Matrizes, vetores e sistemas lineares no R

Ivette Luna

26 de março de 2019

Contents

Vetores

Já sabemos que usamos a função `c()` para criar vetores no R (ver apostila de introdução ao R).

Operações matemáticas com vetores

```r
# Operações básicas (matemática)

v1 <- c(5, 8, 9, 6.25, 7, 7)

v2 <- c(7, 5, 10, 3, 3, 4)

soma_vetores <- v1 + v2

soma_vetores
```

```
## [1] 12.00 13.00 19.00  9.25 10.00 11.00
```

```r
# para ver o tamanho do vetor
length(v1)
```

```
## [1] 6
```

```r
# produto por um escalar - calculando a media
vetor <- soma_vetores*0.5

vetor
```

```
## [1] 6.000 6.500 9.500 4.625 5.000 5.500
```

```r
# produto escalar

prod_escalar = sum( v1*v2 )

prod_escalar
```

```
## [1] 232.75
```

```r
# a norma de um vetor

norma_v1 = sqrt( sum(v1*v1) )

norma_v1
```

```
## [1] 17.5232
```

```r
# tamanho do vetor
n <- length(soma_vetores)
```

Outras funções importantes com vetores

```r
2:10
```

```
## [1]  2  3  4  5  6  7  8  9 10
```

```r
seq(1,10,by=1)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
rep(2, 10)
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2
```

```r
v = c(2, 8, 3, 1, 9)

sort(v)
```

```
## [1] 1 2 3 8 9
```

```r
sort(v, decreasing = TRUE)
```

```
## [1] 9 8 3 2 1
```

Matrizes no R

Uma matriz 2×3 contendo os elementos 1 a 6, nas colunas, é gerada por:

```r
A <- matrix( seq(1:6), nrow=2 )

A
```

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

Note que os valores no vetor de entrada à função `matrix` são colocados na matriz preencheno por colunas. O mesmo resultado seria obtido se se defini-se o número de colunas.

```r
A <- matrix( seq(1:6), ncol=3 )

A
```

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

A dimensão de uma matriz pode ser verificada utilizando as funções `dim`, `nrow` e `ncol`:

```r
dim(A)
```

```
## [1] 2 3
```

```r
nrow(A)
```

```
## [1] 2
```

```r
ncol(A)
```

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

Suponha uma matriz de ordem 3 com os seguintes elementos em v:

```r
set.seed(1)

v = round( 10*rnorm(10) )
v
```

```
##  [1] -6  2 -8 16  3 -8  5  7  6 -3
```

```r
A = matrix( v, ncol=3)
```

```
## Warning in matrix(v, ncol = 3): comprimento dos dados [10] não é um
## submúltiplo ou múltiplo do número de linhas [4]
```

```r
A
```

```
##      [,1] [,2] [,3]
## [1,]   -6    3    6
## [2,]    2   -8   -3
## [3,]   -8    5   -6
## [4,]   16    7    2
```

```r
# para preencher por linhas
A = matrix(v, ncol=3, byrow = TRUE)
```

```
## Warning in matrix(v, ncol = 3, byrow = TRUE): comprimento dos dados [10]
## não é um submúltiplo ou múltiplo do número de linhas [4]
```

```r
A
```

```
##      [,1] [,2] [,3]
## [1,]   -6    2   -8
## [2,]   16    3   -8
## [3,]    5    7    6
## [4,]   -3   -6    2
```

Matrizes especiais

```r
# Matriz nula
matrix(0, nrow=2, ncol=2)
```

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

```r
# Matriz unitaria
matrix(1, nrow=2, ncol=2)
```

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

```r
# Matriz diagonal
diag( c(1,2,3) )
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

```r
# Matriz identidade

diag(1, 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
# Para obter a diagonal principal
diag( A )
```

```
## [1] -6  3  6
```

Operações com matrizes

Neste caso, as restrições matemáticas devem ser respeitadas:

```r
A = matrix( c(1,3,2,2,8,9), ncol=2)
A
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    8
## [3,]    2    9
```

```r
B = matrix( 1:6, ncol=2 )
B
```

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

```r
C = matrix( c(5,8,4,2), ncol=2)
C
```

```
##      [,1] [,2]
## [1,]    5    4
## [2,]    8    2
```

```r
A + B
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    5   13
## [3,]    5   15
```

```r
A - B
```

```
##      [,1] [,2]
## [1,]    0   -2
## [2,]    1    3
## [3,]   -1    3
```

```r
A %*% C # produto matricial
```

```
##      [,1] [,2]
## [1,]   21    8
## [2,]   79   28
## [3,]   82   26
```

```r
2*A # produto por um escalar
```

```
##      [,1] [,2]
## [1,]    2    4
## [2,]    6   16
## [3,]    4   18
```

```r
t(A)  # matriz transposta
```

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

```
## [2,]    2    8    9
```

Acesso aos valores na matriz

Precisamos de dois indexadores (linha e coluna):

```r
A[1,2]
```

```
## [1] 2
```

```r
A[ , 2]
```

```
## [1] 2 8 9
```

```r
A[3, ]
```

```
## [1] 2 9
```

```r
A*B # produto elemento a elemento
```

```
##      [,1] [,2]
## [1,]    1    8
## [2,]    6   40
## [3,]    6   54
```

Sistemas lineares no R

Alguns pacotes interessantes para tratamento de matrizes e sistemas lineares:

- matrixcalc
- Matrix (para matrizes esparsas)
- limSolve
- matlib

Alguns operadores padrão:

http://www.statmethods.net/advstats/matrix.html

Visualização de sistemas simples

Seja o sistema

$$\left\{ \begin{array}{rcrcl} 2x & + & y & = & 5 \\ x & - & 3y & = & 6 \end{array} \right.$$

Matricialmente:

$$A = \left[\begin{array}{cc} 2 & 1 \\ 1 & -1 \end{array} \right] \quad B = \left[\begin{array}{c} 5 \\ 6 \end{array} \right]$$

No R, usaremos o pacote **matlib**, logo:

```r
library(matlib)

A <- matrix( c(2, 1, 1, -1), ncol=2 )
A
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    1   -1
```

```
B <- matrix( c(5, 6), ncol=1 )
B
```

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

Visualização de sistemas de duas variáveis

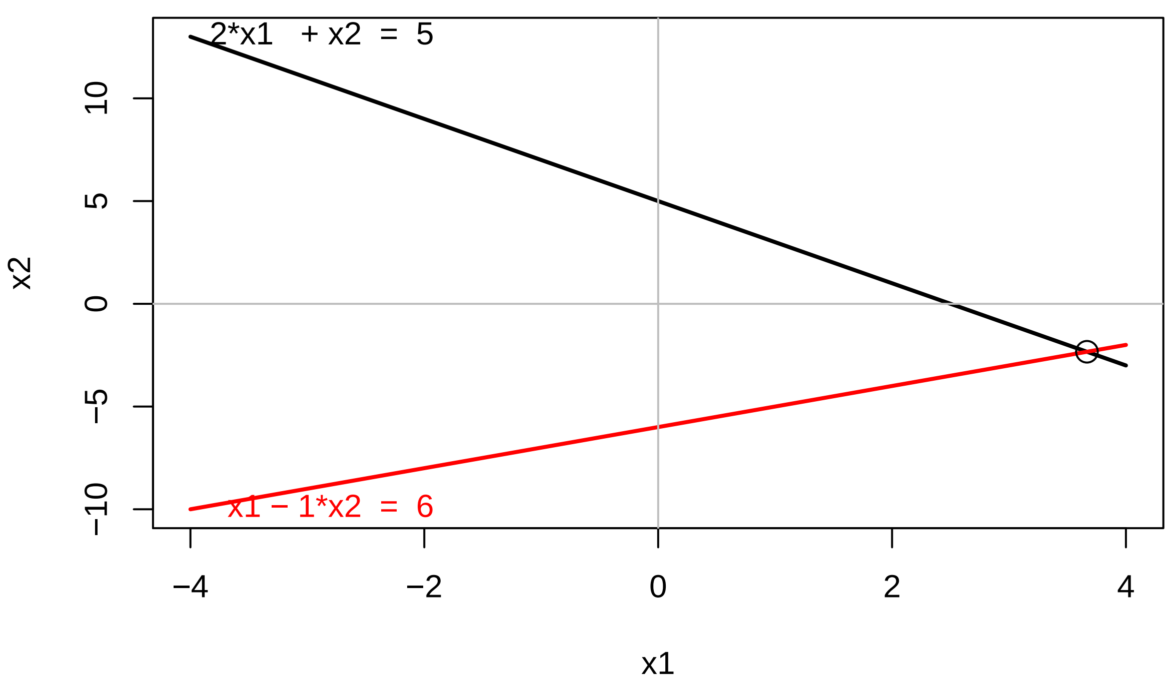
```
# Para visualizar as equações no console e plotar

showEqn(A, B)
```

```
## 2*x1 + 1*x2  =  5
## 1*x1 - 1*x2  =  6
```

```
plotEqn(A, B)
```

```
## 2*x1   + x2  =  5
##   x1 - 1*x2  =  6
```



No caso de três equações de um sistema determinado (ainda no $\mathbb{R}^2$):

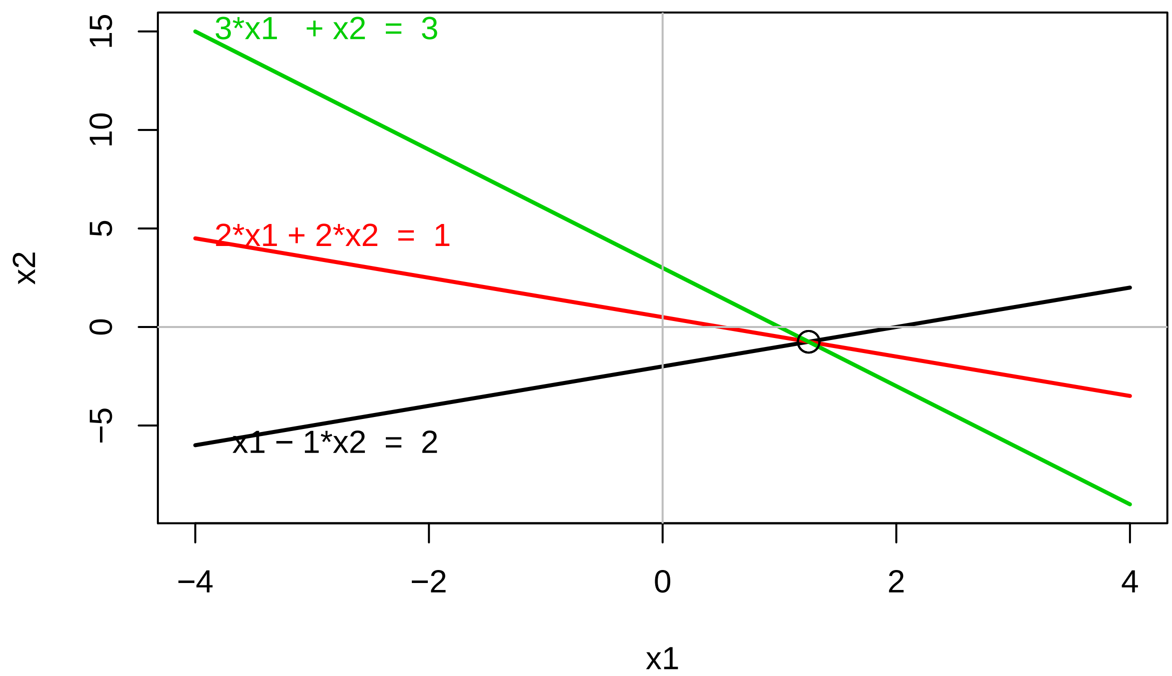
```
A <- matrix(c(1,2,3, -1, 2, 1), 3, 2)
B <- c(2,1,3)

