First steps with R

Probabilités et statistique pour la biologie (STAT1)

Jacques van Helden

2019-09-12

Contents

Goal of this tutorial

This tutorial aims at discovering the fundamental elements of the ***R*** statistical language.

We will briefly survey the following concepts.

- Handling variables
  - Assigning a value to a variable
  - Basic operations on numbers
  - Basic data structures

    * vectors
    * matrices
    * data.frames
    * lists

- Using functions
- Graph drawing
- Distributions of probabilities

Using *R* as a calculator

Example: an addition with R.

At the R prompt, type the following instruction and press the Enter key.

```
2 + 5
```

```
[1] 7
```

The result (7) of the addition is printed out, preceded by an index [1] (we will explain later why this index appears).

Assigning a value to a variable

In ***R***, the succession of a the hyphen and “smaller than” characters (`<-`) serves to assign a value to a variable. If the variable does not exist yet, it is created.

For example

```
a <- 2
```

creates a variable named a, and assigns it the value 2.

The result can be displayed with the `print()` function.

```
print(a)
```

```
[1] 2
```

Remark: R also allows to use the equal symbol (`=`) to assign a value to a variable. However, we prefer to use the original assignation (`<-`), to follow the R style recommendations.

Naming and syntactic conventions in R

A priori, are several conventions can be envisaged to ensure a consistent naming of variables, functions, operators, etc.

For each programming language, the community of programmers defines some standard(s) to ensure a consistency of the published code.

For this course, we will follow the recommendations of hte **Google R style guide :**

https://google.github.io/styleguide/Rguide.xml

However, for variable idendifiers, the traditional notation `variable.name` raises some issues for programmers who are familiar with object-oriented languages (e.g. java, python), where the point serves to apply a method (that follows the point) to an object (that precedes the point).

To avoid this confusion, we will use the alternative so-called *camel back* notation (e.g. `variableName`). Attention, according to this convention, variable names always start with a lower case, whereas function / method names start with an uppercase.

Computing with variables

- Create a variable named b with value 5
- Compute $a + b$ and store the result in a variable named c
- Print the result

```
b <- 5
c <- a + b
print(a)
```

```
[1] 2
```

```
print(b)
```

```
[1] 5
```

```
print(c)
```

```
[1] 7
```

Assignation $\neq$ equality

Exercise

- Replace the value of a by 3
- Print out the value of c
- Is-it still true that $c = a + b$? Why?

Solution

```
a <- 3 ## Change the value of a
print(a)
```

```
[1] 3
```

```
print(b)
```

```
[1] 5
```

```
print(c)
```

```
[1] 7
```

```
## Check whether c equals a + b
c == a + b
```

```
[1] FALSE
```

Interpretation: `==` tests whether two variables have the same content. The result is a logical value (TRUE or FALSE).

Recomputing a result

When the content of a given variable a is changed, another variable (c) previously computed from it has no reason to be recomputed if not explicitly requested.

Example:

- Replace the value of a by 27,
- Recompute the value of c
- Test the equality $c = a + b$

```
a <- 27 ## Change the value of a
c <- a + b
print(c) ## Print the value of c
```

```
[1] 32
```

```
## Check whether c  equals a + b
c == a + b
```

```
[1] TRUE
```

Vectors of values

In ***R***, the simplest data structure is a **vector**.

- In the previous example, the variable *a* contained a single number, but in practice it was stored in a single-entry vector.
- The R function `print()` displays the indices at the beginning of each row. This is useful when displaying a vector with alarge number of entries.

Example: create a variable named ***threeNumbers***, and initialise it with a vector containing the values ***27***, ***12*** and ***3000***.

Tips:

- the function `c()` combines several values into a vector.

```
threeNumbers <- c(27,12,3000)
print(threeNumbers)
```

```
[1]   27   12 3000
```

Series of numbers

The simplest way to create a series of number is to use the column character `:`, which generates all integer values between two boundaries;

```
x <- 0:30
print(x)
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[24] 23 24 25 26 27 28 29 30
```

Note: if the printout of the values extends beyond the width of the console, R goes to the next row but displays between square brackets the index of the first element at the beginning of the new row.

