Premiers pas avec R

Probabilités et statistique pour la biologie (STAT1)

Jacques van Helden

2017-09-13

But de ce tutoriel

Le but de ce tutoriel est de décuvrir les éléments fondamentaux du langage de statistiques ***R***.

Nous aborderons les éléments suivants:

- Manupulation de variables
    - Assignation d'une valeur à une variable
    - Opérations basiques sur des nombres
    - Structures de base: Ecteurs, matrices et data.frames
- Graphiques
- Distributions de probabilités

Utiliser *R* comme calculette

Exemple: une addition avec R.

```
2 + 5
```

```
[1] 7
```

Le résultat affiché (7) est précédé d'un indice ([1]). Nous verrons ultérieurement l'utilisation de ces indices.

Assignation d'une valeur à une variable

En ***R***, **<–** signifie "crée une variable et assigne-lui une valeur".

- créer une variable nommée a,
- lui assigner la valeur 2,
- imprimer le résultat avec la fonction ***print()***.

```
a <- 2
print(a)
```

```
[1] 2
```

Calculer avec des variables

- créer une variable nommée b avec une valeur 5,
- calculer $a + b$ et enregistrer le résultat dans une variable nommée c,
- imprimer le résultat.

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

```
[1] 2
```

```
print(b)
```

```
[1] 5
```

```
print(c)
```

```
[1] 7
```

Assignation $\neq$ égalité

- remplacer la valeur de a par 3,
- imprimer la valeur de c
- est-il toujours vrai que $c = a + b$? Pourquoi ?

```
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
```

Recalculer un résultat

Quand on change le contenu d'une variable a, une autre variable (c) préalablement calculées à partir de cette variable n'a aucune raison d'être recalculée si on ne le demande pas explicitement.

Exemple:

- remplacer la valeur de a par 27,
- recalculer et imprimer la valeur de c
- tester l'égalité $c = a + b$

```
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
```

Vecteurs de valeurs

En ***R***, la structure de données la plus simple est un **vecteur**.

- Dans l'exemple précédent, la variable a ne contenait qu'un seul nombre, mais en pratique elle était stockée dans un vecteur à une seule entrée.
- La fonction `print()` indique les indices en début de lignes, ce qui est utile quand on affiche des vecteurs avec un grand nombre d'entrées.

Exemple: créer une variable nommée ***trois.nombres***, et l'initialiser avec un vecteur dont les valeurs sont ***27***, ***12*** and ***3000***.

Trucs: - en R, les noms de variables peuvent comporter plusieurs mots séparés par des points; - la fonction `c()` combine plusieurs valeurs en un vecteur.

```
trois.nombres <- c(27,12,3000)
print(trois.nombres)
```

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

Séries de nombres

La façon la plus simple de créer une série de nombre est d'utiliser le double point, qui génère toutes les valeurs entières entre deux limites.

```
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
```

Calculer avec des vecteurs

R permet de manipuler les vecteurs de façon extrêmement pratique. Une opération sur un vecteur s'applique automatiquement à tous ses éléments.

```
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
```

Séquences de nombres

La fonction `seq()` permet de générer des séries de nombres avec un espacement arbitraire.

```
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
```

Les variables peuvent contenir du texte

Les variables ne se limitent pas aux nombres: elles peuvent contenir du texte ("chaînes de caractères", "strings" en anglais).

```
# 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"
```

Concaténation de chaînes de caractèrs

La fonction `paste()` permet de concaténer les variables contenant du texte.

```
# Define  a vector named "names", and
# containing two names
prenom <- "Denis Siméon"
nom <- "Poisson"

# Paste the values of a vector of string
print(paste(sep=" ", prenom, nom))
```

