

# Álgebra Matricial

Prof. Wagner Hugo Bonat



# Vetores e escalares

## Vetores e escalares

- ▶ Um vetor é uma lista de  $n$  números escritos em linha ou coluna.
- ▶ Notação

$$\mathbf{a} = (a_1 \quad \dots \quad a_n) \quad \text{ou} \quad \mathbf{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}.$$

- ▶ Vetor **linha** e vetor **coluna**.
- ▶ Um elemento do vetor é chamado de  $a_i$ , sendo  $i$  a sua posição.

- ▶ O **tamanho de um vetor** é o seu número de elementos.
- ▶ O **módulo de um vetor** é o seu comprimento

$$|\mathbf{a}| = \sqrt{a_1^2 + \dots + a_n^2}.$$

- ▶ Vetor unitário é aquele que tem tamanho 1. Vetor padronizado

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|}.$$

- ▶ Dois vetores são iguais se tem o mesmo tamanho e os seus elementos em posições equivalentes são iguais.

# Operações com vetores

## Operações com vetores

1. Soma  $\mathbf{a} + \mathbf{b} = (a_i + b_i) = (a_1 + b_1, \dots, a_n + b_n)$ .
2. Subtração  $\mathbf{a} - \mathbf{b} = (a_i - b_i) = (a_1 - b_1, \dots, a_n - b_n)$ .
3. Multiplicação por escalar  $\alpha \mathbf{a} = (\alpha a_1, \dots, \alpha a_n)$ .
4. Transposta de um vetor:

$$\mathbf{a} = (a_1 \quad \dots \quad a_n) \quad \mathbf{a}^\top = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}.$$

5. Produto interno ou escalar entre dois vetores resulta em um escalar

$$\mathbf{a} \cdot \mathbf{b} = (a_1 b_1 + a_2 b_2 + \dots + a_n b_n).$$

► **Condições:** os vetores devem ser do mesmo tipo e tamanho.

## Vetores ortogonais

- ▶ Dois vetores são **ortogonais** entre si se o ângulo  $\theta$  entre eles é de  $90^\circ$ .
- ▶ Implicações:  $\cos(\theta) = 0$  e  $\mathbf{a}^\top \mathbf{b} = 0$ .
- ▶ O co-seno do ângulo  $\theta$  entre os vetores é dado por:

$$\cos(\theta) = \frac{\mathbf{a}^\top \mathbf{b}}{\sqrt{\mathbf{a}^\top \mathbf{a}} \sqrt{\mathbf{b}^\top \mathbf{b}}}.$$

# Operações com vetores em R

## ► Declarando vetores

```
a <- c(4,5,6)
b <- c(1,2,3)
```

## ► Sendo a e b compatíveis

```
#### Soma
```

```
a + b
```

```
## [1] 5 7 9
```

```
#### Subtração
```

```
a - b
```

```
## [1] 3 3 3
```

## ► Multiplicação por escalar

```
alpha = 10
```

```
alpha*a
```

```
## [1] 40 50 60
```

## ► Produto de Hadamard

```
a*b
```

```
## [1] 4 10 18
```

## ► Produto vetorial

```
a%*%b
```

```
##      [,1]
```

```
## [1,]    32
```

## ► Co-seno do ângulo entre dois vetores

```
cos <- t(a)%*%b/(sqrt(t(a)%*%a)*sqrt(t(b)%*%b))
```

## ► Lei da reciclagem

```
a <- c(4,5,6,5,6,7)
```

```
b <- c(1,2,3)
```

```
a + b
```

```
## [1] 5 7 9 6 8 10
```

# Matrizes



# Matriz

- ▶ Uma **matriz** é um arranjo retangular ou quadrado de números ou variáveis.
- ▶ Uma matriz  $(n \times m)$  tem  $n$  linhas e  $m$  colunas:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \ddots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \dots & \dots & a_{nm} \end{pmatrix}.$$

- ▶ O primeiro subscrito representa **linha** e o segundo representa **coluna**.
- ▶ A **dimensão** de uma matriz é o seu número de linhas e colunas.
- ▶ Duas matrizes são iguais se tem a mesma dimensão e se os elementos das correspondentes posições são iguais.

# Matriz transposta

- A operação de transposição rearranja uma matriz de forma que suas linhas são transformadas em colunas e vice-versa.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^{\top} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}.$$

- Note que  $(A^{\top})^{\top} = A$ .

- Computacionalmente
- Declarando matrizes

```
a <- c(1,2,3,4,5,6)
A <- matrix(a, nrow = 3, ncol = 2)
A
```

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

- O *default* preenche por colunas.
- Transposta de uma matriz

```
t(A)
```

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

# Operações com matrizes

# Operações com matrizes

- Multiplicação matriz por escalar.

$$\alpha A = \begin{pmatrix} \alpha a_{11} & \alpha a_{12} & \dots & \alpha a_{1m} \\ \alpha a_{21} & \alpha a_{22} & \ddots & \alpha a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{n1} & \dots & \dots & \alpha a_{nm} \end{pmatrix}.$$

- Computacionalmente

```
A <- matrix(c(1,2,3,4,5,6),  
            nrow = 3, ncol = 2)
```

```
alpha <- 10
```

```
alpha*A
```

```
##      [,1] [,2]  
## [1,]   10  40  
## [2,]   20  50  
## [3,]   30  60
```

# Operações com matrizes

- Duas matrizes podem ser somadas ou subtraídas somente se tiverem o mesmo tamanho.