Another example

```
print(58:157)
```

```
  [1]  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
 [18]  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91
 [35]  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 [52] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 [69] 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142
 [86] 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
```

Computing with vectors

R enables to handle vectors in a every practical ways: mathematical operations involving a vector automatically apply to all its elements.

```
x <- 1:10 # Define a series from 1 to 10
print(x)
```

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

```
y <- x^2 # Compute the square of each number
print(y)
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

Sequences of numbers

The function `seq()` enables to generate series of bumbers separated by an arbitrary interval.

```
seq(from=-1, to=1, by=0.1)
```

```
 [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3
[15]  0.4  0.5  0.6  0.7  0.8  0.9  1.0
```

Variables can also contain text

Variables are not restricted to numbers: they can contain text (strings of characters).

We will now use the function `c()` to combine severak character strings into a vector.

```
# The # symbol allows to insert comments in R code

# Define  a vector named "whoami", and
# containing two names
whoami <- c("Denis", "Siméon")
print(whoami) # Comment at the end of a line
```

```
[1] "Denis"  "Siméon"
```

Sring concatenation

The function `paste()` enables to concatenate string-containing variables.

```
# To concatenate the elements of a vector in a single chain, use "collapse"
firstName <- paste(collapse = " ", whoami)
print(firstName)
```

```
[1] "Denis Siméon"
```

```
# TO concatenate two vectors, use "sep"
lastName <- "Poisson"
print(paste(sep = " ", firstName, lastName))
```

```
[1] "Denis Siméon Poisson"
```

```
## Concatenate 2 vectors with 3 values each
firstNames <- c("George", "Alfred", "Frédéric")
lastNames <- c("Sand",  "Musset", "Chopin")
fullNames <- paste(sep = " ", firstNames, lastNames)
print(fullNames)
```

```
[1] "George Sand"     "Alfred Musset"   "Frédéric Chopin"
```

Note that the `paste()` functions can also be used to concatenate all the values of a given vector, but this requires to use the `collapse` argument insteead of `sep`.

