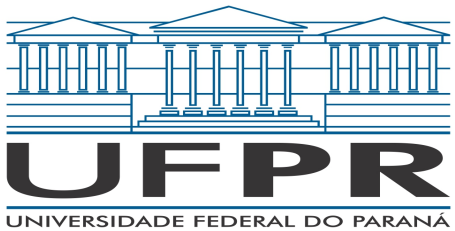
Solução de sistemas não-lineares

Prof. Wagner H. Bonat

Universidade Federal do Paraná
Departamento de Estatística
Laboratório de Estatística e Geoinformação



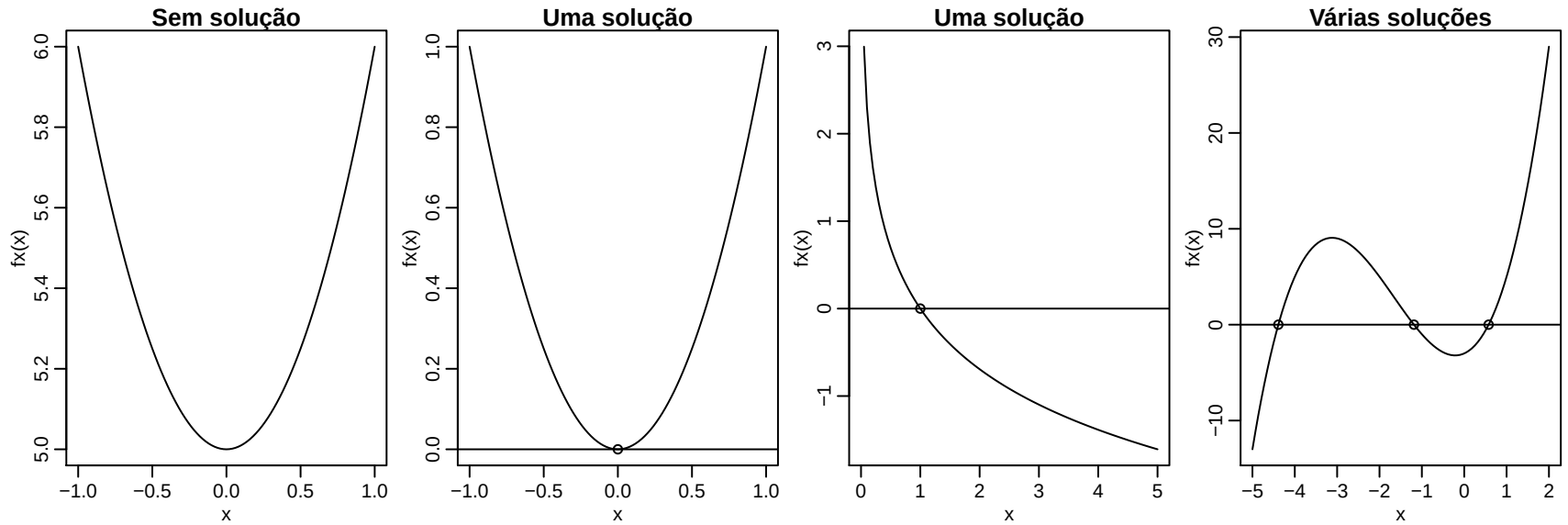
Sumário

1 Resolvendo equações não-lineares

2 Sistemas de equações

Equações não-lineares

- Equações precisam ser resolvidas frequentemente em todas as áreas da ciência.
- Equação de uma variável: $f(x) = 0$.
- A **solução** ou **raiz** é um valor numérico de x que satisfaz a equação.