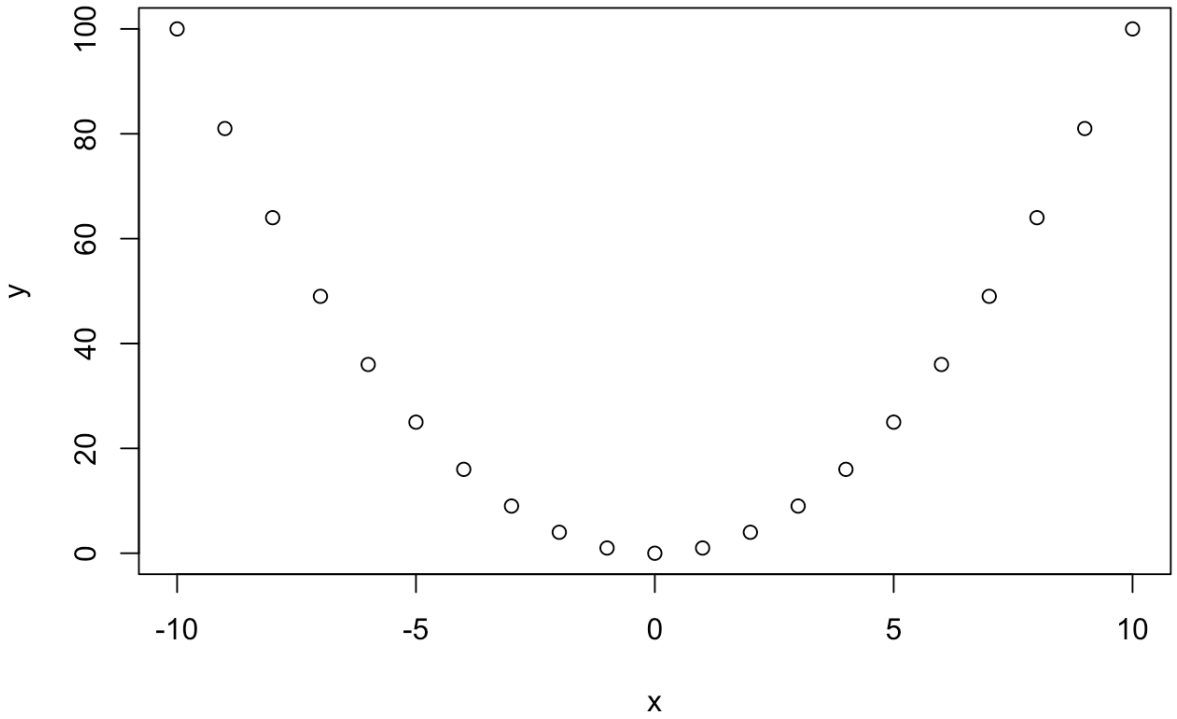
```
[1] "Denis Siméon Poisson"
```

Fonctions graphiques

R comporte un grand nombre de fonctions permettant de dessiner des graphiques simples ou élaborés.
Nous allons explorer ici les méthodes les plus simples.

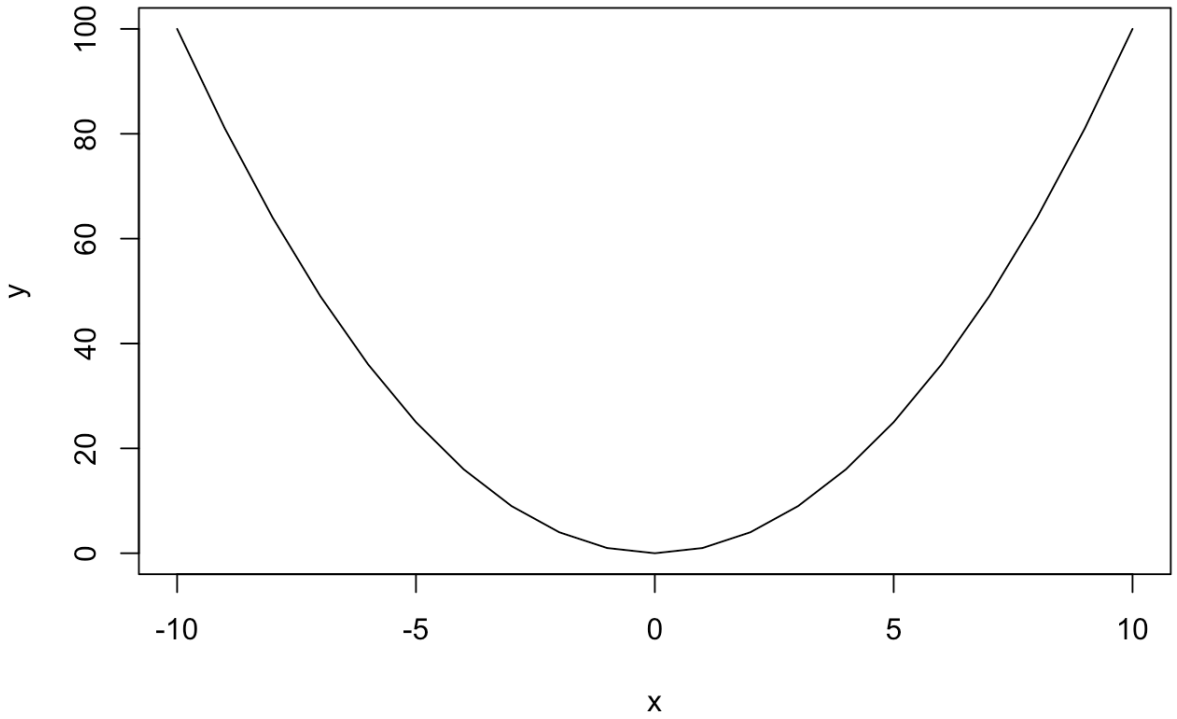
Nuage de points (scatter plot)

```
x <- -10:10
y <- x^2
plot(x,y)
```



Courbe (line plot)

```
x <- -10:10
y <- x^2
plot(x,y, type="l")
```



Améliorer un dessin

Exercice: lisez l'aide de la fonction `plot()` en tapant `help(plot)`, et explorez les paramètres afin d'améliorer le graphique. Consultez également l'aide de la fonction `par()` (paramètres graphiques).