showEqn(A,B)
```

```
## 1*x1 - 1*x2  =  2
## 2*x1 + 2*x2  =  1
## 3*x1 + 1*x2  =  3
```

```r
plotEqn(A, B)
```

```
##   x1 - 1*x2  =  2
## 2*x1 + 2*x2  =  1
## 3*x1   + x2  =  3
```



No $\mathbb{R}^2$, um sistema indeterminado:

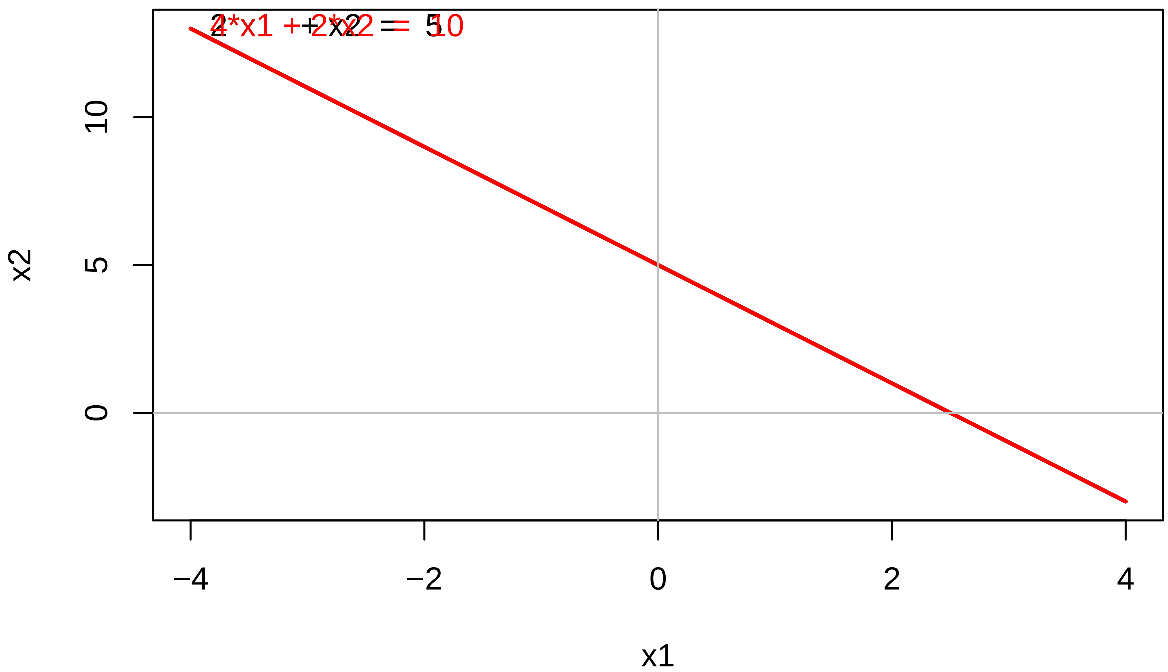
```r
A <- matrix( c(2,4,1,2), ncol=2 )
B <- matrix( c(5,10) )

