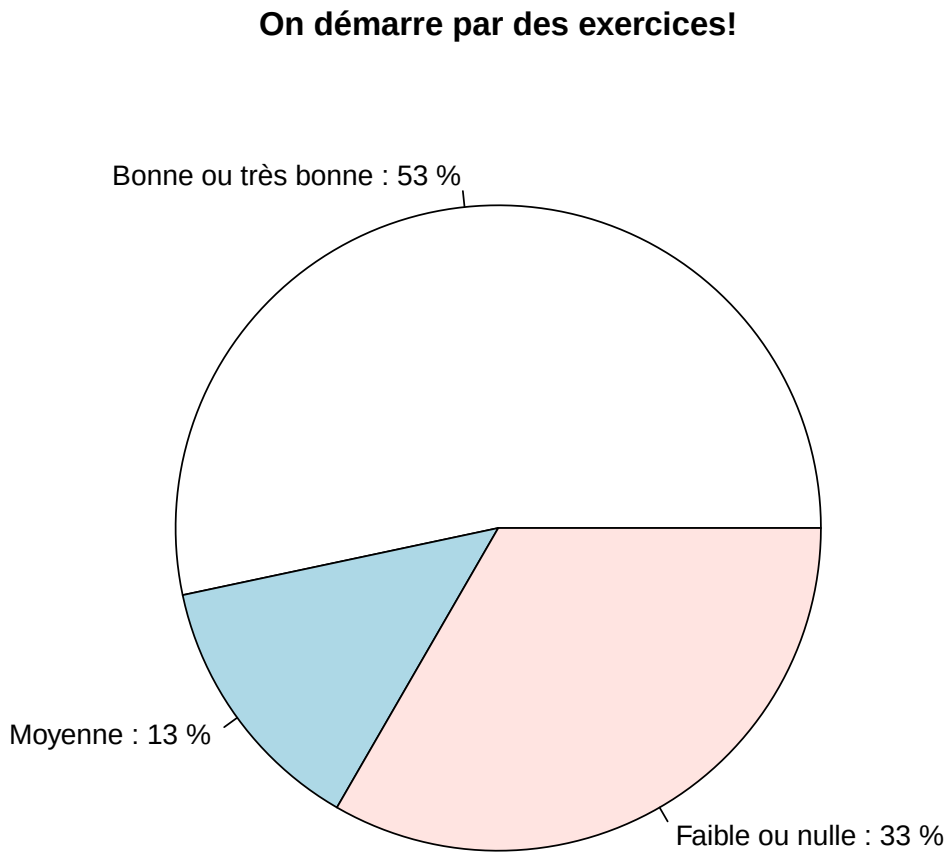
Introduction à R et aperçu des méthodes de simulations

Gilles Faÿ

2023-09-25

1. Prise en main du logiciel R

Votre connaissance de R (autoévaluation)



Installation

- Se rendre sur www.r-project.org et choisir l'installation correspondant à votre OS (Windows, MacOS, ou Linux)
- L'interface de R est assez rudimentaire. Les commandes se saisissent séquentiellement dans la console, à la suite du caractère d'invitation >.
- Vous pouvez éventuellement installer des environnements de développement (IDE) plus conviviaux que l'environnement R de base, par exemple
  - R-studio
  - VScode.
  - Jupyter Notebooks/Lab (NB: Jupyter = JUlia + PYThon + R)

Références

Il existe beaucoup de documentation sur ce logiciel répandu. En complément de cette très brève introduction, nous conseillons

- cette introduction à R : alea.fr.eu.org/git/doc_intro_r.git/blob_plain/HEAD:/intro.pdf
- le manuel officiel R-intro.pdf.
- Pour de plus amples détails, voir cran.r-project.org/manuals.html

Python is better for...	R is better for...
Handling massive amounts of data	Creating graphics and data visualizations
Building deep learning models	Building statistical models
Performing non-statistical tasks, like web scraping, saving to databases, and running workflows	Its robust ecosystem of statistical packages

Figure 1: Forces et faiblesses de R et python, d'après Coursera

Environnement pour ces travaux dirigés

Une version conteneurisée de rstudio a été construire et déployée pour chaque étudiant · e., grâce à la technologie MyDocker.

1. Dans un navigateur, aller sur l'espace Edunao du cours
2. Choisir l'activité MyDocker, cliquer sur "Ouvrir dans une nouvelle fenêtre"
3. Cliquer sur "Demander un environnement"
4. Copier le mot de passe dans le presse-papier et cliquer sur "Connexion à l'interface"
5. Sur la fenêtre de connexion, utiliser rstudio comme Username et le mot de passe copié précédemment

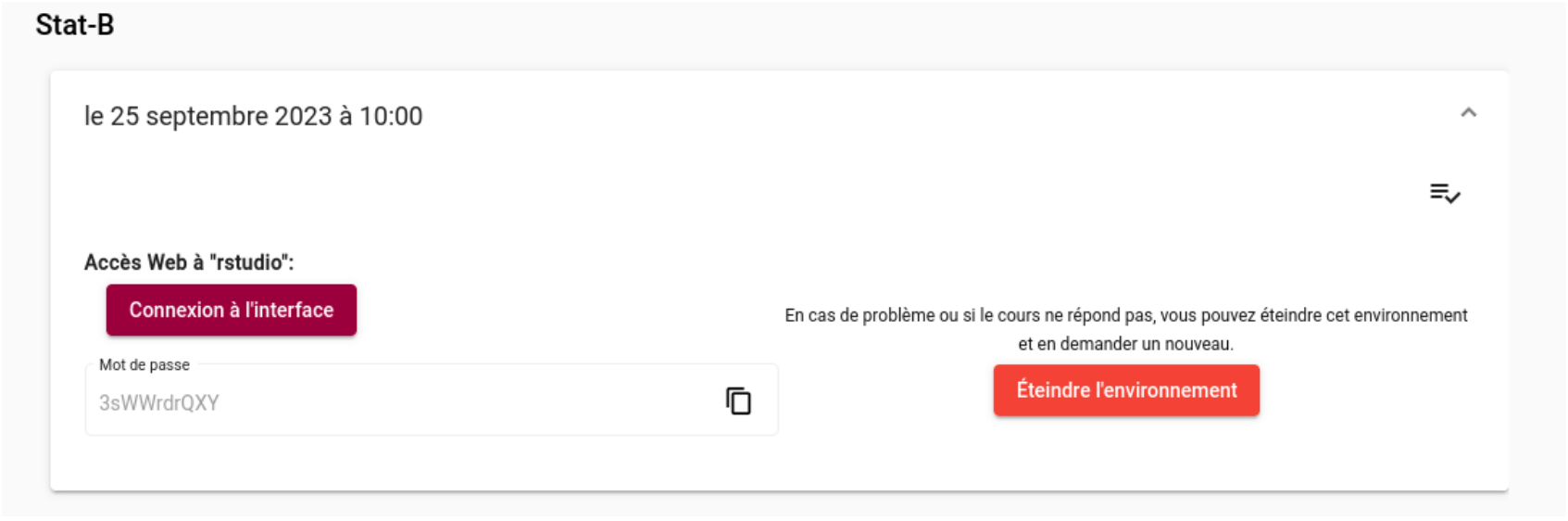


Figure 2: Ecran MyDocker avant l'accès à Rstudio

RStudio/Posit

RStudio est un IDE pour R, commercialisé par Posit, mais dont une version individuelle très complète est accessible librement www.

L'interface RStudio est composée de 4 zones :

1. Fenêtre de commande ou console (à gauche) : console de saisie du codes R, onglet Terminal (shell dans l'OS sous-jacent), onglets de sorties de tâches de compilation etc.
2. Fenêtre espace de travail (en haut à droite) : contient les objets en mémoire, que l'on peut consulter en cliquant sur leur nom, l'historique des commandes exécutées les connexions aux sources de données (spark, odbc).
3. Fenêtre explorateur (en bas à droite) : permet de se déplacer dans les répertoires, de visualiser les graphiques, montre les librairies installées et permet d'en installer de nouvelles, permet d'accéder à la documentation.
4. Fenêtre édition (en haut à gauche) : cette fenêtre n'apparaît que s'il existe des fichiers contenant les scripts R ouverts. Il est recommander de taper son code dans cette fenêtre et de l'exécuter ensuite dans la console afin de pouvoir sauvegarder le code.

R Markdown

- Pour la rédaction d'un projet par exemple, l'utilisation de R Markdown dans RStudio est pratique. Elle a des points communs avec les notebook jupyter ainsi que des différences importantes .
- Caractéristique commune majeure: possibilité d'entremêler des commentaires en texte enrichi (format markdown ou LaTeX) et des lignes de commandes. La compilation d'un fichier Rmarkdown consiste à la fois à la production d'un document au format choisi (pdf, html, doc), voire d'une application interactive (rshiny) et l'exécution d'un code en R dont les sorties (numériques, graphiques) viendront alimenter dynamiquement le rapport.
- