Sistema de Equações Lineares

Prof. Wagner Hugo Bonat

Curso de Especialização em
Data Science & Big Data
Universidade Federal do Paraná

12 de março de 2020

Conteúdo

Conteúdo

Sistemas de equações

- Sistema com duas equações:

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

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

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

- Equações podem ser lineares ou não-lineares.

Sistemas de equações lineares

Sistemas de equações lineares

- Cada equação é linear na incógnita.
- Solução analítica em geral é possível.
- Exemplo:

$$\begin{aligned} 7x_1 + 3x_2 &= 45 \\ 4x_1 + 5x_2 &= 29. \end{aligned}$$

- Solução analítica: $x_1 = 6$ e $x_2 = 1$.
- Resolver no quadro (tedioso!!).
- Três possíveis casos:
  1. Uma única solução (sistema não singular).
  2. Infinitas soluções (sistema singular).
  3. Nenhuma solução (sistema impossível).

Sistemas de equações lineares

- Representação matricial do sistema de equações lineares:

$$\mathbf{A} = \begin{bmatrix} 7 & 3 \\ 4 & 5 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{e} \quad \mathbf{b} = \begin{bmatrix} 45 \\ 29 \end{bmatrix}.$$

- De forma geral, tem-se

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

Operações com linhas

- Sem qualquer alteração na relação linear, é possível
  1. Trocar a posição de linhas:

$$\begin{aligned} 4x_1 + 5x_2 &= 29 \\ 7x_1 + 3x_2 &= 45. \end{aligned}$$

  2. Multiplicar qualquer linha por uma constante, aqui $4x_1 + 5x_2$ por $\frac{1}{4}$, obtendo

$$\begin{aligned} x_1 + \frac{5}{4}x_2 &= \frac{29}{4} && (1) \\ 7x_1 + 3x_2 &= 45. && (2) \end{aligned}$$

  3. Subtrair um múltiplo de uma linha de uma outra, aqui $7 * Eq.(1)$ menos Eq. (2), obtendo

$$\begin{aligned} x_1 + \frac{5}{4}x_2 &= \frac{29}{4} \\ 0x_1 + (\frac{35}{4} - 3)x_2 &= \frac{203}{4} - 45. \end{aligned}$$

  4. Fazendo as contas, tem-se

$$0x_1 + \frac{23}{4}x_2 = \frac{23}{4}.$$

Solução de sistemas lineares

- Forma geral de um sistema com n equações lineares:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

- Matricialmente, tem-se

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Métodos diretos e métodos iterativos.

Métodos diretos

- O sistema de equações é manipulado até se transformar em um sistema equivalente de fácil resolução.
- Triangular superior:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

- Substituição regressiva

$$x_n = \frac{b_n}{a_{nn}} \quad x_i = \frac{b_i - \sum_{j=i+1}^{j=n} a_{ij} x_j}{a_{ii}}, \quad i = n-1, n-2, \ldots, 1.$$

Métodos diretos

- Triangular inferior:

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

- Substituição progressiva

$$x_1 = \frac{b_1}{a_{11}} \quad x_i = \frac{b_i - \sum_{j=i}^{j=i-1} a_{ij} x_j}{a_{ii}}, \quad i = 2, 3, \ldots, n.$$

Métodos diretos

- Diagonal:

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

Métodos diretos: Eliminação de Gauss

- Método de eliminação de Gauss consiste em manipular o sistema original usando operações de linha até obter um sistema triangular superior.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{23} & a_{33} & a_{34} \\ a_{41} & a_{24} & a_{34} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

- Usar eliminação progressiva no novo sistema para obter a solução.
- Resolva o seguinte sistema usando Eliminação de Gauss.

$$\begin{bmatrix} 3 & 2 & 6 \\ 2 & 4 & 3 \\ 5 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 23 \\ 33 \end{bmatrix}$$

Métodos diretos: Eliminação de Gauss

- Passo 1: Encontrar o pivô e eliminar os elementos abaixo dele usando operações de linha.

$$\begin{bmatrix} [3] & 2 & 6 \\ 2-\frac{2}{3}3 & 4-\frac{2}{3}2 & 3-\frac{2}{3}6 \\ 5-\frac{5}{3}3 & 3-\frac{5}{3}2 & 4-\frac{5}{3}6 \end{bmatrix} \begin{bmatrix} 24 \\ 23-\frac{2}{3}24 \\ 33-\frac{5}{3}24 \end{bmatrix} \longrightarrow \begin{bmatrix} [3] & 2 & 6 \\ 0 & \frac{8}{3} & -1 \\ 0 & -\frac{1}{3} & -6 \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -7 \end{bmatrix}$$

- Passo 2: Encontrar o segundo pivô e eliminar os elementos abaixo dele usando operações de linha.

$$\begin{bmatrix} 3 & 2 & 6 \\ 0 & [\frac{8}{3}] & -1 \\ 0 & -\frac{1}{3}-\left(-\frac{3}{24}\right)\left(\frac{8}{3}\right) & -6-(-\frac{3}{24})(-1) \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -7-(-\frac{3}{24})(7) \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 2 & 6 \\ 0 & [\frac{8}{3}] & -1 \\ 0 & 0 & -\frac{147}{24} \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -\frac{147}{24} \end{bmatrix}$$

- Passo 3: Substituição regressiva.

Métodos diretos: Eliminação de Gauss

- Usando a fórmula de substituição regressiva temos:
  1. $x_3 = \frac{b_3}{a_{33}} = 1.$
  2. $x_2 = \frac{b_2 - a_{23}x_3}{a_{22}} = 3.$
  3. $x_1 = \frac{(b_1 - (a_{12}x_2 + a_{13}x_3)}{a_{11}} = 4.$
- A extensão do procedimento para um sistema com n equações é trivial.
  1. Transforme o sistema em triangular superior usando operações linhas.
  2. Resolva o novo sistema usando substituição regressiva.
- Potenciais problemas do método de eliminação de Gauss:
  1. O elemento pivô é zero.
  2. O elemento pivô é pequeno em relação aos demais termos.