1. Soma  $c_{ij} = a_{ij} + b_{ij}$ .
2. Subtração  $c_{ij} = a_{ij} - b_{ij}$ .

- Exemplo

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} + \begin{pmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{pmatrix} = \begin{pmatrix} 11 & 22 \\ 33 & 44 \\ 55 & 66 \end{pmatrix}.$$

- Soma de duas matrizes

```
A <- matrix(c(1,2,3,4,5,6),  
             nrow = 3, ncol = 2)  
B <- matrix(c(10,20,30,40,50,60),  
             nrow = 3, ncol = 2)  
  
C = A + B  
C
```

```
##      [,1] [,2]  
## [1,]   11  44  
## [2,]   22  55  
## [3,]   33  66
```

## Operações com matrizes

- Condição para multiplicar matrizes

$$\underset{m \times n}{C} = \underset{m \times q}{A} \underset{q \times n}{B}.$$

- Cada elemento  $c_{ij} = \sum_{k=1}^q a_{ik} b_{kj}$ .

$$\begin{pmatrix} 2 & -1 \\ 8 & 3 \\ 6 & 7 \end{pmatrix} \begin{pmatrix} 4 & 9 & 1 & -3 \\ -5 & 2 & 4 & 6 \end{pmatrix} =$$

$$\begin{pmatrix} ((2 \cdot 4) + (-1 \cdot -5)) & ((2 \cdot 9) + (-1 \cdot 2)) & ((2 \cdot 1) + (-1 \cdot 4)) & ((2 \cdot -3) + (-1 \cdot 6)) \\ ((8 \cdot 4) + (3 \cdot -5)) & ((8 \cdot 9) + (3 \cdot 2)) & ((8 \cdot 1) + (3 \cdot 4)) & ((8 \cdot -3) + (3 \cdot 6)) \\ ((6 \cdot 4) + (7 \cdot -5)) & ((6 \cdot 9) + (7 \cdot 2)) & ((6 \cdot 1) + (7 \cdot 4)) & ((6 \cdot -3) + (7 \cdot 6)) \end{pmatrix} =$$

$$\begin{pmatrix} 13 & 16 & -2 & -12 \\ 17 & 78 & 20 & -6 \\ -11 & 68 & 34 & 24 \end{pmatrix}.$$

# Operações com matrizes

► Computacionalmente.

► Matrizes compatíveis

```
A <- matrix(c(2,8,6,-1,3,7),  
            nrow = 3, ncol = 2)  
B <- matrix(c(4,-5,9,2,1,4,-3,6),  
            nrow = 2, ncol = 4)  
C = A%*%B  
C
```

```
##      [,1] [,2] [,3] [,4]  
## [1,]  13  16  -2  -12  
## [2,]  17  78  20  -6  
## [3,] -11  68  34  24
```

► Matrizes não compatíveis

```
B %*% A
```

```
## Error in B %*% A: argumentos não compatíveis
```

# Produto de Hadamard

## ► Produto simples ou de Hadamard

$$A \odot B = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1m}b_{1m} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2m}b_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \cdots & a_{nm}b_{nm} \end{pmatrix}$$

## ► Computacionalmente

```
A <- matrix(c(1,2,3,4),  
            nrow = 2, ncol = 2)  
B <- matrix(c(10,20,30,40),  
            nrow = 2, ncol = 2)  
A*B  
##      [,1] [,2]  
## [1,]   10  90  
## [2,]   40 160
```



## Propriedades envolvendo operações com matrizes

► Sendo  $A, B, C$  e  $D$  compatíveis temos,

1.  $A + B = B + A$ .
2.  $(A + B) + C = A + (B + C)$ .
3.  $\alpha(A + B) = \alpha A + \alpha B$ .
4.  $(\alpha + \beta)A = \alpha A + \beta A$ .
5.  $\alpha(AB) = (\alpha A)B = A(\alpha B)$ .
6.  $A(B \pm C) = AB \pm AC$ .
7.  $(A \pm B)C = AC \pm BC$ .
8.  $(A - B)(C - D) = AC - BC - AD + BD$ .

► Propriedades envolvendo transposta e multiplicação

1. Se  $A$  é  $n \times m$  e  $B$  é  $m \times n$ , então

$$(AB)^T = B^T A^T.$$

2. Se  $A, B$  e  $C$  são compatíveis

$$(ABC)^T = C^T B^T A^T.$$

# Matrizes de formas especiais

## Matrizes de formas especiais

### ► Matriz quadrada

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}.$$

- $a_{ii}$  são os elementos da **diagonal**.
- $a_{ij}$  para  $i \neq j \rightarrow$  **fora da diagonal**.
- $a_{ij}$  para  $j > i \rightarrow$  **acima da diagonal**.
- $a_{ij}$  para  $i > j \rightarrow$  **abaixo da diagonal**.

### ► Matriz diagonal

$$D = \begin{pmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{pmatrix}.$$

### ► Matriz identidade

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

## Matrizes de formas especiais

### ► Triangular superior

$$U = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}.$$

### ► Triangular inferior

$$L = \begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}.$$

### ► Matriz nula

$$0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

### ► Matriz quadrada simétrica

$$A = \begin{pmatrix} 1 & 0.8 & 0.6 & 0.4 \\ 0.8 & 1 & 0.2 & 0.4 \\ 0.6 & 0.2 & 1 & 0.1 \\ 0.4 & 0.4 & 0.1 & 1 \end{pmatrix}.$$

## Combinações lineares

- ▶ Um conjunto de vetores  $a_1, a_2, \dots, a_n$  é dito ser **linearmente dependente** se puderem ser encontrados escalares  $c_1, c_2, \dots, c_n$  e estes escalares não sejam todos iguais a 0 de tal forma que

$$c_1 a_1 + c_2 a_2 + \dots + c_n a_n = 0.$$

- ▶ Caso contrário é dito ser **linearmente independente**.
- ▶ Notação matricial

$$Ac = 0.$$

- ▶ As colunas de  $A$  são linearmente independentes se  $Ac = 0$  implicar que  $c = 0$ .

# *Rank* e inversa de uma matriz

## Rank ou posto de uma matriz

- ▶ O *rank* ou **posto** de qualquer matriz quadrada ou retangular  $A$  é definido como
$$\text{rank}(A) = \text{número de colunas ou linhas linearmente independentes em } A.$$
- ▶ Sendo  $A$  uma matriz retangular  $n \times m$  o maior *rank* possível para  $A$  é o  $\min(n, m)$ .
- ▶ O rank da matrix nula é 0.
- ▶ Se o *rank* da matriz é o  $\min(n, m)$  dizemos que a matriz tem *rank* completo.

## Matriz não singular e matriz inversa

- ▶ Uma matriz quadrada de **posto completo** é chamada de **não singular**.
- ▶ Sendo  $A$  quadrada de posto completo a **matriz inversa** de  $A$  é única tal que

$$AA^{-1} = I.$$

- ▶ Não quadrada (posto incompleto)  $\rightarrow$  não terá inversa e é dita ser **singular**.
- ▶ Note que  $(A^{-1})^{-1} = A$ .