showEqn(A,B)
```

```
## 2*x1 + 1*x2  =   5
## 4*x1 + 2*x2  =  10
```

```r
plotEqn(A, B)
```

```
## 2*x1   + x2  =   5
## 4*x1 + 2*x2  =  10
```



No $\mathbb{R}^2$, um sistema impossível/inconsistente:

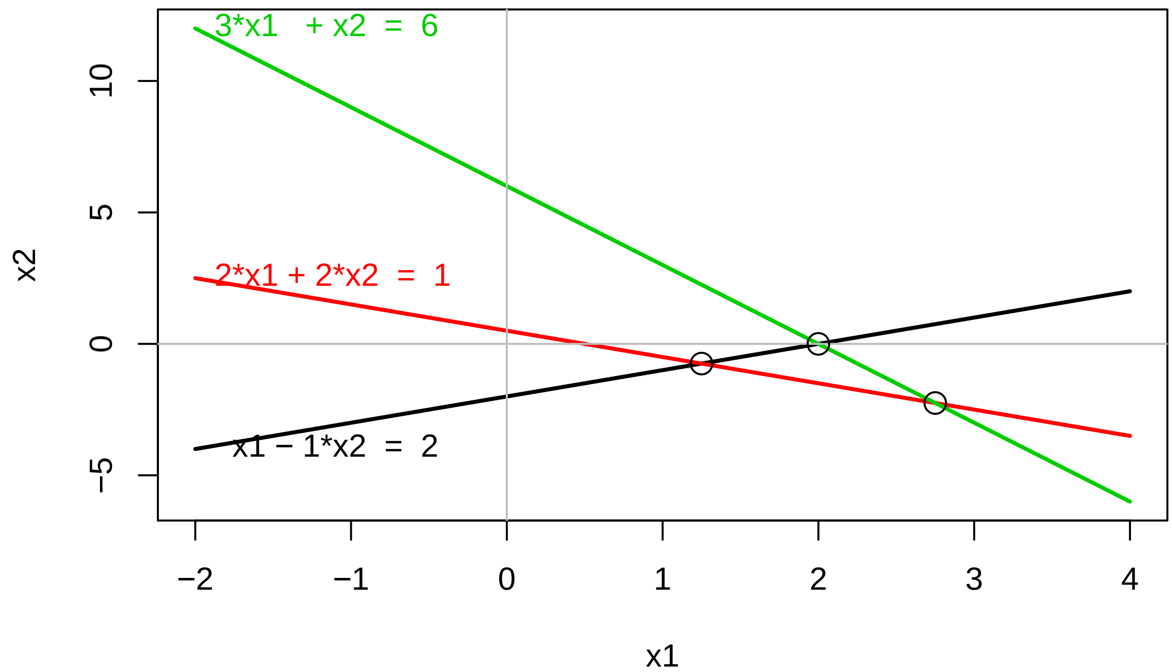
```
A <- matrix(c(1,2,3, -1, 2, 1), 3, 2)
B <- c(2,1,6)

showEqn(A,B)
```

```
## 1*x1 - 1*x2  =  2
## 2*x1 + 2*x2  =  1
## 3*x1 + 1*x2  =  6
```

```
plotEqn(A,B, xlim=c(-2, 4))
```

```
##   x1 - 1*x2  =  2
## 2*x1 + 2*x2  =  1
## 3*x1   + x2  =  6
```



Sistemas com três variáveis

No $\mathbb{R}^3$,

```
# 1. Sistema determinado
A <- matrix(c(6,2,3, 2, 4, 2, 1, 1, 8), 3, 3)
B <- c(7,7,13)

plotEqn3d(A,B, xlim=c(0,4), ylim=c(0,4))


# 2. Um sistema inconsistente
A <- matrix(c(1,  3, 1,
              1, -2, -2,
              2,  1, -1), 3, 3, byrow=TRUE)

# podemos alterar os nomes das linhas e colunas
colnames(A) <- paste0('x', 1:3) # equivale ao paste(...,sep="")
rownames(A) <- paste0('x', 1:3)

A
```

```
##    x1 x2 x3
## x1  1  3  1
## x2  1 -2 -2
## x3  2  1 -1
```

```r
B <- c(2, 3, 6)

showEqn(A, B)
```

```
## 1*x1 + 3*x2 + 1*x3  =  2
## 1*x1 - 2*x2 - 2*x3  =  3
## 2*x1 + 1*x2 - 1*x3  =  6
```

```r
plotEqn3d(A,B, xlim=c(0,4), ylim=c(0,4))
```

Escalonamento de matrizes (Gauss-Jordan)

```r
A <- matrix(c(6,2,3, 2, 4, 2, 1, 1, 8), 3, 3)

A
```

```
##      [,1] [,2] [,3]
## [1,]    6    2    1
## [2,]    2    4    1
## [3,]    3    2    8
```

