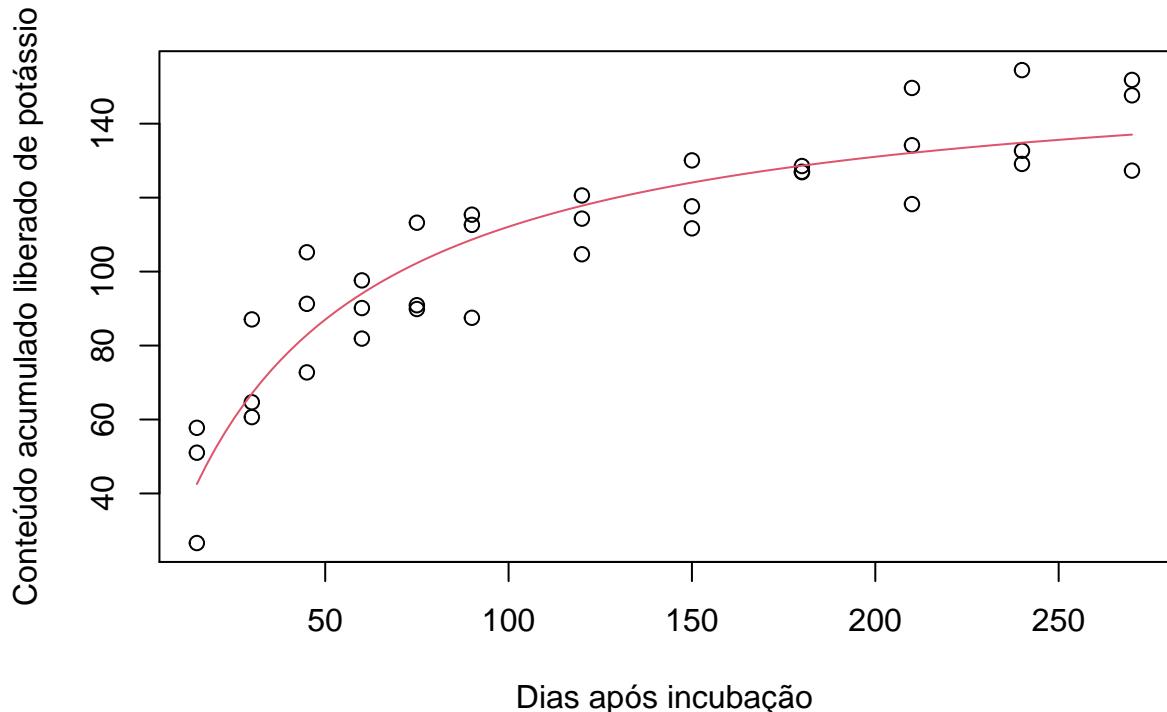
##          2.5%      97.5%
## A 145.75164 171.54842
## V 30.02655  54.21178

```