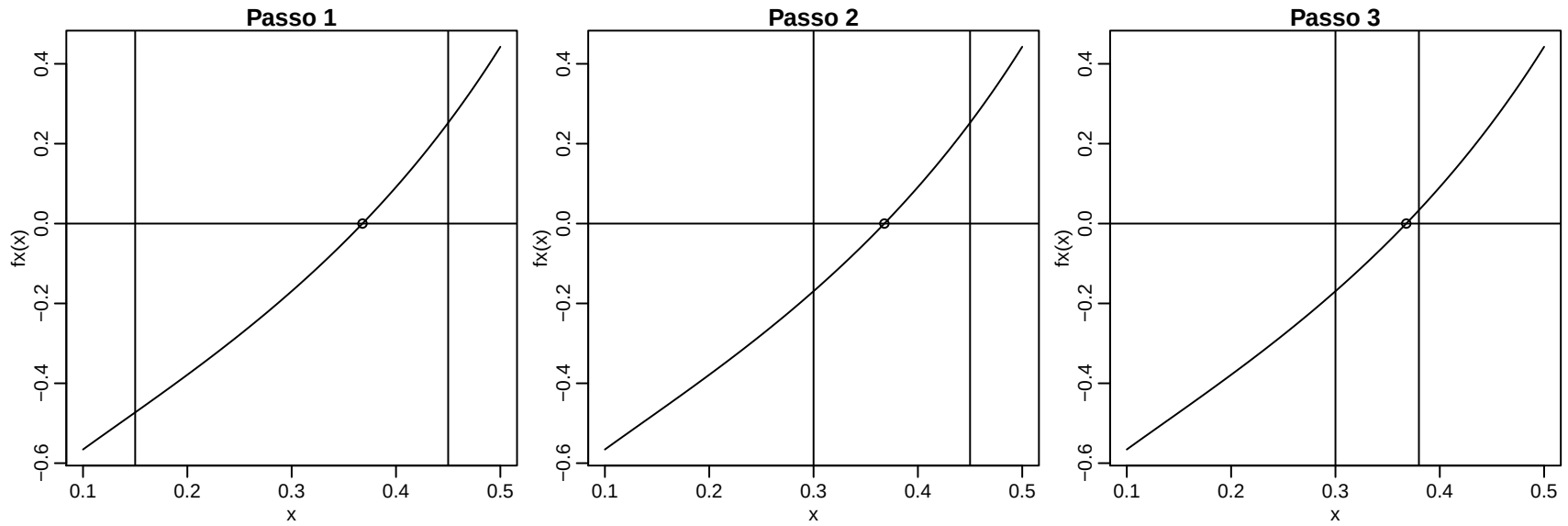
- A solução de uma equação do tipo $f(x) = 0$ é o ponto onde $f(x)$ cruza ou toca o eixo x.

Solução de equações não lineares

- Quando a equação é simples a **raiz** pode ser determinada analiticamente.
- Exemplo trivial $3x + 8 = 0 \rightarrow x = -\frac{8}{3}$.
- Em muitas situações é impossível determinar a **raiz** analiticamente.
- Exemplo não-trivial $8 - 4,5(x - \sin(x)) = 0 \rightarrow x = ?$
- Solução numérica de $f(x) = 0$ é um valor de x que satisfaz à equação de forma aproximada.
- Métodos numéricos para resolver equações são divididos em dois grupos:
  1. Métodos de confinamento;
  2. Métodos abertos.

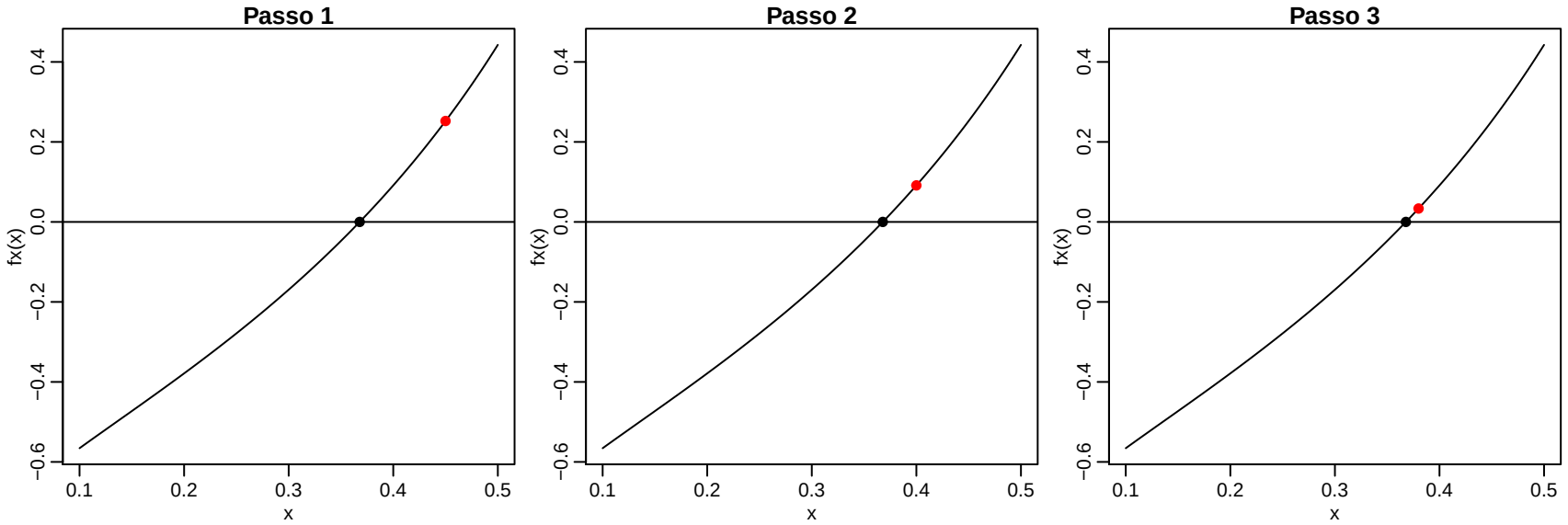
Métodos de confinamento

- Identifica-se um intervalo que possui a solução.
- Usando um esquema numérico, o tamanho do intervalo é reduzido sucessivamente até uma precisão desejada.