```r
Aesc <- echelon(A)
Aesc
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
# Para visualizar cada paso do processo de escalonamento
echelon(A, verbose=TRUE, fractions=TRUE) # matriz escalonada e reduzida
```

```
##
## Initial matrix:
##      [,1] [,2] [,3]
## [1,] 6    2    1
## [2,] 2    4    1
## [3,] 3    2    8
##
## row: 1
##
##  multiply row 1 by 1/6
##      [,1] [,2] [,3]
## [1,]   1  1/3  1/6
## [2,]   2    4    1
## [3,]   3    2    8
##
##  multiply row 1 by 2 and subtract from row 2
##      [,1] [,2] [,3]
## [1,]    1  1/3  1/6
## [2,]    0 10/3  2/3
## [3,]    3    2    8
##
##  multiply row 1 by 3 and subtract from row 3
##      [,1] [,2] [,3]
## [1,]    1  1/3  1/6
```

```
## [2,]    0 10/3  2/3
## [3,]    0    1 15/2
##
## row: 2
##
##  multiply row 2 by 3/10
##      [,1] [,2] [,3]
## [1,]    1  1/3  1/6
## [2,]    0    1  1/5
## [3,]    0    1 15/2
##
##  multiply row 2 by 1/3 and subtract from row 1
##      [,1] [,2] [,3]
## [1,]    1    0 1/10
## [2,]    0    1  1/5
## [3,]    0    1 15/2
##
##  subtract row 2 from row 3
##      [,1]  [,2]  [,3]
## [1,]     1     0  1/10
## [2,]     0     1   1/5
## [3,]     0     0 73/10
##
## row: 3
##
##  multiply row 3 by 10/73
##      [,1] [,2] [,3]
## [1,]    1    0 1/10
## [2,]    0    1  1/5
## [3,]    0    0    1
##
##  multiply row 3 by 1/10 and subtract from row 1
##      [,1] [,2] [,3]
## [1,]   1    0    0
## [2,]   0    1  1/5
## [3,]   0    0    1
##
##  multiply row 3 by 1/5 and subtract from row 2
##      [,1] [,2] [,3]
## [1,] 1    0    0
## [2,] 0    1    0
## [3,] 0    0    1
```

Para criar a matriz ampliada precisamos *juntar* a matriz de coeficientes e a matriz de termos independentes:

```r
A <- matrix(c(6,2,3, 2, 4, 2, 1, 1, 8), 3, 3)
B <- c(7,7,13)

# Para adicionar uma linha
C <- rbind(A,B) # row binding

# podemos novamente renomar as colunas e linhas

colnames(C)
```

```
## NULL
```

```r
colnames(C) <- paste0("y",1:ncol(C))

colnames(C)
```

```
## [1] "y1" "y2" "y3"
```

```r
rownames(C) <- paste0("x",1:nrow(C))


# Para adicionar uma coluna
Ahat <- cbind(A,B) # column binding

Ahat
```

```
##              B
## [1,] 6 2 1  7
## [2,] 2 4 1  7
## [3,] 3 2 8 13
```

```r
D <- A[ ,-3] # para eliminar uma coluna
E <- A[-1, ] # para eliminar uma linha
```

Cálculo do posto de matrizes

```r
posto_A <- R(A)
posto_Ahat <- R(Ahat)

# se os postos são iguais o sistema terá solução
c(  posto_A, Ahat )
```

```
##  [1]  3  6  2  3  2  4  2  1  1  8  7  7 13
```

```r
# outra maneira de calcular o posto de matrizes
p_A <- qr(A)$rank

condicao <- all.equal( posto_A, posto_Ahat )

condicao
```

```
## [1] TRUE
```

Se os postos são iguais ao número de variáveis, temos um S.P.D (uma única solução). Como criar esta condição e a sua avaliação?

Resolução numérica de sistemas determinados e indeterminados

Resolução de um sistema determinado

Para o sistema determinado, temos que

$$p_A = p_{\hat{A}} = n$$

onde n é o número de variáveis do sistema.

No R? Precisamos de a estrutura de ccontrole `IF` (ver no final do pdf):