Vous pouvez par exemple ajouter les éléments suivants:

- titre
- étiquettes des axes
- coloration de la courbe
- épaisseur de la courbe
- grille
- tracer l'axe $X =$
- tout autre paramètre qui améliorera la lisibilité du résultat

Distributions de probabilités

R fournit 4 fonctions pour chacune des des distributions de probabilité classiques.

Avant d'aller plus loin, lisez attentivement l'aide pour les fonctions associées à la distribution binomiale.

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

Questions:

1. Quelle est la fonction R qui permet de calculer la fonction de masse de probabilité ?
2. Quelle fonction R correspond à la fonction de répartition ?
3. A quoi sert la fonction ?

Graphe de la binomiale

Exercice: en supposant des nucléotides équiprobables et indépendants, dessiner la probazbilité du nombre d'adénines pour un oligonucléotide de taille 30.

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
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 (pour x de 0 à 3) …

```
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).

```
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.

```
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
```

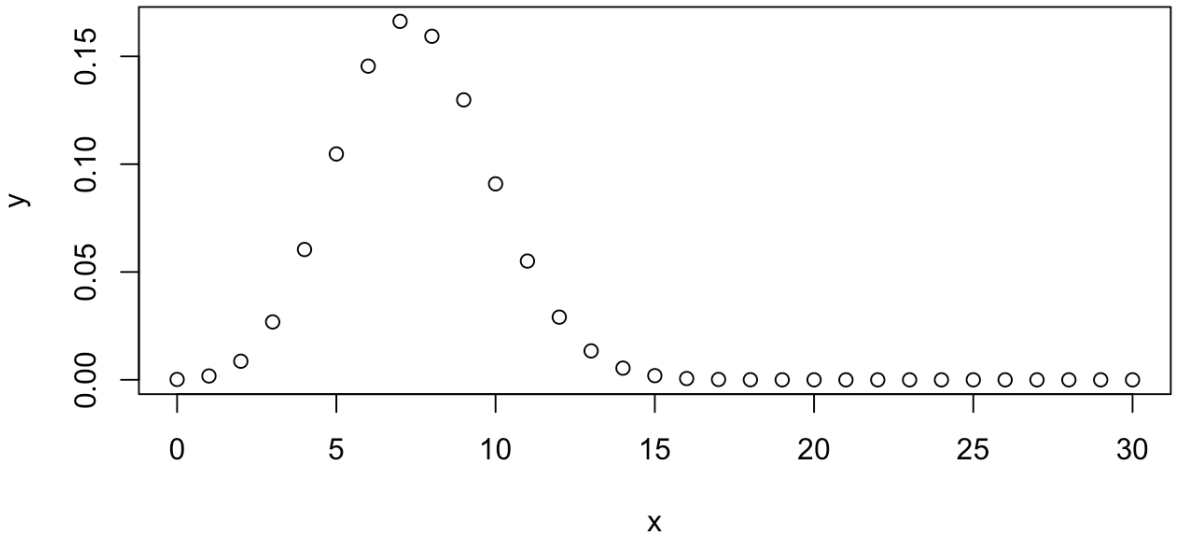
Pour des distributions de probabilités, on descend assez facilement à des valeurs très faibles, dont on désire connaître l'ordre de grandeur tout en affichant un nombre raisonnable de chiffres significatifs (ceux qui suivent la première décimale non nulle). Pour cela, il est plus pratique d'utiliser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

```
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

```
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 convaincant.

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: améliorez le graphique d'une distribution de probabilité

A vous de jouer: utilisez tous les paramètres que vous pouvez pour améliorer la lisibilité du graphique de la distribution binomiale.

Astuce: consultez l'aide de la fonction `par()`, et testez le type "histogram".

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 3.3.2 (2016-10-31)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: macOS Sierra 10.12.6

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_1.15.1

loaded via a namespace (and not attached):
 [1] backports_1.0.5 magrittr_1.5    rprojroot_1.2   tools_3.3.2
 [5] htmltools_0.3.5 yaml_2.1.14     Rcpp_0.12.10    stringi_1.1.5
 [9] rmarkdown_1.5   stringr_1.2.0   digest_0.6.12   evaluate_0.10
```