Métodos abertos

- Assume-se uma estimativa inicial.
- Tentativa inicial deve ser próxima a solução.
- Usando um esquema numérico a solução é melhorada.
- O processo para quando a precisão desejada é atingida.



Erros em soluções numéricas

- Soluções numéricas não são exatas.
- Critério para determinar se uma solução é suficientemente precisa.
- Seja x_{ts} a solução verdadeira e x_{ns} uma solução numérica.
- Quatro medidas podem ser consideradas para avaliar o erro:

1. Erro real $x_{ts} - x_{ns}$.
2. Tolerância em $f(x)$

$$|f(x_{ts}) - f(x_{ns})| = |0 - \epsilon| = |\epsilon|.$$

3. Tolerância na solução: Tolerância máxima da qual a solução numérica pode desviar da solução verdadeira. Útil em geral quando métodos de confinamento são usados

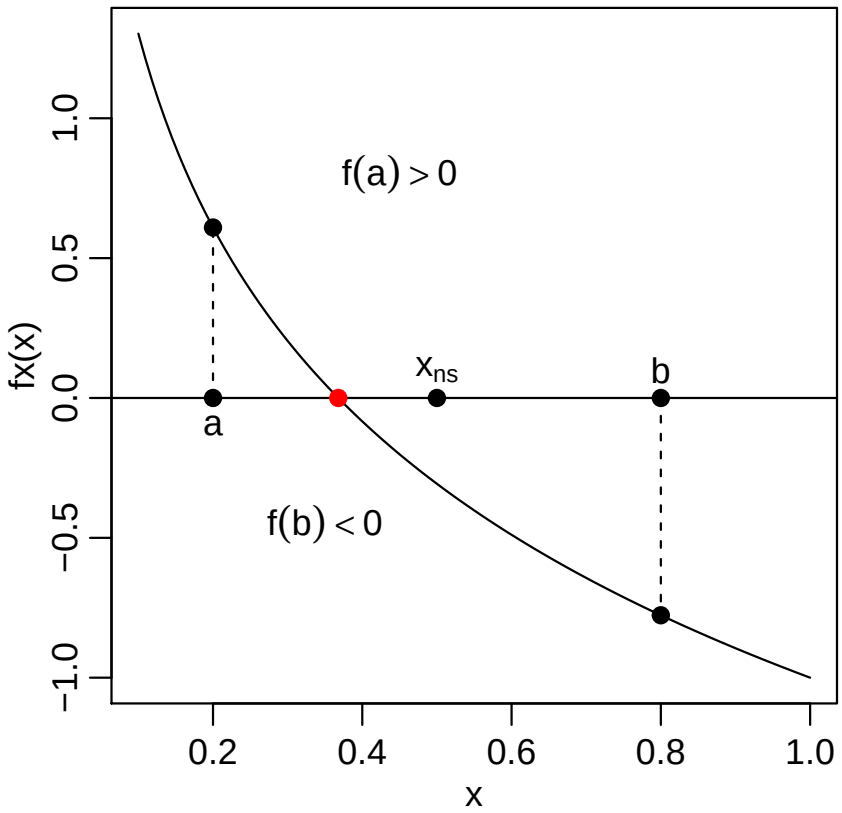
$$\left| \frac{b-a}{2} \right|.$$

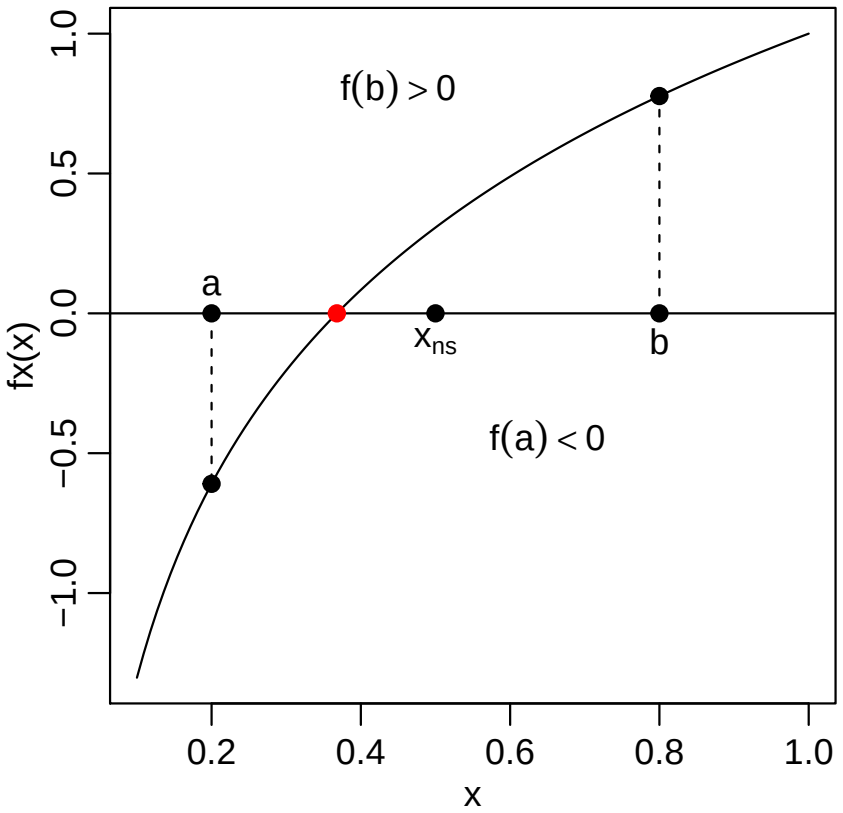
4. Erro relativo estimado:

$$\left| \frac{x_{ns}^{n} - x_{ns}^{n-1}}{x_{ns}^{n-1}} \right|.$$

Método da bisseção

- Método de confinamento.
- Sabe-se que dentro de um intervalo $[a, b]$, $f(x)$ é contínua e possui uma solução.
- Neste caso $f(x)$ tem sinais opostos nos pontos finais do intervalo.





Algoritmo: Método da bisseção

- Encontre $[a, b]$, tal que $f(a)f(b) < 0$.
- Calcule a primeira estimativa $x_{ns}^{(1)}$ usando $x_{ns}^{(1)} = \frac{a+b}{2}$.
- Determine se a solução exata está entre a e $x_{ns}^{(1)}$ ou entre $x_{ns}^{(1)}$ e b. Isso é feito verificando o sinal do produto $f(a)f(x_{ns}^{(1)})$:
  1. Se $f(a)f(x_{ns}^{(1)}) < 0$, a solução está entre a e $x_{ns}^{(1)}$.
  2. Se $f(a)f(x_{ns}^{(1)}) > 0$, a solução está entre $x_{ns}^{(1)}$ e b.
- Selecione o subintervalo que contém a solução e volte ao passo 2.
- Repita os passos 2 a 4 até que a tolerância especificada seja satisfeita.

Implementação **R**: Método da bisseção

```
bissecao <- function(fx, a, b, tol = 1e-04, max_iter = 100) {
  fa <- fx(a);  fb <- fx(b)
  if(fa*fb > 0) stop("Solução não está no intervalo")
  solucao <- c()
  sol <- (a + b)/2
  solucao[1] <- sol
  limites <- matrix(NA, ncol = 2, nrow = max_iter)
  for(i in 1:max_iter) {
    test <- fx(a)*fx(sol)
    if(test < 0) {
      solucao[i+1] <- (a + sol)/2
      b = sol
      }
    if(test > 0) {
      solucao[i+1] <- (b + sol)/2
      a = sol
      }
    if( abs( (b-a)/2) < tol) break
    sol = solucao[i+1]
    limites[i,] <- c(a,b)
  }
  out <- list("Tentativas" = solucao, "Limites" = limites, "Raiz" = solucao[i+1])
  return(out)}
```

Exemplo

- Encontrando a raiz de

$$D(\theta) = 2n\left[\log\left(\frac{\hat{\theta}}{\theta}\right) + \bar{y}(\theta - \hat{\theta})\right] \leq 3.84.$$