Eliminação de Gauss com pivotação

- Considere o sistema

$$\begin{aligned} 0x_1 + 2x_2 + 3x_2 &= 46 \\ 4x_1 - 3x_2 + 2x_3 &= 16 \\ 2x_1 + 4x_2 - 3x_3 &= 12 \end{aligned}$$

- Neste caso o pivô é zero e o procedimento não pode começar.
- Pivotação - trocar a ordem das linhas.
  1. Evitar pivôs zero.
  2. Diminuir o número de operações necessárias para triangular o sistema.

$$\begin{aligned} 4x_1 - 3x_2 + 2x_3 &= 16 \\ 2x_1 + 4x_2 - 3x_3 &= 12 \\ 0x_1 + 2x_2 + 3x_2 &= 46 \end{aligned}$$

Eliminação de Gauss com pivotação

- Se durante o procedimento uma equação pivô tiver um elemento nulo e o sistema tiver solução, uma equação com um elemento pivô diferente de zero sempre existirá.
- Cálculos numéricos são menos propensos a erros e apresentam menores erros de arredondamento se o elemento pivô for grande em valor absoluto.
- É usual ordenar as linhas para que o maior valor seja o primeiro pivô.

Implementação: Eliminação de Gauss sem pivotação

- Passo 1: Obtendo uma matriz triangular superior.

```
gauss <- function(A, b) {
  # Sistema aumentado
  Ae <- cbind(A, b)
  n_row <- nrow(Ae)
  n_col <- ncol(Ae)
  # Matriz para receber os resultados
  SOL <- matrix(NA, n_row, n_col)
  # Pivotação
  #Ae <- Ae[order(Ae[,1], decreasing = TRUE),]
  SOL[1,] <- Ae[1,]
  pivo <- matrix(0, n_col, n_row)
  for(j in 1:c(n_row-1)) {
    for(i in c(j+1):c(n_row)) {
      pivo[i,j] <- Ae[i,j]/SOL[j,j]
      SOL[i,] <- Ae[i,] - pivo[i,j]*SOL[j,]
      Ae[i,] <- SOL[i,]
    }
  }
  return(SOL)
}
```

Implementação: Eliminação de Gauss sem pivotação

- Passo 2: Substituição regressiva.

```
sub_reg <- function(SOL) {
  n_row <- nrow(SOL)
  n_col <- ncol(SOL)
  A <- SOL[1:n_row,1:n_row]
  b <- SOL[,n_col]
  n <- length(b)
  x <- c()
  x[n] <- b[n]/A[n,n]
  for(i in (n-1):1) {
    x[i] <- (b[i] - sum(A[i,c(i+1):n]*x[c(i+1):n] ))/A[i,i]
  }
  return(x)
}
```

Aplicação: Eliminação de Gauss sem pivotação

- Resolva o sistema:

$$\begin{bmatrix} 3 & 2 & 6 \\ 2 & 4 & 3 \\ 5 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 23 \\ 33 \end{bmatrix}.$$

```
A <- matrix(c(3,2,5,2,4,3,6,3,4),3,3)
b <- c(24,23,33)
# Passo 1: Triangularização
S <- gauss(A, b)
# Passo 2: Substituição regressiva
sol = sub_reg(SOL = S)
sol

## [1] 4 3 1

# Verificando a solução
A%*%sol

##      [,1]
## [1,]   24
## [2,]   23
## [3,]   33
```

Métodos diretos: Eliminação de Gauss-Jordan

- O sistema original é manipulado até obter um sistema equivalente na forma diagonal.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{23} & a_{33} & a_{34} \\ a_{41} & a_{24} & a_{34} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

- Algoritmo Gauss-Jordan
  1. Normalize a equação pivô com a divisão de todos os seus termos pelo coeficiente pivô.
  2. Elimine os elementos fora da diagonal principal em TODAS as demais equações usando operaçõs de linha.
- O método de Gauss-Jordan pode ser combinado com pivotação igual ao método de eliminação de Gauss.

Decomposição LU

- Nos métodos de eliminação de Gauss e Gauss-Jordan resolvemos sistemas do tipo

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}.$$

- Sendo dois sistemas

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}_1, \quad \text{e} \quad \mathbf{A}\boldsymbol{x} = \boldsymbol{b}_2.$$

- Cálculos do primeiro não ajudam a resolver o segundo.
- IDEAL! - Operações realizadas em $\mathbf{A}$ fossem dissociadas das operações em $\boldsymbol{b}$.

Decomposição LU

- Suponha que precisamos resolver vários sistemas do tipo

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}.$$

  para diferentes $\boldsymbol{b}'s$.
- Opção 1 - Calcular a inversa $\mathbf{A}^{-1}$, assim a solução

$$\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{b}.$$

- Cálculo da inversa é computacionalmente ineficiente.

Algoritmo: Decomposição LU

1. Decomponha (fatore) a matriz $\mathbf{A}$ em um produto de duas matrizes

$$\mathbf{A} = \mathbf{LU},$$

   onde $\mathbf{L}$ é triangular inferior e $\mathbf{U}$ é triangular superior.
2. Baseado na decomposição o sistema tem a forma:

$$\mathbf{LU}\boldsymbol{x} = \boldsymbol{b}. \tag{3}$$

3. Defina $\mathbf{U}\boldsymbol{x} = \boldsymbol{y}$.
4. Substituindo em 3 tem-se

$$\mathbf{L}\boldsymbol{y} = \boldsymbol{b}. \tag{4}$$

5. Solução é obtida em dois passos
   - 5.1 Resolva Eq.(4) para obter $\boldsymbol{y}$ usando substituição progressiva.
   - 5.2 Resolva Eq.(3) para obter $\boldsymbol{x}$ usando substituição regressiva.