```
paste(fullNames, collapse = ", ")
```

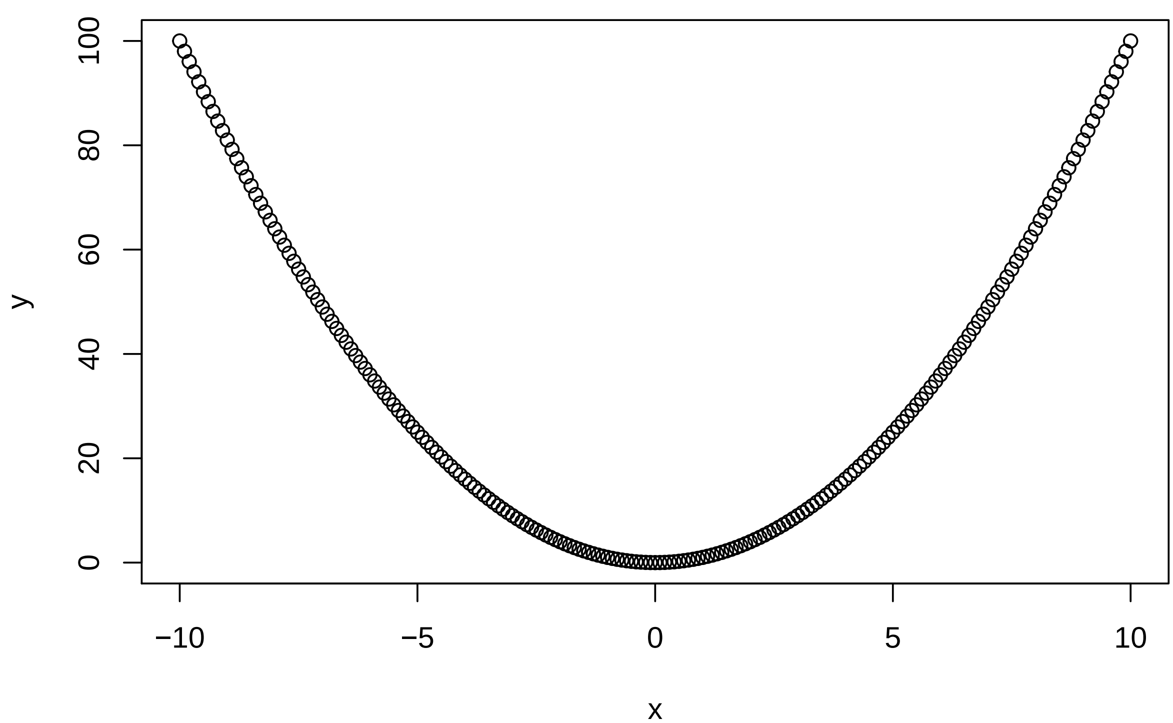
```
[1] "George Sand, Alfred Musset, Frédéric Chopin"
```

Graphical functions

R includes a large number of functions enabling to draw simple or elaborate graphics. We explore hereafter the simplest methods.

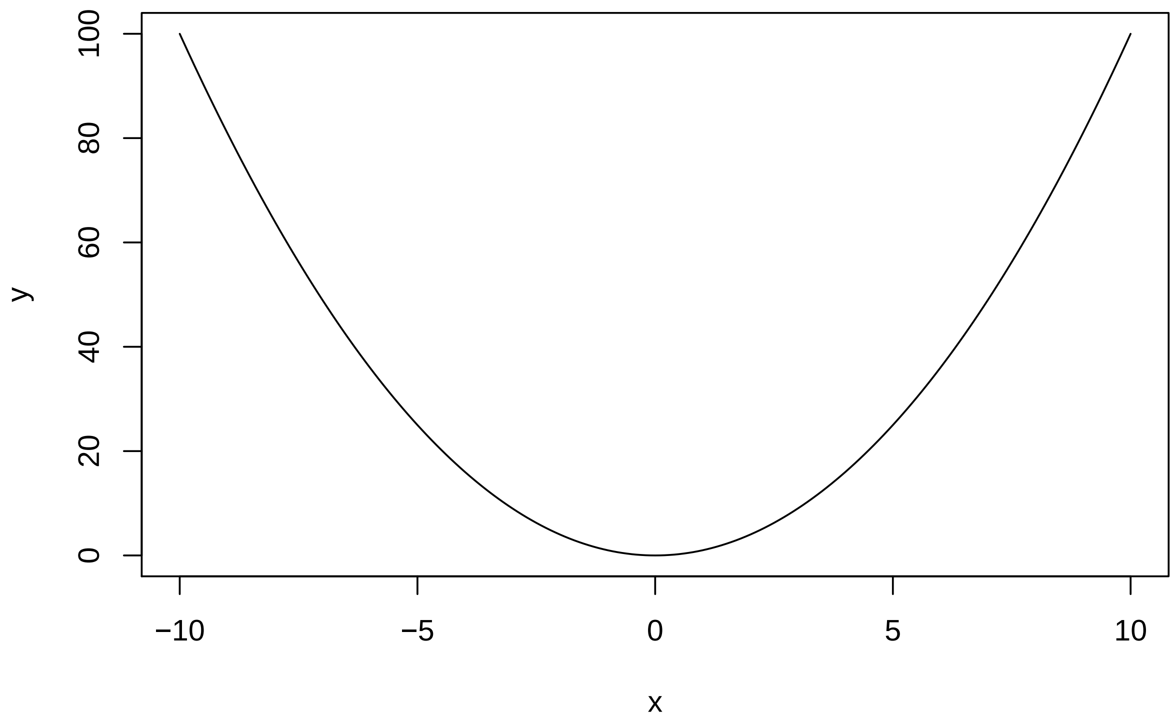
Scatter plot

```
x <- seq(from = -10, to = 10, by = 0.1)
y <- x^2
plot(x,y)
```



Curve (line plot)

```
x <- seq(from = -10, to = 10, by = 0.1)
y <- x^2
plot(x,y, type="l")
```



Improving a graphics

Exercise: on the R console, type `help(plot)`, read the help of the `plot()` function, and explore the parameters in order to improve the previous graphics. Also consult the help, function for the graphical parameters (`help(par)`).

You could for example, attempt to add the following elements to the figure:

- title
- axis labels
- line color
- line width
- grid
- trace an horizontal line to mark the Y axis at coordinate $X = 0$
- trace a vertical line to mark the X axis at coordinate $Y = 0$
- any other parameter that will improve the readability / interpretability of the resulting Figure

An example of solution is shown in the next section (don't look before having done the exercise!).

Solution

Run the following chunk of code to generate the improved figure.

```
x <- seq(from = -10, to = 10, by = 0.1)
y <- x^2
plot(x,y,
```

```
     type="l", # Plot type
     main = "Parabole", # Main title
     xlab = "x", #X label
     ylab = "y = x^2", # Y label
     col = "blue", # Curve color
     lwd = 3, # Line width
     las = 1 # display axis labels horizontally
     )
grid(lty = "dashed", col="gray") # Grid
abline(h = 0) # Horizontal line
abline(v = 0) # Vertical line
```

Probability distributions

For each one of the classical distribution of probabilities, **R** provides 4 functions-.

Before going any further, read carefully the help for the functions associated to the binomial distribution.

```
help("Binomial")
```

Questions:

1. Which R function enables to compuote the probabilty mass function (also called "density" for convenience)?
2. Which R function corresponds to the cumulative distribution function (CDF)?
3. What is the role of the function *rbinom()*?

Drawing the binomial distribution

Exercise: assuming a DNA seqsuence with equiprobable nucleotides, draw the expected distribution for the number of Adenin (A) residues in an oligonucleotide of length 30 (count the adenines on a single strand).

- fonction de masse de la probabilité
- fonction de répartition

Dans les diapos suivantes, nous vous guidons pas à pas pour débuter l'exercice, et vous pourrez ensuite améliorer le résultat à vote guise.

Formule de la solution

Le nombre d'adénines peut prendre n'importe quelle valeur entre 0 et 30. On peut modéliser le problème comme un schéma de Bernoulli avec $n = 30$ essais pouvant chacun résulter en un succès (une adénine) avec une probabilité $p = 0.25$, ou un échec (tout autre nucléotide), avec une probabilité $q = 1 - p = 0.75$.

La probabilité d'observer exactement x adénine vaut donc.

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} = \frac{30!}{x!(30-x)!} \cdot 0.25^x \cdot 0.75^{n-x}$$

où x peut prendre n'importe quelle valeur entre 0 et 30.

Calcul de la fonction de masse de probabilité

```
## Define all possible values for X
n <- 30
x <- 0:n
```

```r
p <- 0.25

## Compute the binomial PMF
pmf <- dbinom(x = x, size = n, prob = p)
```

Sous-ensemble des valeurs d'un vecteur

Nous pouvons imprimer les 4 premières valeurs de la variable *pmf* (pour x de 0 à 3) . . .

```r
pmf[1:5]
```

```
[1] 0.0001785821 0.0017858209 0.0086314677 0.0268534550 0.0604202738
```

. . . ou les 4 dernières valeurs (pour x de 27 à 30).

```r
pmf[(n-3):n]
```

```
[1] 1.925374e-12 9.508019e-14 3.395721e-15 7.806256e-17
```

Restriction du nombre de décimales

La fonction `round()` arrondit un résultat à un nombre donné de décimales.

```r
round(pmf, digit=3)
```

```
 [1] 0.000 0.002 0.009 0.027 0.060 0.105 0.145 0.166 0.159 0.130 0.091
[12] 0.055 0.029 0.013 0.005 0.002 0.001 0.000 0.000 0.000 0.000 0.000
[23] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