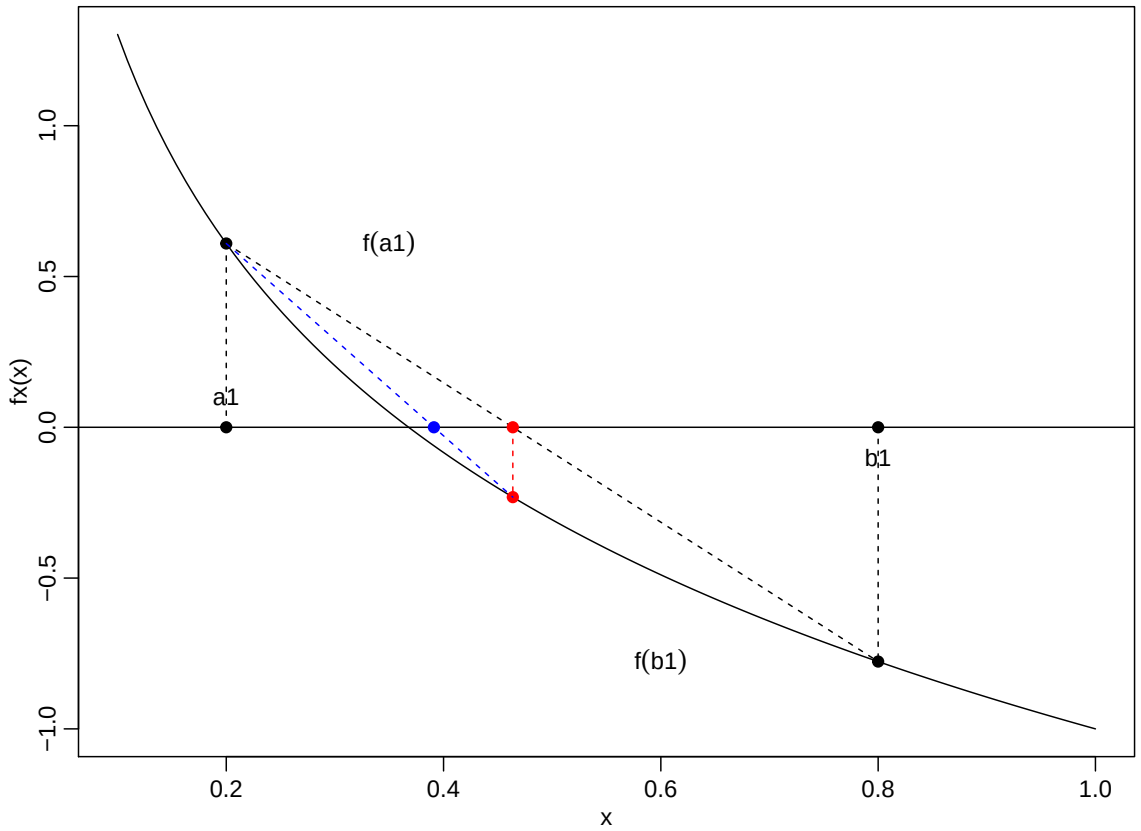
```
# Implementando a função
ftheta <- function(theta){
  dd <- 2*length(y)*(log(theta.hat/theta) + mean(y)*(theta - theta.hat))
  return(dd - 3.84)
}

# Resolvendo numericamente
set.seed(123)
y <- rexp(20, rate = 1)
theta.hat <- 1/mean(y)
Ic_min <- bissecao(fx = ftheta, a = 0, b = theta.hat)
Ic_max <- bissecao(fx = ftheta, a = theta.hat, b = 3)
# Solução aproximada
c(Ic_min$Raiz, Ic_max$Raiz)
```

```
# [1] 0.7684579 1.8545557
```

Método regula falsi

- Método de confinamento.
- Sabe-se que dentro de um intervalo $[a, b]$, $f(x)$ é contínua e possui uma solução.
- Ilustração.



Algoritmo: Método regula falsi

- Escolha os pontos a e b entre os quais existe uma solução.
- Calcule a primeira estimativa: $x^{(i)} = \frac{af(b) - bf(a)}{f(b) - f(a)}$.
- Determine se a solução está entre a e x^i, ou entre $x^{(i)}$ e b.
  1. Se $f(a)f(x^{(i)}) < 0$, a solução está entre a e $x^{(i)}$.
  2. Se $f(a)f(x^{(i)}) > 0$, a solução está entre $x^{(i)}$ e b.
- Selecione o subintervalo que contém a solução como o novo intervalo $[a, b]$ e volte ao passo 2.
- Repita passos 2 a 4 até convergência.

Implementação **R**: Método regula falsi

```
regula_falsi <- function(fx, a, b, tol = 1e-04, max_iter = 100) {
  fa <- fx(a);  fb <- fx(b)
  if(fa*fb > 0) stop("Solução não está no intervalo")
  solucao <- c()
  sol <- (a*fx(b) - b*fx(a))/(fx(b) - fx(a))
  solucao[1] <- sol
  limites <- matrix(NA, ncol = 2, nrow = max_iter)
  for(i in 1:max_iter) {
    test <- fx(a)*fx(sol)
    if(test < 0) {
      b = sol
      solucao[i+1] <- (a*fx(b) - b*fx(a))/(fx(b) - fx(a))
      }
    if(test > 0) {
      a = sol
      solucao[i+1] <- sol <- (a*fx(b) - b*fx(a))/(fx(b) - fx(a))
      }
    if( abs(solucao[i+1] - solucao[i]) < tol) break
    sol = solucao[i+1]
    limites[i,] <- c(a,b)
  }
  out <- list("Tentativas" = solucao, "Limites" = limites, "Raiz" = sol)
  return(out)}
```

Aplicação: Regula-falsi

- Encontre as raízes de

$$D(\theta) = 2n\left[\log\left(\frac{\hat{\theta}}{\theta}\right) + \bar{y}(\theta - \hat{\theta})\right] \leq 3.84.$$

```
# Resolvendo numericamente
Ic_min <- regula_falsi(fx = ftheta, a = 0.1, b = theta.hat)
Ic_max <- regula_falsi(fx = ftheta, a = theta.hat, b = 3)
# Solução aproximada
c(Ic_min$Raiz, Ic_max$Raiz)
```

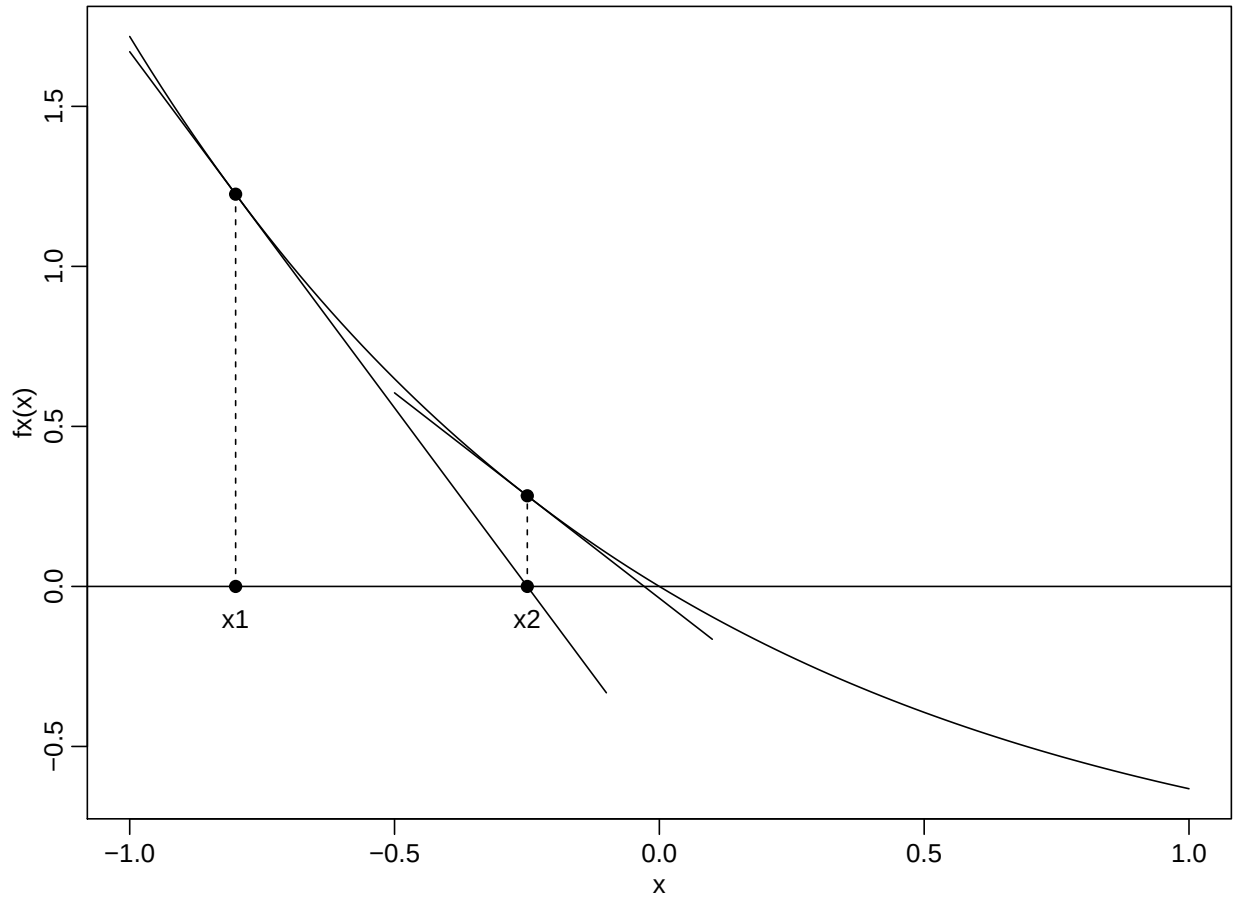
```
# [1] 0.7688934 1.8545456
```

Comentários: Métodos de confinamento

- Sempre convergem para uma resposta, desde que uma raiz esteja no intervalo.
- Podem falhar quando a função é tangente ao eixo x, não cruzando em $f(x) = 0$.
- Convergência é lenta em comparação com outros métodos.
- São difíceis de generalizar para sistemas de equações não-lineares.