```
A <- matrix(c(6,2,3, 2, 4, 2, 1, 1, 8), 3, 3)

B <- c(7,7,13)

Ahat = cbind(A, B)

posto_A <- R(A)

posto_Ahat <- R(Ahat)


condicao <- all.equal( posto_A, posto_Ahat )
condicao
```

```
## [1] TRUE
```

```
# equivale a condicao <- posto_A==posto_Aamp

n <- ncol(A)

if (condicao & posto_A==n){ # observe que podiamos checar a condicao por fora

        X <- solve(A,B) # inv(A)%*% B
        X

        # usando o escalonamento de matrizes
        Ahat_esc <- echelon(Ahat)
        print( Ahat_esc )

        X_esc <- Ahat_esc[ ,n+1]
        print( round( X_esc, 2 )  )

}
```

```
##                    B
## [1,] 1 0 0 0.5890411
## [2,] 0 1 0 1.1780822
## [3,] 0 0 1 1.1095890
## [1] 0.59 1.18 1.11
```

Resolução de um sistema indeterminado

O comando

$$solve(A,\ B)$$

dará erro caso o sistema seja ideterminado. Podemos opta rpela solução via métodos numéricos.

```
library(limSolve)
```

```
##
## Attaching package: 'limSolve'

## The following object is masked from 'package:matlib':
```

```
##
##     Solve
```

```r
A <- matrix( c(2,4,1,2), ncol=2 )

B <- matrix( c(5,10) )

qr(A)$rank
```

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

```r
qr( cbind(A,B) )$rank
```

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

```r
# os postos sao iguais mas menores que n =2
#solve(A,B) # da erro, logo, nao podemos usar esse comando



# ldei - finds the "least distance" (or parsimonious) solution, i.e. the one where the sum
# of squared unknowns is minima

