Formation Package R / Ecriture de Tests

Mathieu Depetris (IRD), Ghislain Durif (IMAG)
Charles-Elie Rabier (IMAG), Isabelle Sanchez (INRAE)

Montpellier Bio-Stat (MBS)

18/05/2021

Première étape : Création d'un package minimaliste

Contexte :

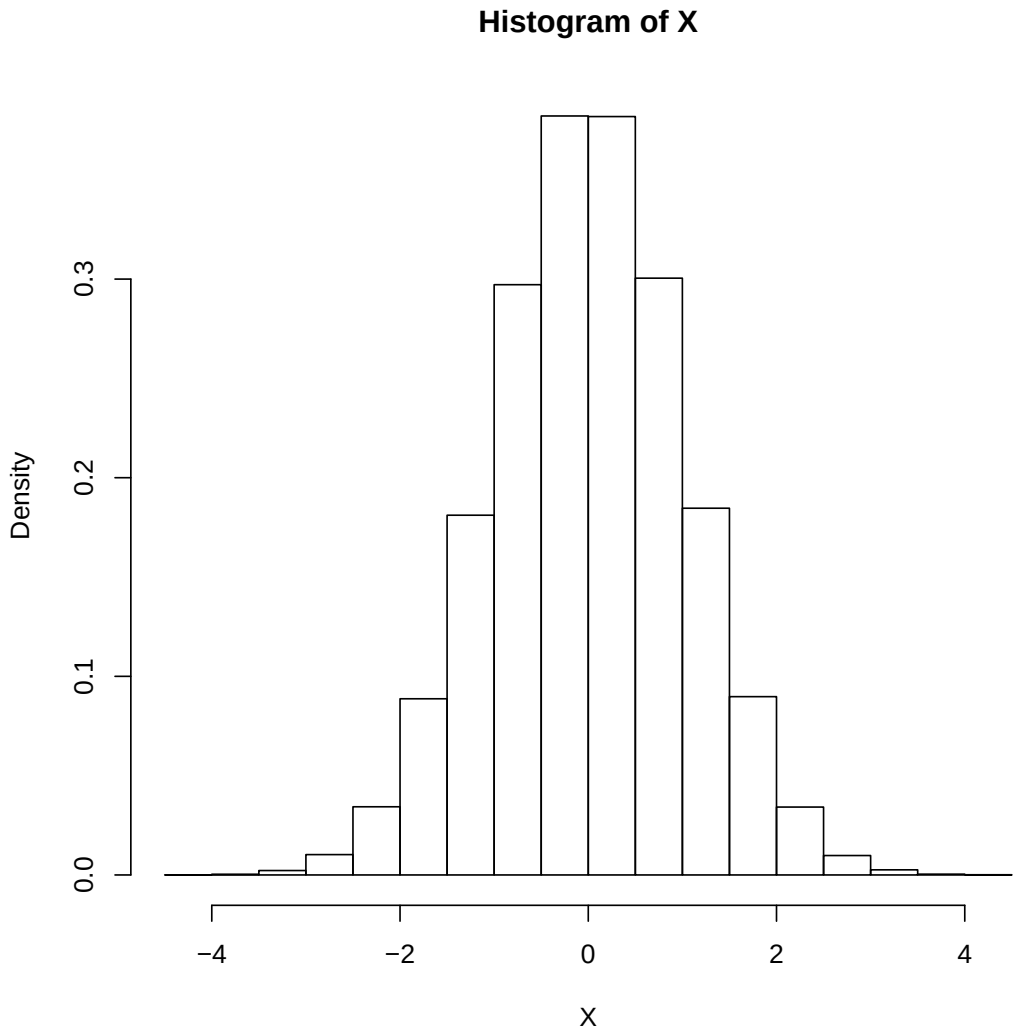
- Nom du package : Randomness
- On cherche à illustrer le générateur aléatoire de R
- On se concentrera sur la loi normale (i.e. loi Gaussienne)