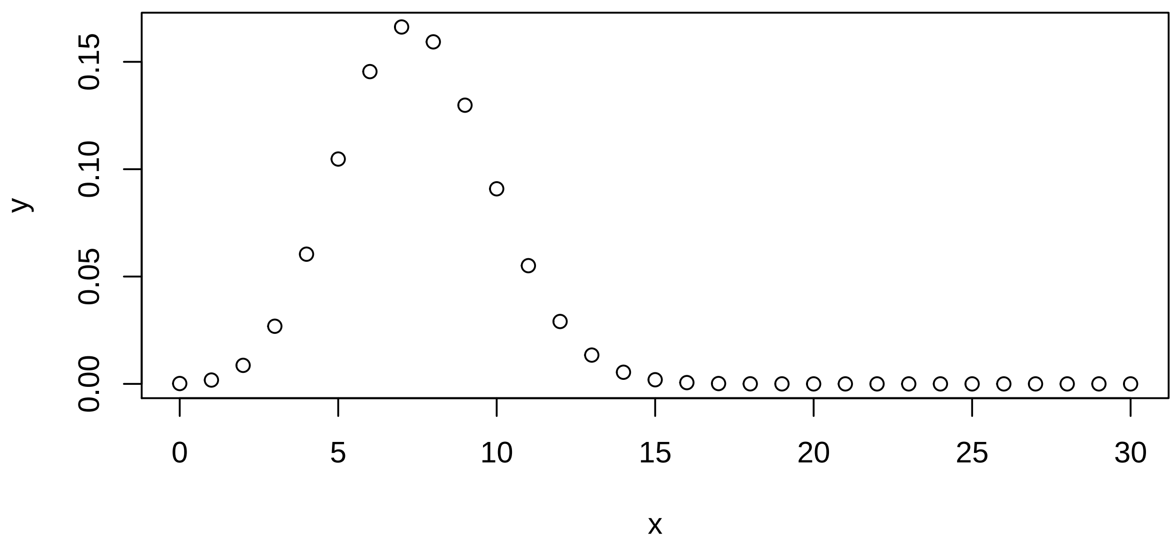
Pour des distributions de probabilités, on descend assez facilement à des valeurs très faibles, dont on désire connaître l'ordre de grandeur tout en affichant un nombre raisonnable de chiffres significatifs (ceux qui suivent la première décimale non nulle). Pour cela, il est plus pratique d'utiliser la fonction `signif()`,

```r
signif(pmf, digit=3)
```

```
 [1] 1.79e-04 1.79e-03 8.63e-03 2.69e-02 6.04e-02 1.05e-01 1.45e-01
 [8] 1.66e-01 1.59e-01 1.30e-01 9.09e-02 5.51e-02 2.91e-02 1.34e-02
[15] 5.43e-03 1.93e-03 6.03e-04 1.66e-04 3.99e-05 8.39e-06 1.54e-06
[22] 2.44e-07 3.33e-08 3.86e-09 3.75e-10 3.00e-11 1.93e-12 9.51e-14
[29] 3.40e-15 7.81e-17 8.67e-19
```

Dessin de la distribution binomiale

```r
n <- 30; x <- 0:n   # Define the X values from 0 to 14
y <- dbinom(x = x, size = n, prob = 0.25) # Poisson density
plot(x,y) # Check the result
```



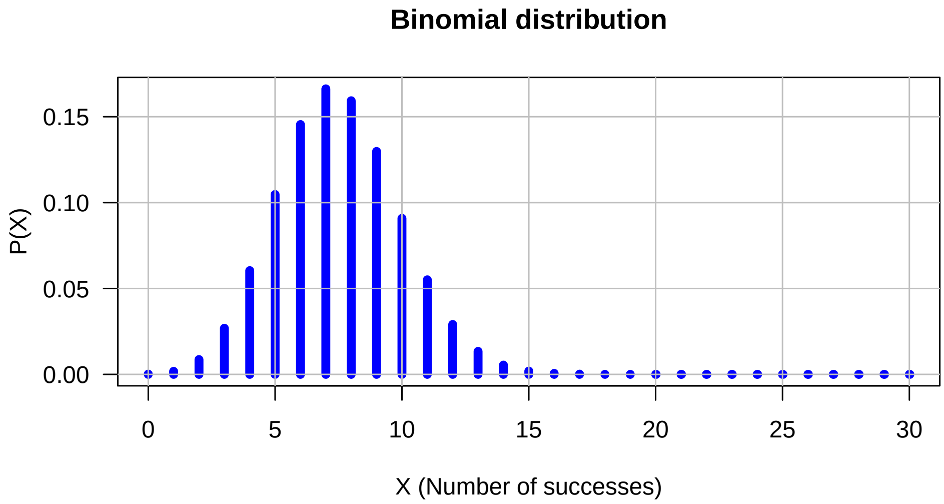
Ce premier dessin n'est pas très élégant. Les points ne se détachent pas très bien du fond. Nous allons améliorer ce dessin.

Exercice 1 : améliorer le dessin de la binomiale

Utilisez les différentes option de la fonction `plot()` pour mieux mettre en évidence la forme de la distribution (résultat attendu sur la diapo suivante).

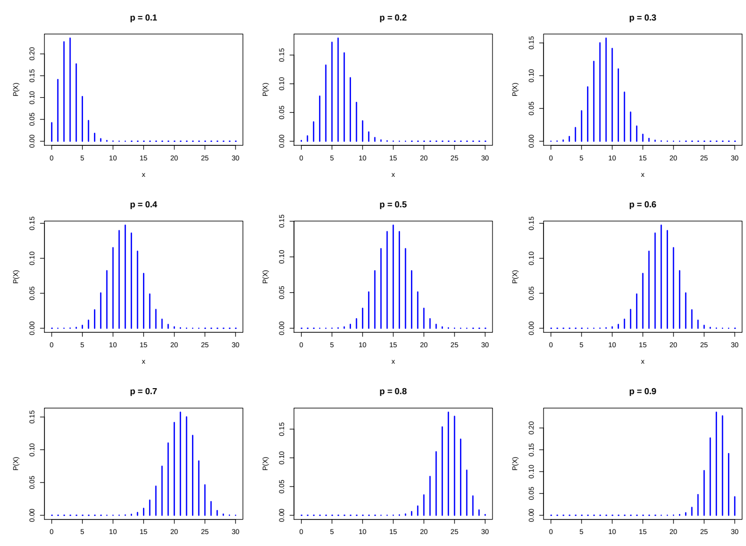
- Commencez par choisir un type de points (option `type`) qui donne l'impression de la hauteur de la probabilité (valeur Y) correspondant à chaque point de l'axe X.
- Ajoutez un titre (option `main`) et adaptez les légendes aux axes (options `xlab`et `ylab`)
- Colorez le dessin (option `col`)
- Epaississez les traits (option `lwd`)
- Ajoutez une grille horizontale (fonction `grid()`)
- Assurez-vous que les étiquettes des axes soient toutes horizontales (option `las`).

Résultat attendu

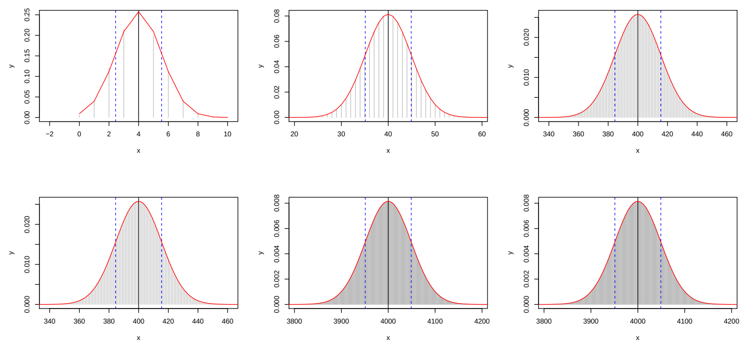


Exercice: série de courbes binomiales

Dessinez une série de courbes binomiales avec $n = 30$ essais, et des valeurs de p allant de 0.1 à 0.9 par pas de 0.1.



Exercice: convergence de la binomiale vers une normale



Avant de terminer : conservez la trace de votre session

La traçabilité constitue un enjeu essentiel en sciences. La fonction ***R*** `sessionInfo()` fournit un résumé des conditions d'une session de travail: version de R, système opérateur, bibliothèques de fonctions utilisées.

```
sessionInfo()
```

```
R version 3.6.1 (2019-07-05)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] knitr_1.23

loaded via a namespace (and not attached):
 [1] compiler_3.6.1  magrittr_1.5    tools_3.6.1     htmltools_0.3.6
 [5] yaml_2.2.0      Rcpp_1.0.2      stringi_1.4.3   rmarkdown_1.14
 [9] stringr_1.4.0   xfun_0.8        digest_0.6.20   evaluate_0.14
```