```

plot(y~x, data=da, xlab="Dias após incubação", ylab="Conteúdo acumulado liberado de potássio")
with(as.list(coef(n0)), curve(A*x/(V+x), add=TRUE, col=2) )

```



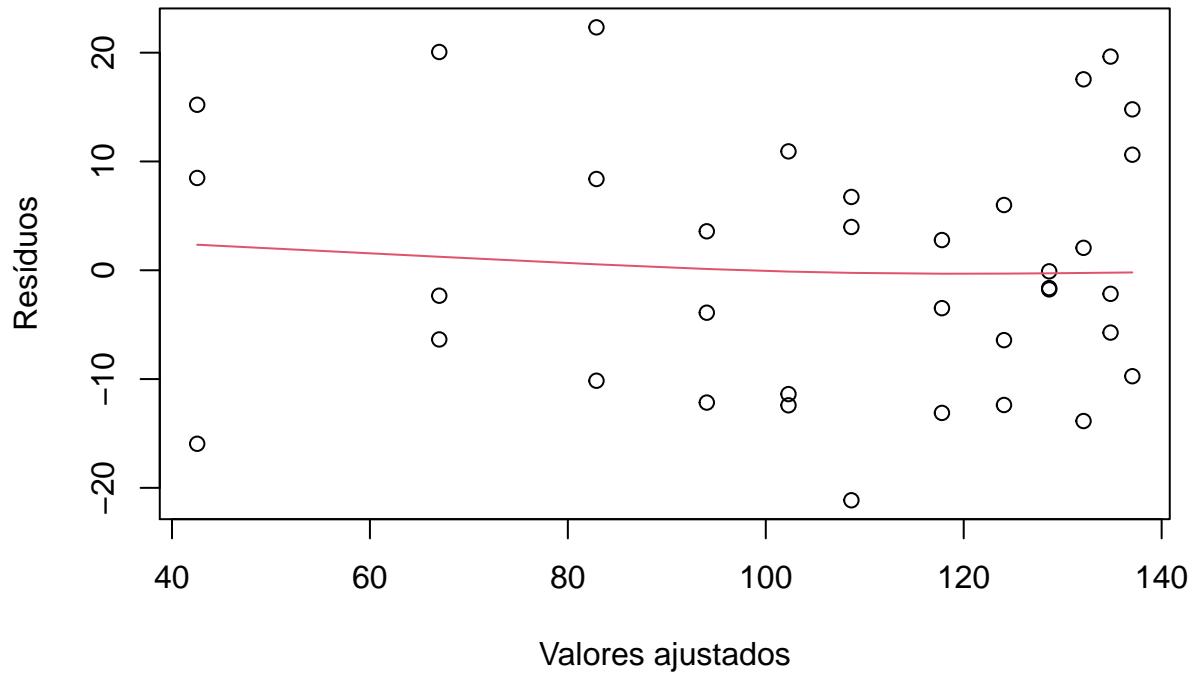
Quanto à análise de resíduos