Obtendo as matrizes **L** e **U**

- Método de eliminação de Gauss e método de Crout.
- Dentro do processo de eliminação de Gauss as matrizes **L** e **U** são obtidas como um subproduto, i.e.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{41} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ m_{21} & 1 & & \\ m_{31} & m_{32} & 1 & \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix}.$$

- Os elementos $m'_{ij}s$ são os multiplicadores que multiplicam a equação pivô.

Obtendo as matrizes **L** e **U**

- Relembre o exemplo de eliminação de Gauss.

$$\begin{bmatrix} [3] & 2 & 6 \\ 2-\frac{2}{3}3 & 4-\frac{2}{3}2 & 3-\frac{2}{3}6 \\ 5-\frac{5}{3}3 & 3-\frac{5}{3}2 & 4-\frac{5}{3}6 \end{bmatrix} \begin{bmatrix} 24 \\ 23-\frac{2}{3}24 \\ 33-\frac{5}{3}24 \end{bmatrix} \longrightarrow \begin{bmatrix} [3] & 2 & 6 \\ 0 & \frac{8}{3} & -1 \\ 0 & -\frac{1}{3} & -6 \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -7 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 2 & 6 \\ 0 & [\frac{8}{3}] & -1 \\ 0 & -\frac{1}{3}-\left(-\frac{3}{24}\right)\left(\frac{8}{3}\right) & -6-(-\frac{3}{24})(-1) \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -7-(-\frac{3}{24})(7) \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 2 & 6 \\ 0 & [\frac{8}{3}] & -1 \\ 0 & 0 & -\frac{147}{24} \end{bmatrix} \begin{bmatrix} 24 \\ 7 \\ -\frac{147}{24} \end{bmatrix}$$

- Neste caso, tem-se

$$\mathbf{L} = \begin{bmatrix} 1 & & \\ \frac{2}{3} & 1 & \\ \frac{5}{3} & -\frac{3}{24} & 1 \end{bmatrix} \quad \text{e} \quad \mathbf{U} = \begin{bmatrix} 3 & 2 & 6 \\ 0 & \frac{8}{3} & -1 \\ 0 & 0 & -\frac{147}{24} \end{bmatrix}.$$

Decomposição LU com pivotação

- O método de eliminação de Gauss foi realizado sem pivotação.
- Como discutido a pivotação pode ser necessária.
- Quando realizada a pivotação as mudanças feitas devem ser armazenadas, tal que

$$\mathbf{PA} = \mathbf{LU}.$$

- **P** é uma matriz de permutação.
- Se as matrizes **LU** forem usadas para resolver o sistema

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b},$$

então a ordem das linhas de $\boldsymbol{b}$ deve ser alterada de forma consistente com a pivotação, i.e. $\mathbf{P}\boldsymbol{b}$.

Implementação: Decomposição LU

- Podemos facilmente modificar a função `gauss()` para obter a decomposição LU.

```
my_lu <- function(A) {
  n_row <- nrow(A)
  n_col <- ncol(A)
  # Matriz para receber os resultados
  SOL <- matrix(NA, n_row, n_col)
  SOL[1,] <- A[1,]
  pivo <- matrix(0, n_col, n_row)
  for(j in 1:c(n_row-1)) {
    for(i in c(j+1):c(n_row)) {
      pivo[i,j] <- A[i,j]/SOL[j,j]
      SOL[i,] <- A[i,] - pivo[i,j]*SOL[j,]
      A[i,] <- SOL[i,]
    }
  }
  diag(pivo) <- 1
  return(list("L" = pivo, "U" = SOL))
}
```

Aplicação: Decomposição LU

- Fazendo a decomposição.

```
LU <- my_lu(A) # Decomposição
LU

## $L
##           [,1]   [,2] [,3]
## [1,] 1.0000000  0.000    0
## [2,] 0.6666667  1.000    0
## [3,] 1.6666667 -0.125    1
##
## $U
##      [,1]          [,2]   [,3]
## [1,]    3  2.000000e+00  6.000
## [2,]    0  2.666667e+00 -1.000
## [3,]    0 -5.551115e-17 -6.125

LU$L %*% LU$U # Verificando a solução

##      [,1] [,2] [,3]
## [1,]    3    2    6
## [2,]    2    4    3
## [3,]    5    3    4
```

Aplicação: Decomposição LU

- Resolvendo o sistema de equações.

```
# Passo 1: Substituição progressiva
y = forwardsolve(LU$L, b)
# Passo 2: Substituição regressiva
x = backsolve(LU$U, y)
x

## [1] 4 3 1

A%*%x # Verificando a solução

##      [,1]
## [1,]   24
## [2,]   23
## [3,]   33
```

- Função `lu()` do `Matrix` fornece a decomposição LU.

```
require(Matrix)
LU_M <- lu(A) # Calcula mas não retorna
LU_M <- expand(LU_M) # Captura as matrizes L U e P
# Substituição progressiva. NOTE MATRIZ DE PERMUTAÇÃO
y <- forwardsolve(LU_M$L, LU_M$P%*%b)
x = backsolve(LU_M$U, y) # Substituição regressiva
x

## [1] 4 3 1
```

Obtendo a inversa via decomposição LU

- O método LU é especialmente adequado para o cálculo da inversa.
- Lembre-se que a inversa de $\mathbf{A}$ é tal que

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

- O procedimento de cálculo da inversa é essencialmente o mesmo da solução de um sistema de equações lineares, porém com mais incognitas.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Quatro sistemas de equações diferentes, em cada sistema, uma coluna da matriz $\mathbf{X}$ é a incognita.

Implementação: Inversa via decomposição LU

- Função para resolver o sistema usando decomposição LU.

```
solve_lu <- function(LU, b) {
  y <- forwardsolve(LU_M$L, LU_M$P%*%b)
  x = backsolve(LU_M$U, y)
  return(x)
}
```

- Resolvendo vários sistemas

```
my_solve <- function(LU, B) {
  n_col <- ncol(B)
  n_row <- nrow(B)
  inv <- matrix(NA, n_col, n_row)
  for(i in 1:n_col) {
    inv[,i] <- solve_lu(LU, B[,i])
  }
  return(inv)
}
```

Aplicação: Inversa via decomposição LU

- Calcule a inversa de

$$\mathbf{A} = \begin{bmatrix} 3 & 2 & 6 \\ 2 & 4 & 3 \\ 5 & 3 & 4 \end{bmatrix}$$

```
A <- matrix(c(3,2,5,2,4,3,6,3,4),3,3)
I <- Diagonal(3, 1)
# Decomposição LU
LU <- my_lu(A)
# Obtendo a inversa
inv_A <- my_solve(LU = LU, B = I)
inv_A
```

```
##            [,1]        [,2]        [,3]
## [1,] -0.1428571 -0.20408163  0.36734694
## [2,] -0.1428571  0.36734694 -0.06122449
## [3,]  0.2857143 -0.02040816 -0.16326531
```

```
# Verificando o resultado
A%*%inv_A
```

```
##      [,1]          [,2]          [,3]
## [1,]    1  6.938894e-17  0.000000e+00
## [2,]    0  1.000000e+00 -5.551115e-17
## [3,]    0 -2.775558e-17  1.000000e+00
```

Cálculo da inversa via método de Gauss-Jordan

- Procedimento Gauss-Jordan:

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{32} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & a'_{11} & a'_{21} & a'_{31} \\ 0 & 1 & 0 & a'_{21} & a'_{22} & a'_{32} \\ 0 & 0 & 1 & a'_{31} & a'_{32} & a'_{33} \end{bmatrix}.$$

- Função `solve()` usa a decomposição LU com pivotação.
- `R` básico é construído sobre a biblioteca `lapack` escrita em `C`.
- Veja documentação em http://www.netlib.org/lapack/lug/node38.html.

Métodos iterativos

- Nos métodos iterativos, as equações são colocadas em uma forma explícita onde cada incógnita é escrita em termos das demais, i.e.

$$\begin{array}{ll} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 & x_1 = [b_1 - (a_{12}x_2 + a_{13}x_3)]/a_{11} \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \rightarrow & x_2 = [b_2 - (a_{21}x_1 + a_{23}x_3)]/a_{22}. \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 & x_3 = [b_3 - (a_{31}x_1 + a_{32}x_2)]/a_{33} \end{array}$$

- Dado um valor inicial para as incógnitas estas serão atualizadas até a convergência.
- Atualização: Método de Jacobi

$$x_i = \frac{1}{a_{ii}} \left[b_i - \left(\sum_{j=1; j \neq i}^{j=n} a_{ij} x_j \right) \right] \quad i = 1, \ldots, n.$$

Método iterativo de Jacobi

- Implementação computacional

```
jacobi <- function(A, b, inicial, max_iter = 10, tol = 1e-04) {
  n <- length(b)
  x_temp <- matrix(NA, ncol = n, nrow = max_iter)
  x_temp[1,] <- inicial
  x <- x_temp[1,]
  for(j in 2:max_iter) {
    for(i in 1:n) {
      x_temp[j,i] <- (b[i] - sum(A[i,1:n][-i]*x[-i]))/A[i,i]
    }
    x <- x_temp[j,]
    if(sum(abs(x_temp[j,] - x_temp[c(j-1),])) < tol) break
  }
  return(list("Solucao" = x, "Iteracoes" = x_temp))
}
```

- Método de Gauss-Seidel: usa a versão mais recente da solução para dar o próximo passo.

Aplicação: Método iterativo de Jacobi

- Cuidado!! convergência não é garantida !!

```
A <- matrix(c(9,2,-3,-2,-2,8,2,3,3,-2,11,2,2,3,-4,10),4,4)
b <- c(54.5, -14, 12.5, -21)
ss <- jacobi(A = A, b = b, inicial = c(0,0,0,0), max_iter = 15)
# Solução aproximada
ss$Solucao

## [1]  4.999502 -1.999771  2.500056 -1.000174

# Solução exata
solve(A, b)

## [1]  5.0 -2.0  2.5 -1.0
```

- Método de Gauss-Seidel: usa a versão mais recente da solução para dar o próximo passo.

Aplicação: Método iterativo de Jacobi e Gauss-Seidel

- Em R o pacote `Rlinsolve` fornece implementações eficientes dos métodos de Jacobi e Gauss-Seidel.
- `Rlinsolve` inclui suporte para matrizes esparsas via `Matrix`.

```
A <- matrix(c(9,2,-3,-2,-2,8,2,3,3,-2,11,2,2,3,-4,10),4,4)
b <- c(54.5, -14, 12.5, -21)
#Loading extra package
require(Rlinsolve)
#Jacobi's method
lsolve.jacobi(A, b)$x

##            [,1]
## [1,]  4.9999633
## [2,] -2.0000807
## [3,]  2.5000216
## [4,] -0.9999336

# Gauss-Seidel's method
lsolve.gs(A, b)$x

##            [,1]
## [1,]  4.9999684
## [2,] -2.0000498
## [3,]  2.5000127
## [4,] -0.9999775
```

- `Rlinsolve` é implementado em `C++` usando o pacote `Rcpp`.

Autovalores e Autovetores

Motivação

- Redução de dimensionalidade é fundamental em *Data Science*.
- Análise de Componentes principais (PCA) e análise fatorial (AF) são ferramentas populares.
- Decompor grandes e complicados relacionamentos multivariados em simples componentes não relacionados.
- Aplicações: Regressão ridge, PCA e análise fatorial.
- Vamos discutir apenas os aspectos matemáticos.

Intuição

- Podemos decompor um vetor $\boldsymbol{v}$ em duas informações separadas: direção $\boldsymbol{d}$ e tamanho λ, i.e

$$\lambda = ||\boldsymbol{v}|| = \sqrt{\sum_j \nu_j^2}, \quad \text{e} \quad \boldsymbol{d} = \frac{\boldsymbol{v}}{\lambda}.$$

- É mais fácil interpretar o tamanho de um vetor enquanto ignorando a sua direção e vice-versa.
- Esta idéia pode ser extendida para matrizes.
- Uma matriz nada mais é do que um conjunto de vetores.
- IDÉIA - decompor a informação de uma matriz em outros componentes de mais fácil interpretação/representação matemática.

Autovalores e Autovetores

- Autovalores e autovetores são definidos por uma simples igualdade

$$\mathbf{A}\boldsymbol{v} = \lambda \boldsymbol{v}. \tag{5}$$

- Qualquer vetor unitário $\boldsymbol{v}$ que satisfaz a Eq. (5) tem apenas o efeito de alongar ou encolher $\boldsymbol{v}$.
- O vetor não é rotacionado e a direção é preservada.
- Os vetores $\boldsymbol{v}$'s que satisfazem Eq. (5) são os autovetores.
- Os valores λ's que satisfazem Eq. (5) são os autovalores.
- Vamos considerar o caso em que $\mathbf{A}$ é simétrica.
- A idéia pode ser extendida para matrizes não simétrica.

Autovalores e Autovetores

- Se $\mathbf{A}$ é uma matriz simétrica $n \times n$, então existe exatamente n pares $(\lambda_j, \boldsymbol{v}_j)$ que satisfazem a equação:

$$\mathbf{A}\boldsymbol{v} = \boldsymbol{\lambda}\boldsymbol{v}.$$

- Se $\mathbf{A}$ tem autovalores $\lambda_1, \ldots, \lambda_n$, então:
  1. $\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i$.
  2. $\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i$.
  3. $\mathbf{A}$ é positiva definida, se e somente se todos $\lambda_j > 0$.
  4. $\mathbf{A}$ é semi-positiva definida, se e somente se todos $\lambda_j \geq 0$.
- A idéia de PCA é decompor/fatorar a matrix $\mathbf{A}$ em componentes mais simples de interpretar.

Decomposição em autovalores e autovetores

- Teorema: Qualquer matriz simétrica $\mathbf{A}$ pode ser fatorada em

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top,$$

  onde $\mathbf{\Lambda}$ é diagonal contendo os autovalores de $\mathbf{A}$ e as colunas de $\mathbf{Q}$ contêm os autovetores ortonormais.
- Vetores ortonormais: são multuamente ortogonais e de comprimento unitário.
- Teorema: Se $\mathbf{A}$ tem autovetores $\mathbf{Q}$ e autovalores λ_j. Então $\mathbf{A}^{-1}$ tem autores $\mathbf{Q}$ e autovalores λ_j^{-1}.
- Implicação: Se $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ então $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^\top$.

Diagonalização

- Autovalores são utéis porque eles permitem lidar com matrizes da mesma forma que lidamos com números.
- Todos os cálculos são feitos na matriz diagonal $\mathbf{\Lambda}$.
- Este processo é chamado de diagonalização.
- Um dos resultados mais poderosos em Álgebra Linear é que qualquer matriz pode ser diagonalizada.
- O processo de diagonalização é chamado de Decomposição em valores singulares.

Decomposição em valores singulares (SVD)

- Teorema: Qualquer matriz $\mathbf{A}$ pode ser decomposta em,

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top,$$

  onde $\mathbf{D}$ é diagonal com entradas não negativas e $\mathbf{U}$ e $\mathbf{V}$ são ortogonal, i.e. $\mathbf{U}^\top\mathbf{U} = \mathbf{V}^\top\mathbf{V} = \mathbf{I}$.
- Matrizes não quadradas não tem autovalores.
- Os elementos de $\mathbf{D}$ são chamados de valores singulares.
- Os valores singulares são os autovalores de $\mathbf{A}^\top\mathbf{A}$.

Dimensão da SVD

- Se $\mathbf{A}$ é $n \times n$, então $\mathbf{U}$, $\mathbf{D}$ e $\mathbf{V}$ são $n \times n$.
- Se $\mathbf{A}$ é $n \times p$, sendo $n > p$, então $\mathbf{U}$ é $n \times p$, $\mathbf{D}$ e $\mathbf{V}$ são $p \times p$.
- Se $\mathbf{A}$ é $n \times p$, sendo $n < p$, então $\mathbf{V}^\top$ é $n \times p$, $\mathbf{D}$ e $\mathbf{U}$ são $n \times n$.
- $\mathbf{D}$ será sempre quadrada com dimensão igual ao mínimo entre p e n.

Decomposição em autovalores e autovetores em R

- Função `eigen()` fornece a decomposição

```
A <- matrix(c(1,0.8, 0.3, 0.8, 1, 0.2, 0.3, 0.2, 1),3,3)
isSymmetric.matrix(A)

## [1] TRUE

out <- eigen(A)
Q <- out$vectors
D <- diag(out$values)
# Autovetores
Q

##            [,1]       [,2]        [,3]
## [1,] -0.6712373 -0.1815663  0.71866142
## [2,] -0.6507744 -0.3198152 -0.68862977
## [3,] -0.3548708  0.9299204 -0.09651322

# Autovalores
D

##          [,1]      [,2]      [,3]
## [1,] 1.934216 0.0000000 0.0000000
## [2,] 0.000000 0.8726419 0.0000000
## [3,] 0.000000 0.0000000 0.1931419

# Verificando
Q%*%D%*%t(Q)

##      [,1] [,2] [,3]
## [1,]  1.0  0.8  0.3
## [2,]  0.8  1.0  0.2
## [3,]  0.3  0.2  1.0
```

Decomposição em valores singulares em R

- Função `svd()` fornece a decomposição

```
svd(A)

## $d
## [1] 1.9342162 0.8726419 0.1931419
##
## $u
##            [,1]       [,2]        [,3]
## [1,] -0.6712373  0.1815663  0.71866142
## [2,] -0.6507744  0.3198152 -0.68862977
## [3,] -0.3548708 -0.9299204 -0.09651322
##
## $v
##            [,1]       [,2]        [,3]
## [1,] -0.6712373  0.1815663  0.71866142
## [2,] -0.6507744  0.3198152 -0.68862977
## [3,] -0.3548708 -0.9299204 -0.09651322
```

Aplicação: Regressão ridge

- Relembrando: Regressão linear múltipla

$$\underset{n\times 1}{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}} = \underset{n\times p}{\begin{bmatrix} 1 & x_{11} & \dots & x_{p1} \\ 1 & x_{12} & \dots & x_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \dots & x_{pn} \end{bmatrix}} \underset{p\times 1}{\begin{bmatrix} \beta_0 \\ \vdots \\ \beta_p \end{bmatrix}}$$

- Usando uma notação mais compacta,

$$\underset{n\times 1}{\mathbf{y}} = \underset{n\times p}{\mathbf{X}}\underset{p\times 1}{\boldsymbol{\beta}}.$$

- Minimiza a perda quadrática:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}.$$

Aplicação: Regressão ridge

- Se $p > n$ o sistema é singular (múltiplas soluções)!!
- Como podemos ajustar o modelo?
- Introduzir uma penalidade pela complexidade.
- Soma de quadrados penalizada

$$PSQ(\beta) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2.$$

- Matricialmente, tem-se

$$PSQ(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) + \lambda \beta^\top \beta.$$

- IMPORTANTE !!
  1. $\mathbf{y}$ centrado (média zero).
  2. $\mathbf{X}$ padronizada por coluna (média zero e variância um).

Aplicação: Regressão ridge

- Objetivo: Minizar a soma de quadrados penalizada.
- Derivada

$$
\begin{aligned}
\frac{\partial PQS(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \frac{\partial}{\partial \boldsymbol{\beta}} \left[(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta} \right] \\
&= \left[\frac{\partial}{\partial \boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top \right] (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top \left[\frac{\partial}{\partial \boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right] + \lambda \left[\frac{\partial \boldsymbol{\beta}^\top}{\partial \boldsymbol{\beta}} \right] + \boldsymbol{\beta}^\top \left[\frac{\partial \boldsymbol{\beta}}{\partial \boldsymbol{\beta}} \right] \\
&= -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + 2\lambda \boldsymbol{\beta} \\
&= \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}.
\end{aligned}
$$

Aplicação: Regressão ridge

- Resolvendo o sistema linear, tem-se

$$\begin{aligned}
-\mathbf{X}^\top(\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) + \lambda\mathbf{I}\hat{\boldsymbol{\beta}} &= \mathbf{0} \\
-\mathbf{X}^\top\boldsymbol{y} + \mathbf{X}^\top\mathbf{X}\hat{\boldsymbol{\beta}} + \lambda\mathbf{I}\hat{\boldsymbol{\beta}} &= \mathbf{0} \\
\mathbf{X}^\top\mathbf{X}\hat{\boldsymbol{\beta}} + \lambda\mathbf{I}\hat{\boldsymbol{\beta}} &= \mathbf{X}^\top\boldsymbol{y} \\
\left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)\hat{\boldsymbol{\beta}} &= \mathbf{X}^\top\boldsymbol{y} \\
\hat{\boldsymbol{\beta}} &= \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\boldsymbol{y}.
\end{aligned}$$

- Solução depende de λ.
- A inclusão de λ faz o sistema ser não singular.
- Na verdade quando fixamos λ selecionamos uma solução em particular.

Aplicação: Regressão ridge

- Calcular $\hat{\beta}$ envolve a inversão de uma matrix $p \times p$ potencialmente grande.

$$\hat{\beta} = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^\top \boldsymbol{y}.$$

- Usando a decomposição SVD, tem-se

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top.$$

- É fácil mostrar que,

$$\hat{\beta} = \mathbf{V}\mathrm{diag}\left(\frac{d_j}{d_j^2 + \lambda}\right)\mathbf{U}^\top \boldsymbol{y}.$$

Implementação: Regressão ridge

- Simulando o conjunto de dados ($n = 100, p = 200$).

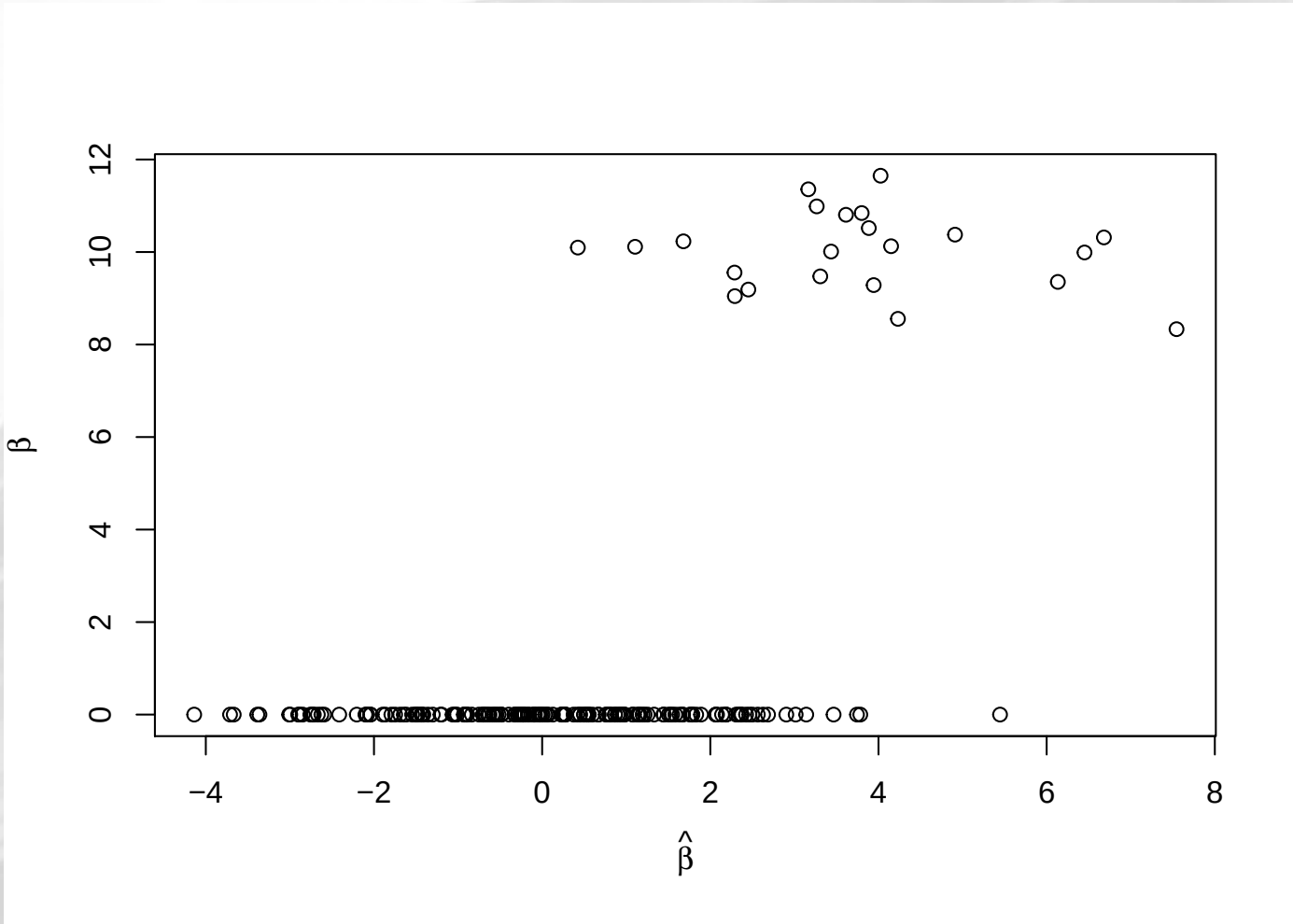
```
set.seed(123)
X <- matrix(NA, ncol = 200, nrow = 100)
X[,1] <- 1 # Intercepto
for(i in 2:200) {
  X[,i] <- rnorm(100, mean = 0, sd = 1)
  X[,i] <- (X[,i] - mean(X[,i]))/var(X[,i])
}
# Parâmetros
beta <- rbinom(200, size = 1, p = 0.1)*rnorm(200, mean = 10)
mu <- X%*%beta
# Observações
y <- rnorm(100, mean = mu, sd = 10)
```

- Implementando o modelo.

```
y_c <- y - mean(y)
X_svd <- svd(X) # Decomposição svd
lambda = 0.5 # Penalização
DD <- Diagonal(100, X_svd$d/(X_svd$d^2 + lambda))
DD[1] <- 0 # Não penalizar o intercepto
beta_hat = as.numeric(X_svd$v%*%DD%*%t(X_svd$u)%*%y_c)
```

Resultados: Regressão ridge

- Ajustados versus verdadeiros.