Método de Newton

- Função deve ser contínua e diferenciável.
- Função deve possuir uma solução perto do ponto inicial.
- Ilustração:



Algoritmo: Método de Newton

- Escolha um ponto x_1 como inicial.
- Para $i = 1, 2, \ldots$ até que o erro seja menor que um valor especificado, calcule

$$x^{(i+1)} = x^{(i)} - \frac{f(x)}{f'(x)}.$$

- Implementação computacional

```
newton <- function(fx, f_prime, x1, tol = 1e-04, max_iter = 10) {
  solucao <- c()
  solucao[1] <- x1
  for(i in 1:max_iter) {
    solucao[i+1] = solucao[i] - fx(solucao[i])/f_prime(solucao[i])
    if( abs(solucao[i+1] - solucao[i]) < tol) break
  }
  return(solucao)
}
```

Aplicação: Método de Newton

- Encontre as raízes de

$$D(\theta) = 2n\left[\log\left(\frac{\hat{\theta}}{\theta}\right) + \bar{y}(\theta - \hat{\theta})\right] \leq 3.84.$$

- Derivada

$$D'(\theta) = 2n(\bar{y} - 1/\theta).$$

```
# Derivada da função a ser resolvida
fprime <- function(theta){2*length(y)*(mean(y) - 1/theta)}
# Solução numerica
Ic_min <- newton(fx = ftheta, f_prime = fprime, x1 = 0.1)
Ic_max <- newton(fx = ftheta, f_prime = fprime, x1 = 2)
c(Ic_min[length(Ic_min)], Ic_max[length(Ic_max)])
```

```
# [1] 0.7684495 1.8545775
```

Método Gradiente Descendente

- Método do Gradiente descendente em geral é usado para otimizar uma função.
- Suponha que desejamos maximizar $F(x)$ cuja derivada é $f(x)$.
- Sabemos que um ponto de inflexão será obtido em $f(x) = 0$.
- Note que $f(x)$ é o gradiente de $F(x)$, assim aponta na direção de máximo/mínimo.
- Assim, podemos caminhar na direção da raiz apenas seguindo o gradiente, i.e.

$$x^{(i+1)} = x^{(i)} - \alpha f(x^{(i)}).$$

- $\alpha > 0$ é um parâmetro de *tuning* usado para controlar o tamanho do passo.
- Escolha do α é fundamental para atingir convergência.
- Busca em gride pode ser uma opção razoável.

Algoritmo: Método Gradiente descendente

- Escolha um ponto x_1 como inicial.
- Para $i = 1, 2, \ldots$ até que o erro seja menor que um valor especificado, calcule

$$x^{(i+1)} = x^{(i)} - \alpha f(x^{(i)}).$$

- Implementação computacional

```
grad_des <- function(fx, x1, alpha, max_iter = 100, tol = 1e-04) {
  sol <- c()
  sol[1] <- x1
  for(i in 1:max_iter) {
    sol[i+1] <- sol[i] + alpha*fx(sol[i])
    if(abs(fx(sol[i+1])) < tol) break
  }
  return(sol)
}
```

Aplicação: Método Gradiente descendente

- Encontre as raízes de

$$D(\theta) = 2n\left[\log\left(\frac{\hat{\theta}}{\theta}\right) + \bar{y}(\theta - \hat{\theta})\right] \leq 3.84.$$

```
# Solução numerica
Ic_min <- grad_des(fx = ftheta, alpha = 0.02, x1 = 0.1)
Ic_max <- grad_des(fx = ftheta, alpha = -0.01, x1 = 4)
c(Ic_min[length(Ic_min)], Ic_max[length(Ic_max)])
```

```
# [1] 0.7684546 1.8545880
```

Sumário

1 Resolvendo equações não-lineares

2 Sistemas de equações

Sistemas de equações

- Sistema com duas equações:

$$\begin{aligned} f_1(x_1, x_2) &= 0 \\ f_2(x_1, x_2) &= 0. \end{aligned}$$

- A solução numérica consiste em encontrar $\hat{x}_1$ e $\hat{x}_2$ que satisfaça o sistema de equações.

Sistemas de equações

- A idéia é facilmente estendida para um sistema com n equações

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= 0. \end{aligned}$$

- Genericamente, tem-se

$$\mathbf{f}(\boldsymbol{x}) = \mathbf{0}.$$

Algoritmo: Método de Newton

- Escolha um vetor $\boldsymbol{x}_1$ como inicial.
- Para $i = 1, 2, \ldots$ até que o erro seja menor que um valor especificado, calcule

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - \mathbf{J}(\boldsymbol{x}^{(i)})^{-1} f(\boldsymbol{x}^{(i)})$$

onde

$$\mathbf{J}(\boldsymbol{x}^{(i)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

é chamado Jacobiano de $\mathbf{f}(\boldsymbol{x})$.