sol_part <- ldei(E=A, F=B)$X

sol_part
```

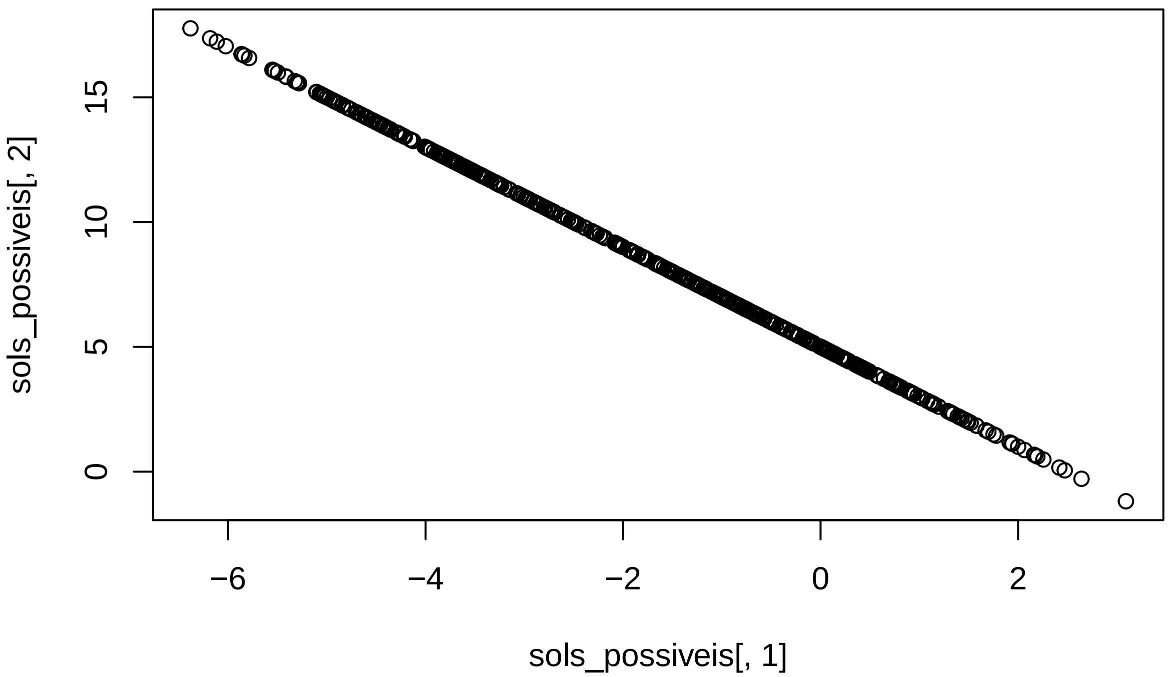
```
## [1] 2 1
```

```r
# It is also possible to randomly sample the solution space. This demonstrates that all valid
# solutions of x2 and x3 are located on a line

# xsample - randomly samples the solution space in a Bayesian way. This method returns
# the conditional probability density function for each unknown

# temos uma solucao numerica, porem, não uma geral
sols_possiveis <- xsample( E=A, F= B, iter=500 )$X

plot( sols_possiveis[ ,1], sols_possiveis[ ,2] )
```




```
# se o sistema e inconsistente, nao ha solucao
```

Estruturas de controle

Control structures in R allow you to control the flow of execution of a series of R expressions. Basically, control structures allow you to put some "logic" into your R code, rather than just always executing the same R code every time. Control structures allow you to respond to inputs or to features of the data and execute different R expressions accordingly (Peng, 2016).

Commonly used control structures are:

- **if and else**: testing a condition and acting on it
- **for**: execute a loop a fixed number of times
- execute a loop **while** a condition is true
  - execute an infinite loop (must break out of it to stop)
  - break the execution of a loop
  - skip an interaction of a loop

Most control structures are not used in interactive sessions, but rather when writing functions or longer expressions.

IF-ELSE

This structure allows you to test a condition and act on it depending on whether it's true or false. Basic structure is given by:

```
if ( <condicao> ) {
```

```
    #comandos a serem repetidos

} # fim do if
```

The above code does nothing if the condition is false. If you have an action you want to execute when the condition is false, then you need an `else` clause.

```
if ( <condicao> ) {

    # executa determinada acao

} else {

    # executa a acao alternativa

}
```

Also, we can nest as much `if` and `else` as we need (nested `if`'s).

Operadores lógicos

http://www.statmethods.net/management/operators.html

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` exactly equal to
- != not equal to
- !x~ Not x
- `x | y` x OR `y`
- `x & y` x AND `y`
- `isTRUE(x)` test if `x` is `TRUE`

FOR

Suponha que queira tirar a raíz cúbica de cinco números

```
x = 2
print( x^(1/3)  )
x=3
print( x^(1/3)  )
x=4
print( x^(1/3)  )
x=5
print( x^(1/3)  )
x=6
print( x^(1/3)  )
```

Mas como faríamos se tivessemos que fazer o mesmo para 2000 números diferentes? Escreveriamos 2000 linhas praticamente iguais? Não seria nada prático! Precisamos de uma estrutura de controle do tipo **FOR**.

O `for` é uma função que repete o código indicado entre `{ }` para o comprimento da sequência indicada entre parênteses. A sua sintaxe:

```
for (indice in sequência) {

```

```
    comandos a serem repetidos

} # fim do for
```

Por exemplo:

```
for (x in 1:5) { # para x=2,3,4,5,6

  print(  x^(1/3)  )

}
```

Ou, usando um índice (`i`)

```
x <- c(1, 2, 3, 4, 5 ) # criamos um vetor com os valores da base

for (i in 1:5) { i = 1, 2, 3, 4, 5

  print( x[ i ]^(1/3)   )

}
```

- Note que o número de linhas da nossa rotina foi reduzido e que, se quisermos agora aplicar essa operação para `x=1,..., 2000` teriamos que mudar apenas um valor nesse *loop*. **Qual?**

O site FULL JOIN oferece uma boa forma de ganhar a intuição sobre loops. O site também mostra um uso mais interessante usando uma base real.