## ► Computacionalmente

```
A <- matrix(c(4, 2, 7, 6), 2, 2)
A_inv <- solve(A)
A_inv
```

```
##      [,1] [,2]
## [1,]  0.6 -0.7
## [2,] -0.2  0.4
```

## ► Verificando

```
A%*%A_inv
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

## ► Propriedades envolvendo inversas

1. Se  $A$  é não singular, então  $A^\top$  é não singular e sua inversa é dada por

$$(A^\top)^{-1} = (A^{-1})^\top.$$

2. Se  $A$  e  $B$  são matrizes não singulares de mesmo tamanho, então o produto  $AB$  é não singular e

$$(AB)^{-1} = B^{-1}A^{-1}.$$

## Inversa generalizada

- A **inversa generalizada** de uma matriz  $A$   $n \times p$  é qualquer matriz  $A^-$  que satisfaça

$$AA^- = A.$$

- Não é única exceto quando  $A$  é não-singular (inversa usual).
- Exemplo,

$$a = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

- $a^- = (1, 0, 0, 0)$

- Verificando

```
a <- matrix(c(1, 2, 3, 4), 4, 1)
a_inv <- matrix(c(1,0,0,0), 1, 4)
a%%a_inv%%a
```

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

- Moore-Penrose *generalized inverse*

```
#### Matriz singular (col 3 = col 2 + col 1)
A <- matrix(c(2, 1, 3, 2, 0,
              2, 3, 1, 4), 3, 3)
library(MASS)
A_ginv <- ginv(A)
A%%A_ginv%%A ## Verificando
```

# Matrizes positivas definidas

## Formas quadráticas

- ▶ Soma de quadrados são importantes em ciência de dados.
- ▶ Considere uma matriz  $A$  simétrica e  $y$  um vetor, o produto

$$y^\top A y = \sum_i a_{ii} y_i^2 + \sum_{i \neq j} a_{ij} y_i y_j,$$

é chamado de **forma quadrática**.

- ▶ Sendo  $y$  de dimensão  $n \times 1$ ,  $y^\top I y = y_1^2 + y_2^2 + \dots, y_n^2$ .
- ▶ Consequentemente,  $y^\top y$  é a soma de quadrados dos elementos do vetor  $y$ .
- ▶ A raiz quadrada da soma de quadrados é o comprimento de  $y$ .

## Matriz positiva definida

- ▶ Sendo  $A$  uma matriz simétrica com a propriedade  $y^T A y > 0$  para todos os possíveis  $y$  exceto para quando  $y = 0$ , então a forma quadrática  $y^T A y$  é chamada **positiva definida**, e  $A$  é dita ser uma **matriz positiva definida**.
- ▶ Exemplo

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}.$$

A forma quadrática associada é dada por

$$y^T A y = (y_1 \ y_2) \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = 2y_1^2 - 2y_1y_2 + 3y_2^2,$$

que é claramente positiva, desde que  $y_1$  e  $y_2$  sejam diferentes de zero.

## Propriedades de matrizes positivas definidas

1. Se  $A$  é positiva definida, então todos os valores da diagonal de  $A$  são positivos.
2. Se  $A$  é positiva semi-definida, então os elementos da diagonal de  $A$  são maiores ou iguais a zero.
3. Sendo  $P$  uma matriz não-singular e  $A$  uma matriz positiva definida, o produto  $P^T A P$  é positiva definida.
4. Sendo  $P$  uma matriz não-singular e  $A$  uma matriz positiva semi-definida, o produto  $P^T A P$  é positiva semi-definida.
5. Uma matriz positiva definida é não-singular.

# Determinante e traço de uma matriz

## Determinante de uma matriz

- O determinante de uma matriz  $A$  é o escalar

$$|A| = \sum_j (-1)^k a_{1j_1} a_{2j_2} \cdots a_{nj_n},$$

onde a soma é realizada para todas as  $n!$  permutações de grau  $n$ , e  $k$  é o número de mudanças necessárias para que os segundos subscritos sejam colocados na ordem  $1, 2, \dots, n$ .

- Considere a matriz

$$A = \begin{pmatrix} 3 & -2 \\ -2 & 4 \end{pmatrix}.$$

$$|A| = (-1)^0 a_{11} a_{22} + (-1)^1 a_{12} a_{21} = 1 \cdot (3 \cdot 4) - (-2 \cdot -2) = 12 - 4 = 8.$$



# Determinante de uma matriz

## ► Computacionalmente.

```
A <- matrix(c(3,-2,-2,4),2,2)
determinant(A, logarithm = FALSE)$modulus
## [1] 8
## attr("logarithm")
## [1] FALSE
```

## ► Determinante em escala log.

```
determinant(A, logarithm = TRUE)$modulus
## [1] 2.079442
## attr("logarithm")
## [1] TRUE
```

## ► Alguns aspectos interessantes sobre determinantes são:

1. Se  $A$  é singular,  $|A| = 0$ .
2. Se  $A$  é não singular,  $|A| \neq 0$ .
3. Se  $A$  é positiva definida,  $|A| > 0$ .
4.  $|A^T| = |A|$ .
5. Se  $A$  é não singular,  $|A^{-1}| = \frac{1}{|A|}$ .

## Traço de uma matriz

- ▶ O **traço** de uma matriz  $A$   $n \times n$  é um escalar definido como a soma dos elementos da diagonal,  $\text{tr}(A) = \sum_{i=1}^n a_{ii}$ .

- ▶ Propriedades

1. Se  $A$  e  $B$  são  $n \times n$ , então

$$\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B).$$

2. Se  $A$  é  $n \times p$  e  $B$  é  $p \times n$ , então

$$\text{tr}(AB) = \text{tr}(BA).$$

- ▶ Computacionalmente