Implementação: Método de Newton

- Implementação computacional

```
newton <- function(fx, jacobian, x1, tol = 1e-04, max_iter = 10) {
  solucao <- matrix(NA, ncol = length(x1), nrow = max_iter)
  solucao[1,] <- x1
  for(i in 1:max_iter) {
    J <- jacobian(solucao[i,])
    grad <- fx(solucao[i,])
    solucao[i+1,] = solucao[i,] - solve(J, grad)
    if( sum(abs(solucao[i+1,] - solucao[i,])) < tol) break
  }
  return(solucao)
}
```

Aplicação: Método de Newton

- Resolva

$$\begin{aligned} f_1(x_1, x_2) &= x_2 - \frac{1}{2}(\exp^{x_1/2} + \exp^{-x/2}) = 0 \\ f_2(x_1, x_2) &= 9x_1^2 + 25x_2^2 - 225 = 0. \end{aligned}$$

- Precisamos obter o Jacobiano, assim tem-se

$$\mathbf{J}(\boldsymbol{x}^{(i)}) = \begin{bmatrix} -\frac{1}{2}(\frac{\exp^{x1/2}}{2} - \frac{\exp^{-x1/2}}{2}) & 1 \\ 18x_1 & 50x_2 \end{bmatrix}.$$

Aplicação: Método de Newton

```r
# Sistema a ser resolvido
fx <- function(x){c(x[2] - 0.5*(exp(x[1]/2) + exp(-x[1]/2)),
                    9*x[1]^2 + 25*x[2]^2 - 225 )}
# Jacobiano
Jacobian <- function(x) {
  jac <- matrix(NA,2,2)
  jac[1,1] <- -0.5*(exp(x[1]/2)/2 - exp(-x[1]/2)/2)
  jac[1,2] <- 1
  jac[2,1] <- 18*x[1]
  jac[2,2] <- 50*x[2]
  return(jac)
}
```

Aplicação: Método de Newton

```
# Resolvendo
sol <- newton(fx = fx, jacobian = Jacobian, x1 = c(1,1))
tail(sol,4) # Solução
```

```
#            [,1]     [,2]
#  [7,] 3.031159 2.385865
#  [8,] 3.031155 2.385866
#  [9,]       NA       NA
# [10,]       NA       NA
```

```
fx(sol[8,]) # OK
```

```
# [1] -3.125056e-12  9.907808e-11
```

Comentários: Método de Newton

- Método de Newton irá convergir tipicamente se três condições forem satisfeitas:
  1. As funções $f_1, f_2, \ldots, f_n$ e suas derivadas forem contínuas e limitadas na vizinhança da solução.
  2. O Jacobiano deve ser diferente de zero na vizinhança da solução.
  3. A estimativa inicial de solução deve estar suficientemente próxima da solução exata.
- Derivadas parciais (elementos da matriz Jacobiana) devem ser determinados. Isso pode ser feito analitica ou numericamente.
- Cada passo do algoritmo envolve a inversão de uma matriz.

Método Gradiente descendente

- O método estende naturalmente para sistema de equações não-lineares.
- Escolha um vetor $\boldsymbol{x}_1$ como inicial.
- Para $i = 1, 2, \ldots$ até que o erro seja menor que um valor especificado, calcule

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha \mathbf{f}(\boldsymbol{x}^{(i)}).$$

- Implementação computacional

```
fx2 <- function(x) { fx(abs(x)) }
grad_des <- function(fx, x1, alpha, max_iter = 100, tol = 1e-04) {
  solucao <- matrix(NA, ncol = length(x1), nrow = max_iter)
  solucao[1,] <- x1
  for(i in 1:c(max_iter-1)) {
    solucao[i+1,] <- solucao[i,] + alpha*fx(solucao[i,])
    #print(solucao[i+1,])
    if( sum(abs(solucao[i+1,] - solucao[i,])) < tol) break
  }
  return(sol)
}
```

Aplicação: Método Gradiente descendente

- Resolva

$$
\begin{aligned}
f_1(x_1, x_2) &= x_2 - \frac{1}{2}(\exp^{x_1/2} + \exp^{-x/2}) = 0 \\
f_2(x_1, x_2) &= 9x_1^2 + 25x_2^2 - 225 = 0.
\end{aligned}
$$

```
#            [,1]     [,2]
#  [5,] 3.154278 2.362746
#  [6,] 3.034792 2.385386
#  [7,] 3.031159 2.385865
#  [8,] 3.031155 2.385866
#  [9,]       NA       NA
# [10,]       NA       NA


# [1] -3.125056e-12  9.907808e-11
```

Comentários: Método Gradiente descendente

- Vantagem: Não precisa calcular o Jacobiano!!
- Desvantagem: Precisa de *tuning*.
- Em geral precisa de mais iterações que o método de Newton.
- Cada iteração é mais barata computacionalmente.
- Uma variação do método é conhecido como *steepest descent*.
- Avalia a mudança em $f(x)$ para um gride de α e da o passo usando o α que torna $F(x)$ maior/menor.
- O tamanho do passo pode ser adaptativo.