Génération avec R d'observations issues de la loi Normale

- on utilisera la fonction rnorm de R
  - rnorm(n, 0, 1) génére n observations issues de la loi normale de moyenne 0 et de variance 1
  - par défaut, rnorm(n) considère la normale centrée réduite
  - rnorm(n, 0, 1) et rnorm(n) sont donc équivalentes

Premiers pas avec la fonction rnorm()

```
X<-rnorm(100000)
hist(X,freq=FALSE)
```



Première étape : Création d'un package minimaliste

Rappel du tutoriel de K. Broman

Pour obtenir un package minimaliste, il nous faut :

- Créer un dossier PackageRandomnessStage1 contenant
  - un fichier nommé DESCRIPTION avec les 2 lignes suivantes
    - Package : Randomness
    - Version : 0.1
  - un dossier nommé R contenant une nouvelle fonction R nommée sampleRandomNumbers.R

Aide : dossier PackageRandomnessStage1 du git de la formation ...

La fonction R sampleRandomNumbers.R

```
#parametre d entree:
#sampleSize est la taille echantillonnale

#cette fonction genere tout d abord un echantillon
# X(1), ..., X(sampleSize) de taille sampleSize
# issu d'une loi Gaussienne (ie loi normale)
# de moyenne 0 et de variance 1.
# Pour simplifier, on appellera cette echantillon X.
# A partir de l'echantillon  X, elle cree un echantillon Y
# issu d'une loi Gaussienne de moyenne egale a 6
# et de variance 1
# Toujours a partir de l'echantillon  X, elle cree un
# echantillon Z issu d'une loi Gaussienne
# de moyenne 6 et de variance 4

##en sortie, cette fonction renvoie une liste contenant :
## la moyenne empirique de X
## la variance empirique de X
## la variance empirique de Y
## la variance empirique de Z
```

La fonction R sampleRandomNumbers.R

Autres précisions quant à la fonction

- elle renvoie un message d'erreur si sampleSize$<$1
- elle renvoie un warning si 0$<$sampleSize$<$30
- au début de la fonction, on a volontairement fixé la graine du générateur aléatoire à une valeur donnée (set.seed(9)).

La fonction est présente dans le git de cette formation

On souhaite instaurer des tests unitaires afin de vérifier si notre fonction réagit comme prévu

On suit le tutoriel de K.Broman “Writing tests”

- Création d’un dossier nommé tests contenant
  - un fichier testthat.R avec les 2 lignes suivantes
    - library(testthat)
    - test_check(“Randomness”)
  - un dossier testthat contenant
    - un fichier test-sampleRandomNumbers.R

Quelques syntaxes du package testthat

La documentation complète (cf. pdf) de testthat est présente ici :

``` ``https://cran.r-project.org/web/packages/testthat/index.html" ```

Quelques syntaxes utiles pour notre package :

- expect_warning( ) si le code doit renvoyer un warning
- expect_error( ) si le code doit renvoyer une erreur
- expect_equal(a,b) si a et b doivent être égaux, à une tolérance près
- expect_gt(a,b) si a doit etre supérieur à b
- expect_lt(a,b) si a doit etre inférieur à b
```

Description de quelques tests possibles ...

Tests implementés au sein de test-sampleRandomNumbers.R

- la taille échantillonnale est-elle suffisante ?
- l'intervalle de confiance pour la vraie valeur de la moyenne de l'échantillon X (i.e. 0) semble-t-il cohérent ?
- les opérations sur la variance semblent-elles cohérentes ?
- Arrive-t-on bien a réobtenir les mêmes résultats en fixant la graine du générateur aléatoire à la même valeur ?