```
A <- matrix(c(3,-2,-2,4),2,2)
sum(diag(A))
```

```
## [1] 7
```

# Cálculo vetorial e matricial

## Cálculo vetorial

- ▶ Seja  $y = f(\mathbf{x})$  uma função das variáveis  $x_1, x_2, \dots, x_p$  e  $\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_p}$  as respectivas derivadas parciais.
- ▶ Assim,

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_p} \end{pmatrix}.$$

## Cálculo vetorial

► Sendo  $\mathbf{a}^\top = (a_1, a_2, \dots, a_p)$  um vetor de constantes e  $A$  uma matriz simétrica de constantes.

1. Seja  $y = \mathbf{a}^\top \mathbf{x} = \mathbf{x}^\top \mathbf{a}$ . Então,

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x}^\top \mathbf{a})}{\partial \mathbf{x}} = \mathbf{a}.$$

2. Seja  $y = \mathbf{x}^\top A \mathbf{x}$ . Então,

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x}^\top A \mathbf{x})}{\partial \mathbf{x}} = 2A\mathbf{x}.$$

- Se  $y = f(\mathbf{X})$  onde  $\mathbf{X}$  é uma matriz  $p \times p$ . As derivadas parciais de  $y$  em relação a cada  $x_{ij}$  são organizadas em uma matriz.

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \cdots & \frac{\partial y}{\partial x_{1p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{p1}} & \cdots & \frac{\partial y}{\partial x_{pp}} \end{pmatrix}.$$

## Cálculo Matricial

► Algumas derivadas importantes envolvendo matrizes são apresentadas abaixo.

1. Seja  $y = \text{tr}(\mathbf{XA})$  sendo  $\mathbf{X}$   $p \times p$  e definida positiva e  $\mathbf{A}$   $p \times p$  constantes. Então,

$$\frac{\partial y}{\partial \mathbf{X}} = \frac{\partial \text{tr}(\mathbf{XA})}{\partial \mathbf{X}} = \mathbf{A} + \mathbf{A}^\top - \text{diag}(\mathbf{A}).$$

2. Sendo  $\mathbf{A}$  não singular com derivadas  $\frac{\partial \mathbf{A}}{\partial x}$ . Então,

$$\frac{\partial \mathbf{A}^{-1}}{\partial x} = -\mathbf{A}^{-1} \frac{\partial \mathbf{A}}{\partial x} \mathbf{A}^{-1}.$$

3. Sendo  $\mathbf{A}$   $n \times n$  positiva definida. Então,

$$\frac{\partial \log |\mathbf{A}|}{\partial x} = \text{tr} \left( \mathbf{A}^{-1} \frac{\partial \mathbf{A}}{\partial x} \right).$$

# Regressão linear múltipla



## Regressão linear múltipla: especificação usual

- ▶ Regressão linear simples

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i.$$

- ▶ Regressão linear múltipla

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon_i.$$

- ▶ Modelo para cada observação

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_p x_{1p} + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_p x_{2p} + \epsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_p x_{np} + \epsilon_n \end{aligned}$$

# Regressão linear múltipla: especificação matricial

## ► Notação matricial

$$\begin{matrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\ n \times 1 \end{matrix} = \begin{matrix} \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix} \\ n \times p \end{matrix} \begin{matrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \\ p \times 1 \end{matrix} + \begin{matrix} \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \\ n \times 1 \end{matrix}$$

## ► Notação mais compacta

$$\begin{matrix} y \\ n \times 1 \end{matrix} = \begin{matrix} \mathbf{X} \\ n \times p \end{matrix} \begin{matrix} \beta \\ p \times 1 \end{matrix} + \begin{matrix} \epsilon \\ n \times 1 \end{matrix}.$$

## Regressão linear múltipla: estimação (treinamento)

- Objetivo: encontrar o vetor  $\hat{\beta}$ , tal que

$$SQ(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta),$$

seja a menor possível.

## Regressão linear múltipla: estimação

1. Passo 1: encontrar o vetor gradiente. Derivando em  $\beta$ , temos

$$\begin{aligned}\frac{\partial SQ(\beta)}{\partial \beta} &= \frac{\partial}{\partial \beta} (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) \\ &= \frac{\partial}{\partial \beta} ((\mathbf{y} - \mathbf{X}\beta)^\top) (\mathbf{y} - \mathbf{X}\beta) + (\mathbf{y} - \mathbf{X}\beta)^\top \frac{\partial}{\partial \beta} (\mathbf{y} - \mathbf{X}\beta) \\ &= -\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\beta) + (\mathbf{y} - \mathbf{X}\beta)^\top (-\mathbf{X}) \\ &= -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\beta).\end{aligned}$$

## Regressão linear múltipla: estimação

2. Passo 2: resolver o sistema de equações lineares

$$\begin{aligned}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\hat{\beta}) &= 0 \\ \mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{X}\hat{\beta} &= 0 \\ \mathbf{X}^\top\mathbf{X}\hat{\beta} &= \mathbf{X}^\top\mathbf{y} \\ \hat{\beta} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}\end{aligned}$$

# Regressão linear múltipla: exemplo

## Regressão linear múltipla: exemplo

- ▶ Conjunto de dados Boston disponível no pacote MASS.
- ▶ Cinco primeiras **covariáveis** disponíveis:
  - ▶ crim: taxa de crimes per capita.
  - ▶ zn: proporção de terrenos residenciais zoneados para lotes com mais de 25.000 pés quadrados.
  - ▶ indus: proporção de acres de negócios não varejistas por cidade.
  - ▶ chas: variável dummy de Charles River (1 se a área limita o rio; 0 caso contrário).
  - ▶ nox: concentração de óxido de nitrogênio (parte por 10 milhões).
- ▶ **Variável resposta:** medv valor mediano das casas ocupadas em \$1000.

# Regressão linear múltipla: implementação computacional

## ► Carregando a base de dados

```
require(MASS)

## Carregando pacotes exigidos: MASS

data(Boston)
head(Boston[, c(1:5,14)])
```

##	crim	zn	indus	chas	nox	medv
## 1	0.00632	18	2.31	0	0.538	24.0
## 2	0.02731	0	7.07	0	0.469	21.6
## 3	0.02729	0	7.07	0	0.469	34.7
## 4	0.03237	0	2.18	0	0.458	33.4
## 5	0.06905	0	2.18	0	0.458	36.2
## 6	0.02985	0	2.18	0	0.458	28.7

## ► Matriz de delineamento (X).

