Sistemas de Equações Lineares

Prof. Wagner H. Bonat

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

Sumário

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

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

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

$$\mathbf{Ax} = \mathbf{b}.$$

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

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

Operações com linhas

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

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

- 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:

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

- 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:

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

Sumário

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)
S
```

```
#      [,1]          [,2]   [,3]   [,4]
# [1,]    3  2.000000e+00  6.000 24.000
# [2,]    0  2.666667e+00 -1.000  7.000
# [3,]    0 -5.551115e-17 -6.125 -6.125
```

Aplicação: Eliminação de Gauss sem pivotação

```r
# Passo 2: Substituição regressiva
sol = sub_reg(SOL = S)
sol
```

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

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

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

Sumário

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.

Sumário

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

- 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.
- Baseado na decomposição o sistema tem a forma:

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

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

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

- Solução é obtida em dois passos
  1. Resolva Eq.(4) para obter $\boldsymbol{y}$ usando substituição progressiva.
  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
$$\mathbf{PA} = \mathbf{LU}.$$
- $\mathbf{P}$ é uma matriz de permutação.
- Se as matrizes $\mathbf{LU}$ 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.

```r
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
```

Aplicação: Decomposição LU

- Verificando a solução.

```
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)
y
```

```
# [1] 24.000  7.000 -6.125
```

```
# 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
```

Aplicação: Decomposição LU

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

```
require(Matrix)
```

```
# Loading required package: 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
```

Sumário

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.

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

- Três 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

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

```r
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
```

Aplicação: Inversa via decomposição LU

```
# 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.

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

- 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))
}
```

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)
```

Aplicação: Método iterativo de Jacobi e Gauss-Seidel

```r
#Jacobi's method
lsolve.jacobi(A, b)$x
```

```
#            [,1]
# [1,]  5.000025
# [2,] -1.999946
# [3,]  2.499986
# [4,] -1.000045
```

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

```
#            [,1]
# [1,]  5.000019
# [2,] -1.999970
# [3,]  2.499992
# [4,] -1.000014
```

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

Decomposição de matrizes

- Uma infinidade de estratégias para decompor uma matriz em outras mais simples estão disponíveis.
- As decomposições mais usadas em estatística são:
  1. Decomposição em autovalores e autovetores (`eigen()`).
  2. Decomposição QR (`qr()`).
  3. Decomposição de Cholesky (`chol()`).
  4. Entre outras.