La taille échantillonnale est-elle suffisante ?

```
context("sampleRandomNumbers")

test_that("la taille echantillonnale est suffisante",  {

expect_warning(sampleRandomNumbers(29))

expect_error(sampleRandomNumbers(-1))

})
```

Intervalle de confiance

```
test_that("l intervalle de confiance pour la vraie valeur
de la moyenne semble coherent",  {
# on s interesse ici a la variable aleatoire X,
# et en particulier a sa moyenne qui est
# egale par definition a 0

# on considere par exemple une taille echantillonnale de 1000

mySampleSize<-1000
U<-sampleRandomNumbers(mySampleSize)
EmpiricalMean<-U[[1]]

# on calcule les bornes d un intervalle de confiance
# a 95 pourcent
# on suppose la variance connue
# 1.96 est le quantile d ordre 0.975 de la
# loi normale centree reduite
lowerBound<-EmpiricalMean-1.96/sqrt(mySampleSize)
upperBound<-EmpiricalMean+1.96/sqrt(mySampleSize)
```

Intervalle de confiance (suite)

```
# la vraie valeur de la moyenne (i.e. 0)
# se doit d etre contenue
# (dans 95 pourcent des cas) dans l intervalle de confiance
# si le generateur aleatoire de R et
# notre fonction marchent correctement
# donc on s attend a ce que cela soit verifie
# sur notre jeux de donnees simulees

expect_gt(0,lowerBound)
expect_lt(0,upperBound)

#en d autres termes 0 doit appartenir
#a l intervalle [lowerBound,upperBound]
```

Opérations sur la variance

```
test_that("les operations sur la variance semblent coherentes"
  {
#pour rappel, les 2 variables aleatoires X et Y
#different uniquement d une translation, car Y=X+6

#par definition Var(X)=Var(X+6)=Var(Y)
# on verifie que la variance est bien la meme pour les
# 2 echantillons simules,
## ie Var(X)=Var(Y)

mySampleSize<-1000
U<-sampleRandomNumbers(mySampleSize)
VarX<-U[[2]]
VarY<-U[[3]]
expect_equal(VarX,VarY)

# pour rappel Z=2X+6, on s attend donc a ce
# que var(Z) soit egale a 4*Var(X)
VarZ<-U[[4]]
expect_equal(VarZ,4*VarX)
})
```

En fixant la graine du générateur aléatoire

```
test_that("j arrive bien a reobtenir les memes
resultats en fixant la graine a la meme valeur",  {

#attention a bien fixer la graine a la meme valeure
#que dans la fonction sampleRandomNumbers
set.seed(9)

mySampleSize<-1000
X<-rnorm(mySampleSize)
U<-sampleRandomNumbers(mySampleSize)
EmpiricalMean<-U[[1]]
EmpiricalVariance<-U[[2]]

# en toute logique la moyenne empirique et la variance
# doivent etre les memes que celles
# obtenues avec la fonction sampleRandomNumbers
expect_equal(mean(X),EmpiricalMean)
expect_equal(var(X),EmpiricalVariance)

})
```

Lancement dans R des 8 tests concoctés

```
# Mon espace de travail
setwd("/Users/MyName/KIMRPackage/PackageRandomnessStage1")

#je charge le package devtools
library(devtools)

#je lance les tests
test()

Loading Randomness
Testing Randomness
Results

FAIL 0 WARN 0 SKIP 0 PASS 8
```

Deuxième étape : Création d'un package plus propre

Sur le git, récupérer le dossier PackageRandomnessStage2.

- Modifier le fichier Description (e.g. avec votre nom)
- Modifier l'entête du fichier sampleRandomNumbers.R afin de personnaliser la documentation du package

Je génère le nouveaux fichier .Rd par Roxygen2

```
setwd("/Users/MyName/KIMRPackage/PackageRandomnessStage2")
library(devtools)
document()
```

Deuxième étape : Création d'un package plus propre

- On "contrôle" l'ensemble du package (i.e. pas uniquement nos tests unitaires) après l'avoir fabriqué :

```
R CMD BUILD PackageRandomnessStage2
R CMD CHECK Randomness_0.2.tar.gz
```

- Dans le dossier Randomness.Rcheck, se trouve le manuel du package : Randomness-manual.pdf

Afin d'améliorer notre package,

- Proposer de nouvelles fonctions R (certaines peuvent être invisibles de l'utilisateur)
- Inclure des jeux de données