```
X <- model.matrix(~ crim + zn + indus +  
                  chas + nox, data = Boston)
```

```
head(X)
```

##	(Intercept)	crim	zn	indus	chas	nox
## 1	1	0.00632	18	2.31	0	0.538
## 2	1	0.02731	0	7.07	0	0.469
## 3	1	0.02729	0	7.07	0	0.469
## 4	1	0.03237	0	2.18	0	0.458
## 5	1	0.06905	0	2.18	0	0.458
## 6	1	0.02985	0	2.18	0	0.458

## ► Variável resposta

```
y <- Boston$medv
```



# Regressão linear múltipla: implementação computacional

- ▶ Estimadores de mínimos quadrados:

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

- ▶ Computacionalmente: versão ingênua

```
round(solve(t(X)%*%X)%*%t(X)%*%y, 2)
```

```
##           [,1]
## (Intercept) 29.49
## crim       -0.22
## zn         0.06
## indus      -0.38
## chas       7.03
## nox       -5.42
```

- ▶ Computacionalmente: versão eficiente

```
round(solve(t(X)%*%X, t(X)%*%y), 2)
```

```
##           [,1]
## (Intercept) 29.49
## crim       -0.22
## zn         0.06
## indus      -0.38
## chas       7.03
## nox       -5.42
```

- ▶ Função nativa do R

```
t(round(coef(lm(medv ~ crim + zn + indus +
               chas + nox, data = Boston)), 2))
```

```
##           (Intercept)  crim   zn indus chas   nox
## [1,]                29.49 -0.22 0.06 -0.38 7.03 -5.42
```

# Matrizes esparsas

## Matrizes esparsas (tópico adicional)

- ▶ Matrizes aparecem em todos os tipos de aplicação em ciência de dados.
- ▶ Modelos estatísticos, *machine learning*, análise de texto, análise de *cluster*, etc.
- ▶ Muitas vezes as matrizes usadas têm uma grande quantidade de zeros.
- ▶ Quando uma matriz tem uma quantidade considerável de zeros, dizemos que ela é **esparsa**, caso contrário dizemos que a matriz é **densa**.
- ▶ Todas as propriedades que vimos para matrizes em geral valem para matrizes esparsas.
- ▶ O R tem um conjunto de métodos altamente eficiente por meio do pacote `Matrix`.
- ▶ Saber que uma matriz é esparsa é útil pois permite:
  - ▶ Planejar formas de armazenar a matriz em memória.
  - ▶ Economizar cálculos em algoritmos numéricos (multiplicação, inversa, determinante, decomposições, etc).

# Matrizes esparsas

- Comparando a quantidade de memória utilizada.

```
library('Matrix')  
m1 <- matrix(0, nrow = 1000, ncol = 1000)  
m2 <- Matrix(0, nrow = 1000, ncol = 1000, sparse = TRUE)  
object.size(m1)
```

```
## 8000216 bytes
```

```
object.size(m2)
```

```
## 9240 bytes
```

# Comparando o tempo computacional

## ► Matriz esparsa

```
y <- rnorm(1000)
X <- Matrix(NA, ncol = 100, nrow = 1000)
for(i in 1:1000) {X[i,] <- rbinom(100, size = 1, p = 0.1)}
X <- Matrix(X, sparse = TRUE)
system.time(replicate(100, solve(t(X)%*%X, t(X)%*%y)))
```

```
##      usuário      sistema decorrido
##      0.223      0.000      0.224
```

## ► Matriz densa

```
y <- rnorm(1000)
X <- matrix(NA, ncol = 100, nrow = 1000)
for(i in 1:1000) {X[i,] <- rbinom(100, size = 1, p = 0.1)}
system.time(replicate(100, solve(t(X)%*%X, t(X)%*%y)))
```

```
##      usuário      sistema decorrido
##      0.819      0.004      0.823
```

## Diferentes formas de implementar as operações matriciais

- Criando a base de dados para a comparação

```
library(Matrix)
n <- 10000; p <- 500
x <- matrix(rbinom(n*p, 1, 0.01), nrow=n, ncol=p)
X <- Matrix(x)
object.size(x)
```

```
## 20000216 bytes
```

```
object.size(X)
```

```
## 600432 bytes
```

# Diferentes formas de implementar as operações matriciais

## ► Diferentes implementações

```
y <- rnorm(n)
system.time(solve(t(x)%*%x, t(x)%*%y))
```

```
##  usuário  sistema decorrido
##    2.053    0.040    2.094
```

```
system.time(solve(crossprod(x), crossprod(x, y)))
```

```
##  usuário  sistema decorrido
##    1.731    0.016    1.748
```

```
system.time(solve(t(X)%*%X, t(X)%*%y))
```

```
##  usuário  sistema decorrido
##    0.071    0.000    0.072
```

```
system.time(solve(crossprod(X), crossprod(X,y)))
```

```
##  usuário  sistema decorrido
##    0.029    0.000    0.050
```

## Pacote adicional glmnet

- Implementação eficiente do modelo de regressão linear múltipla.

```
library(glmnet)
```

```
## Loaded glmnet 4.1-6
```

```
system.time(b <- coef(lm(y~x)))
```

```
##      usuário      sistema decorrido  
##      2.389      0.044      2.434
```

```
system.time(g1 <- glmnet(x, y, nlambda=1, lambda=0, standardize=FALSE))
```

```
##      usuário      sistema decorrido  
##      0.065      0.020      0.086
```