```

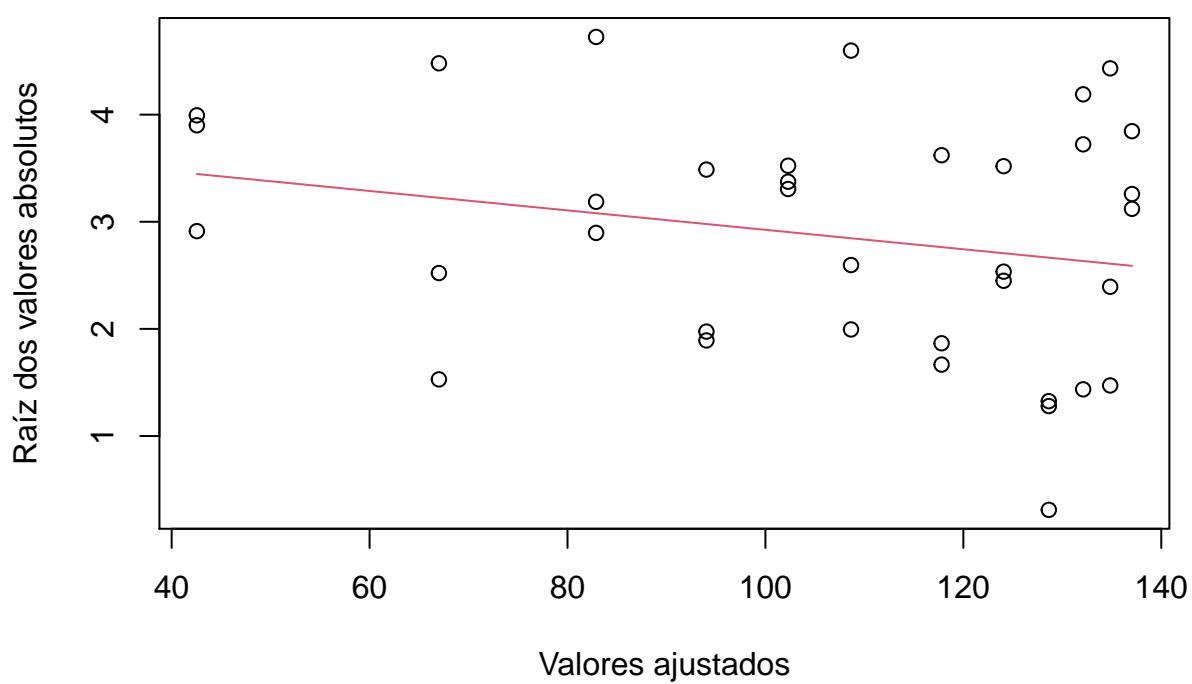
r <- residuals(n0)
f <- fitted(n0)
plot(r-f, xlab="Valores ajustados", ylab="Resíduos", main=expression("Adequação de " * eta(x,theta)))
lines(smooth.spline(r-f), col=2)

```

### Adequação de $\eta(x, \theta)$

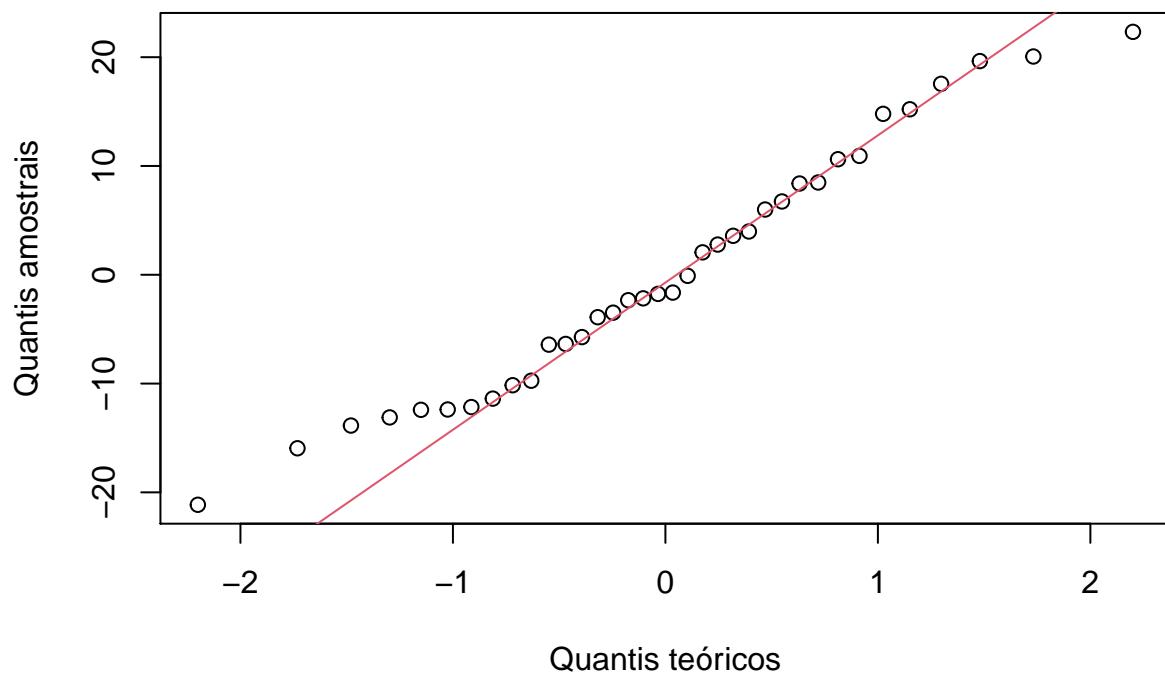


### Relação média–variância



```
qqnorm(r, xlab="Quantis teóricos", ylab="Quantis amostrais", main=list("Normalidade", font=1)); qqline(r)
```

Normalidade



```
layout(1)
```

Zeviani, Walmes Marques; Júnior, Paulo Justiniano Ribeiro; Bonat, Wagner Hugo. Modelos de regressão não linear, 2013.