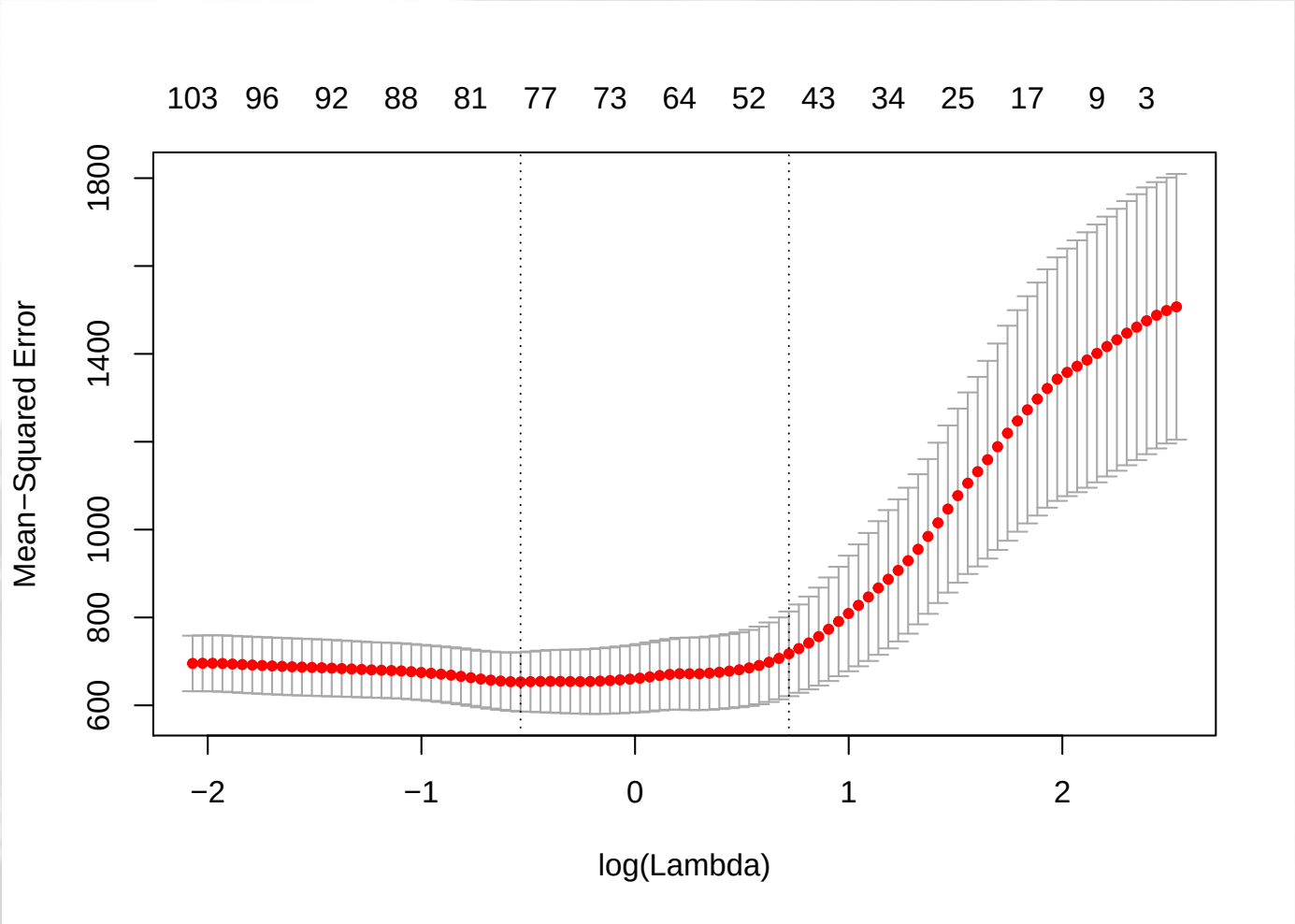
Resultados: Regressão ridge

- Regressão com penalização ridge, bem como, outras penalizações são eficientemente implementadas em `R` via pacote `glmnet`.
- IMPORTANTE! A penalização no `glmnet` é ligeiramente diferente, por isso os $\hat{\beta}$'s não são idênticos a nossa implementação *naive*.
- `glmnet` oferece opções para selecionar λ via validação cruzada.

```
require(glmnet)
beta_glm <- cv.glmnet(X[,-1], y_c, nlambda = 100)
```

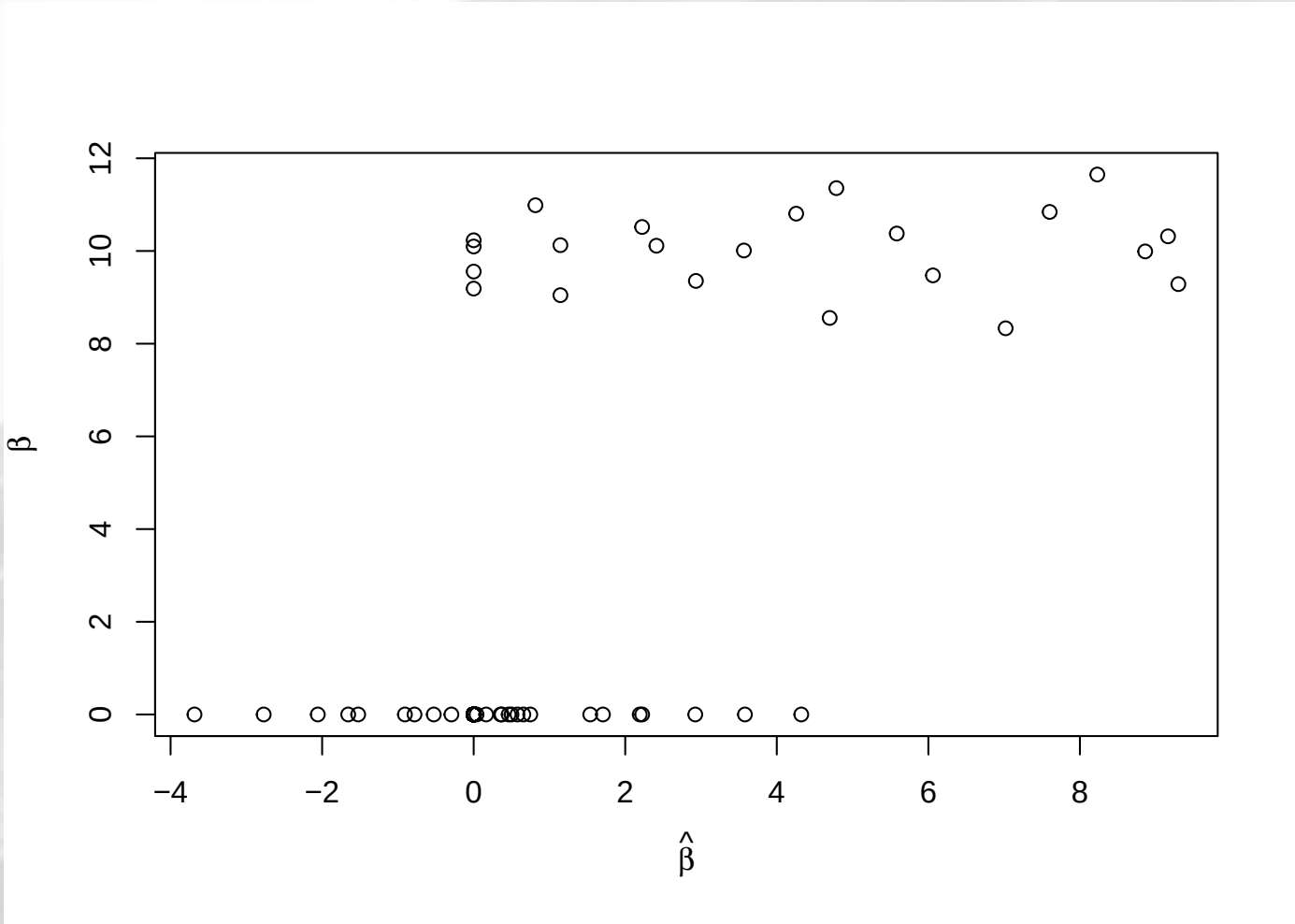
Resultados: Regressão ridge

- Validação cruzada.



Resultados: Regressão ridge

- Ajustados (`glmnet`) versus verdadeiros.



Comentários

- Solução de sistemas lineares:
  1. Métodos diretos: Eliminação de Gauss e Gauss-Jordan.
  2. Métodos iterativos: Jacobi e Gauss-Seidel.
  3. Inversa de matrizes.
- Decomposição ou fatorização
  1. LU resolve sistema lineares e eficiente para obter inversas.
  2. Autovalores e autovetores.
  3. Valores singulares.
  4. Existem muitas outras fatorizações: QR, Cholesky, Cholesky modificadas, etc.