```
system.time(g2 <- glmnet(X, y, nlambda=1, lambda=0, standardize=FALSE))
```

```
##      usuário      sistema decorrido  
##      0.006      0.000      0.006
```



# Sistemas lineares

## Sistemas lineares

- ▶ 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

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

- ▶ Genericamente, tem-se

$$f(x) = 0.$$

- ▶ Equações podem ser lineares ou não-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:  $\hat{x}_1 = 6$  e  $\hat{x}_2 = 1$ .
- ▶ Resolver (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:

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

- De forma geral, tem-se

$$Ax = b.$$

## Operações com linhas

► Sem qualquer alteração na relação linear, é possível

1. Trocar a posição de linhas:

$$4x_1 + 5x_2 = 29$$

$$7x_1 + 3x_2 = 45.$$

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

$$x_1 + \frac{5}{4}x_2 = \frac{29}{4} \quad (1)$$

$$7x_1 + 3x_2 = 45. \quad (2)$$

## Operações com linhas

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

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

► 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:

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

- Matricialmente, tem-se

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

- Métodos diretos e métodos iterativos.

# Métodos diretos



## 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

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

## 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

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

► Diagonal:

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

# Eliminação de Gauss

## 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 regressiva 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.

$$\left[ \begin{array}{ccc|c} [3] & 2 & 6 & 24 \\ 2 - \frac{2}{3}3 & 4 - \frac{2}{3}2 & 3 - \frac{2}{3}6 & 23 - \frac{2}{3}24 \\ 5 - \frac{5}{3}3 & 3 - \frac{5}{3}2 & 4 - \frac{5}{3}6 & 33 - \frac{5}{3}24 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} [3] & 2 & 6 & 24 \\ 0 & \frac{8}{3} & -1 & 7 \\ 0 & -\frac{1}{3} & -6 & -7 \end{array} \right]$$

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

$$\left[ \begin{array}{ccc|c} 3 & 2 & 6 & 24 \\ 0 & [\frac{8}{3}] & -1 & 7 \\ 0 & -\frac{1}{3} - (-\frac{3}{24})(\frac{8}{3}) & -6 - (-\frac{3}{24})(-1) & -7 - (-\frac{3}{24})(7) \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 3 & 2 & 6 & 24 \\ 0 & [\frac{8}{3}] & -1 & 7 \\ 0 & 0 & -\frac{147}{24} & -\frac{147}{24} \end{array} \right]$$

- Passo 3: substituição regressiva.

## Métodos diretos: Eliminação de Gauss

- ▶ Usando a fórmula de substituição regressiva temos:
  - ▶  $x_3 = \frac{b_3}{a_{33}} = 1.$
  - ▶  $x_2 = \frac{b_2 - a_{23}x_3}{a_{22}} = 3.$
  - ▶  $x_1 = \frac{(b_1 - (a_{12}x_2 + a_{13}x_3))}{a_{11}} = 4.$
- ▶ 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:
  - ▶ O elemento pivô é zero.
  - ▶ O elemento pivô é pequeno em relação aos demais termos.

# Eliminação de Gauss com pivotação



## Eliminação de Gauss com pivotação

- Considere o sistema

$$0x_1 + 2x_2 + 3x_2 = 46$$

$$4x_1 - 3x_2 + 2x_3 = 16$$

$$2x_1 + 4x_2 - 3x_3 = 12$$

- 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.

$$4x_1 - 3x_2 + 2x_3 = 16$$

$$2x_1 + 4x_2 - 3x_3 = 12$$

$$0x_1 + 2x_2 + 3x_2 = 46$$

## 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.

## Passo 1: obtendo uma matriz triangular superior.

```
gauss <- function(A, b) {  
  Ae <- cbind(A, b) ## Sistema aumentado  
  rownames(Ae) <- paste0("x", 1:length(b))  
  n_row <- nrow(Ae)  
  n_col <- ncol(Ae)  
  SOL <- matrix(NA, n_row, n_col) ## Matriz para receber os resultados  
  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)  
}
```

## 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_col]  
  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)  
}
```

## 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)
S <- gauss(A, b) ## Passo 1: Triangularização
sol = sub_reg(SOL = S) ## Passo 2: Substituição regressiva
sol
```

```
## [1] 4 3 1
```

```
A*%sol ## Verificando a solução
```

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

# Eliminação de Gauss com pivotação

- Resolva o seguinte sistema usando Eliminação de Gauss com pivotação.

$$0x_1 + 2x_2 + 3x_3 = 46$$

$$4x_1 - 3x_2 + 2x_3 = 16$$

$$2x_1 + 4x_2 - 3x_3 = 12$$

## Entrando com o sistema original

```
A <- matrix(c(0,4,2,2,-3,4,3,2,-3), 3,3)
```

```
b <- c(46,16,12)
```

## Pivoteamento

```
A_order <- A[order(A[,1], decreasing = TRUE),]
```

```
b_order <- b[order(A[,1], decreasing = TRUE)]
```

#### Triangulação

```
S <- gauss(A_order, b_order)
```

```
S
```

```
##      [,1] [,2]      [,3]      [,4]
## [1,]    4 -3.0  2.000000 16.00000
## [2,]    0 5.5 -4.000000  4.00000
## [3,]    0 0.0  4.454545 44.54545
```

#### Substituição regressiva

```
sol <- sub_reg(SOL = S)
```

```
sol
```

```
## [1]  5  8 10
```

#### Solução

```
A_order%*%sol
```

```
##      [,1]
## [1,]    5
## [2,]    8
## [3,]   10
```

# Eliminação de Gauss-Jordan

## 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 operações de linha.

- ▶ O método de Gauss-Jordan pode ser combinado com pivotação igual ao método de eliminação de Gauss.



# Métodos iterativos

## Métodos iterativos

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

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

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

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

## Métodos iterativos

### ► Atualização: Método de Gauss-Seidel

$$x_1^{k+1} = \frac{1}{a_{11}} \left[ b_1 - \sum_{j=2}^{j=n} a_{1j} x_j^{(k)} \right],$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[ b_i - \left( \sum_{j=1}^{j=i-1} a_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{j=n} a_{ij} x_j^{(k)} \right) \right] \quad i = 2, 3, \dots, n-1 \quad \text{e}$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}} \left[ b_n - \sum_{j=1}^{j=n-1} a_{nj} x_j^{(k+1)} \right].$$

# 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) { ##### Equação de atualização  
    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 ##### Critério de parada  
  }  
  return(list("Solucao" = x, "Iteracoes" = x_temp))  
}
```

## Método iterativo de Jacobi

- Resolva o seguinte sistema de equações lineares usando o método de Jacobi.

$$9x_1 - 2x_2 + 3x_3 + 2x_4 = 54.5$$

$$2x_1 + 8x_2 - 2x_3 + 3x_4 = -14$$

$$-3x_1 + 2x_2 + 11x_3 - 4x_4 = 12.5$$

$$-2x_1 + 3x_2 + 2x_3 - 10x_4 = -21$$

- Computacionalmente

```
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
```

## Métodos 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.
- ▶ Rlinsolve é implementado em C++ usando o pacote Rcpp.

```
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)
## pacote extra
require(Rlinsolve)
lsolve.jacobi(A, b)$x ## Método de jacobi
```

```
##           [,1]
## [1,]  4.9999708
## [2,] -2.0000631
## [3,]  2.5000163
## [4,] -0.9999483
```

```
lsolve.gs(A, b)$x ## Método de Gauss-Seidel
```

```
##           [,1]
## [1,]  4.999955
## [2,] -2.000071
## [3,]  2.500018
## [4,] -0.999968
```

# Decomposição LU

## Decomposição LU

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

$$Ax = b.$$

- ▶ Sendo dois sistemas

$$Ax = b_1, \quad \text{e} \quad Ax = b_2.$$

- ▶ Cálculos do primeiro não ajudam a resolver o segundo.
- ▶ IDEAL! - Operações realizadas em  $A$  fossem dissociadas das operações em  $b$ .



## Decomposição LU

- Suponha que precisamos resolver vários sistemas do tipo

$$Ax = b.$$

para diferentes  $b'$ s.

- Opção 1 - calcular a inversa  $A^{-1}$ , assim a solução

$$x = A^{-1}b.$$

- Cálculo da inversa é computacionalmente ineficiente.

## Decomposição LU: algoritmo

- ▶ Decomponha (fatore) a matriz  $A$  em um produto de duas matrizes

$$A = LU,$$

onde  $L$  é triangular inferior e  $U$  é triangular superior.

- ▶ Baseado na decomposição o sistema tem a forma:

$$LUx = b. \tag{3}$$

- ▶ Defina  $Ux = y$ .

- ▶ Substituindo em 3 tem-se

$$Ly = b. \tag{4}$$

- ▶ Solução é obtida em dois passos

- ▶ Resolva Eq.(4) para obter  $y$  usando substituição progressiva.
- ▶ Resolva Eq.(3) para obter  $x$  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.

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

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

- Neste caso, tem-se

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

# Decomposição LU com pivotação

## 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

$$PA = LU.$$

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

$$Ax = b,$$

então a ordem das linhas de  $b$  deve ser alterada de forma consistente com a pivotação, i.e.  $Pb$ .

## 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)  
  SOL <- matrix(NA, n_row, n_col) ## Matriz para receber os resultados  
  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] [,4]  
## [1,] 1.0000000 0.0000000 0.0000000 0  
## [2,] 0.2222222 1.0000000 0.0000000 0  
## [3,] -0.3333333 0.1578947 1.0000000 0  
## [4,] -0.2222222 0.3026316 0.279661 1  
##  
## $U  
##      [,1]      [,2]      [,3]      [,4]  
## [1,] 9 -2.000000e+00 3.000000 2.000000  
## [2,] 0 8.444444e+00 -2.666667 2.555556  
## [3,] 0 0.000000e+00 12.421053 -3.736842  
## [4,] 0 -4.440892e-16 0.000000 10.716102
```

```
LU$L %*% LU$U ## Verificando a solução
```

```
##      [,1] [,2] [,3] [,4]  
## [1,] 9 -2 3 2  
## [2,] 2 8 -2 3  
## [3,] -3 2 11 -4  
## [4,] -2 3 2 10
```



## 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]  5.0 -2.0  2.5 -1.0

A%%x ## Verificando a solução

##      [,1]
## [1,]  54.5
## [2,] -14.0
## [3,]  12.5
## [4,] -21.0
```

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

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

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

# Obtendo a inversa

## Obtendo a inversa via decomposição LU

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

$$AA^{-1} = I.$$

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

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

- ▶ Três sistemas de equações diferentes, em cada sistema, uma coluna da matriz  $X$  é a incógnita.

## 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

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

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

## 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>.

# Autovalores e autovetores

## Autovalores e autovetores

- ▶ Redução de dimensionalidade é fundamental em ciência de dados.
- ▶ Análise de componentes principais (PCA)
- ▶ Análise fatorial (AF).
- ▶ Decompor grandes e complicados relacionamentos multivariados em simples componentes não relacionados.
- ▶ Vamos discutir apenas os aspectos matemáticos.



## Intuição

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

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

- É mais fácil interpretar o tamanho de um vetor enquanto ignorando a sua direção e vice-versa.
- Esta ideia pode ser estendida para matrizes.
- Uma matriz nada mais é do que um conjunto de vetores.
- IDEIA – 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

$$Av = \lambda v. \quad (5)$$

- ▶ Os vetores  $v$ 's que satisfazem Eq. (5) são os autovetores.
- ▶ Os valores  $\lambda$ 's que satisfazem Eq. (5) são os autovalores.
- ▶ Vamos considerar o caso em que  $A$  é simétrica.
- ▶ A ideia pode ser estendida para matrizes não simétricas.

## Autovalores e Autovetores

- ▶ Se  $A$  é uma matriz simétrica  $n \times n$ , então existem exatamente  $n$  pares  $(\lambda_j, v_j)$  que satisfazem a equação:

$$Av = \lambda v.$$

- ▶ Se  $A$  tem autovalores  $\lambda_1, \dots, \lambda_n$ , então:
- ▶  $\text{tr}(A) = \sum_{i=1}^n \lambda_i$ .
- ▶  $\det(A) = \prod_{i=1}^n \lambda_i$ .
- ▶  $A$  é positiva definida, se e somente se todos  $\lambda_j > 0$ .
- ▶  $A$  é semi-positiva definida, se e somente se todos  $\lambda_j \geq 0$ .
- ▶ A ideia do PCA é decompor/fatorar a matriz  $A$  em componentes mais simples de interpretar.

## Decomposição em autovalores e autovetores

- ▶ Teorema: qualquer matriz simétrica  $A$  pode ser fatorada em

$$A = Q\Lambda Q^T,$$

onde  $\Lambda$  é diagonal contendo os autovalores de  $A$  e as colunas de  $Q$  contêm os autovetores ortonormais.

- ▶ Vetores ortonormais: são mutuamente ortogonais e de comprimento unitário.
- ▶ Teorema: se  $A$  tem autovetores  $Q$  e autovalores  $\lambda_j$ . Então  $A^{-1}$  tem autovetores  $Q$  e autovalores  $\lambda_j^{-1}$ .
- ▶ Implicação: se  $A = Q\Lambda Q^T$  então  $A^{-1} = Q\Lambda^{-1}Q^T$ .

# Diagonalização

- ▶ Autovalores são úteis porque eles permitem lidar com matrizes da mesma forma que lidamos com números.
- ▶ Todos os cálculos são feitos na matriz diagonal  $\Lambda$ .
- ▶ 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)

## Decomposição em valores singulares (SVD)

- Teorema: qualquer matriz  $A$  pode ser decomposta em,

$$A = UDV^{\top},$$

onde  $D$  é diagonal com entradas não negativas e  $U$  e  $V$  são ortogonais, i.e.  $U^{\top}U = V^{\top}V = I$ .

- Matrizes não quadradas não tem autovalores.
- Os elementos de  $D$  são chamados de valores singulares.
- Os valores singulares são os autovalores de  $A^{\top}A$ .

## Dimensão da SVD

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



# Decomposição em autovalores e autovetores em R

- Função `eigen()` fornece a decomposição    ► Verificando a solução

```
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$vectors ## Autovetores
D <- diag(out$values) ## Autovalores
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
```

```
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
```

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

##           [,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
```

# Regressão ridge

## Regressão ridge

- Relembrando: regressão linear múltipla

$$\begin{array}{c} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\ n \times 1 \end{array} = \begin{array}{c} \begin{bmatrix} 1 & x_{11} & \cdots & x_{p1} \\ 1 & x_{12} & \cdots & x_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \cdots & x_{pn} \end{bmatrix} \\ n \times p \end{array} \begin{array}{c} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \\ p \times 1 \end{array}$$

- Usando uma notação mais compacta,

$$\begin{array}{c} \mathbf{y} \\ n \times 1 \end{array} = \begin{array}{c} \mathbf{X} \\ n \times p \end{array} \begin{array}{c} \boldsymbol{\beta} \\ p \times 1 \end{array}.$$

- Minimiza a perda quadrática:

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

## 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

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

- ▶ Matricialmente, tem-se

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

- ▶ **IMPORTANTE !!**

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

## Regressão ridge

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

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

## Aplicação: regressão ridge

- Resolvendo o sistema linear, tem-se

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

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

## Aplicação: regressão ridge

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

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

- ▶ Usando a decomposição SVD, tem-se

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

- ▶ É possível mostrar que,

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



# Implementação computacional: regressão ridge

## 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.

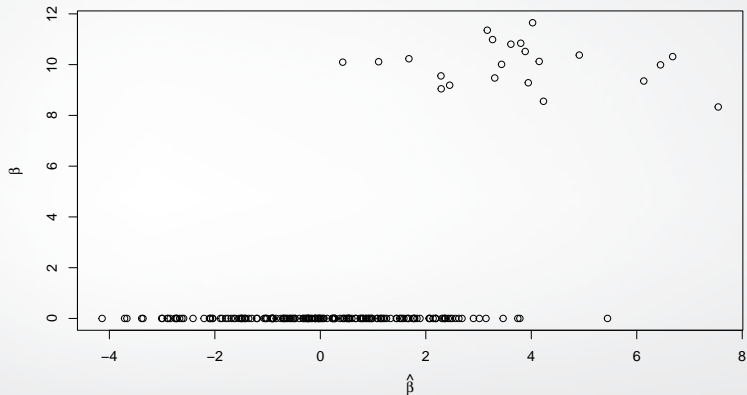
### ► Modelo passo-a-passo

```
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.

```
plot(beta ~ beta_hat, xlab = expression(hat(beta)), ylab = expression(beta))
```



## Resultados: regressão ridge

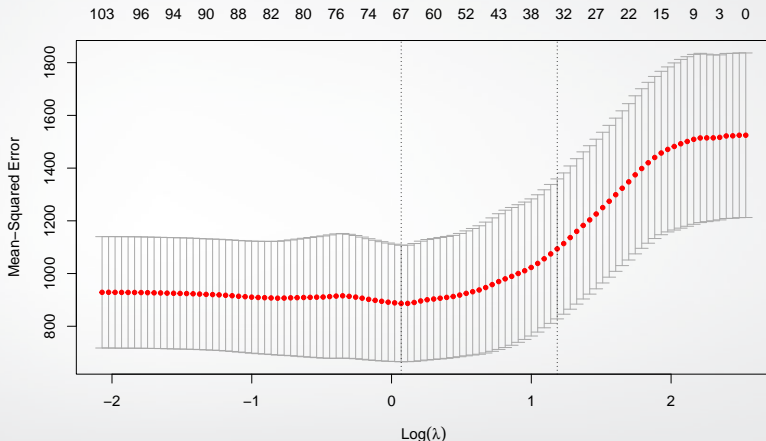
- ▶ Regressão com penalização ridge, bem como, outras penalizações são eficientemente implementadas em R via pacote glmnet.
- ▶ IMPORTANTE! A penalização no glmnet é ligeiramente diferente, por isso os  $\hat{\beta}$ 's não são idênticos a nossa implementação naive.
- ▶ O glmnet oferece opções para selecionar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

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

## Resultados: regressão ridge

### ► Validação cruzada.

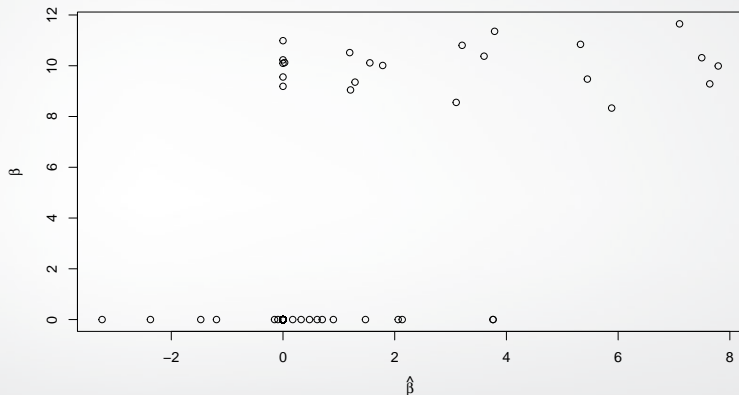
```
plot(beta_glm)
```



## Resultados: regressão ridge

- Ajustados (glmnet) versus verdadeiros.

```
plot(beta ~ as.numeric(coef(beta_glm)), xlab = expression(hat(beta)), ylab = expression(beta))
```



# Comentários



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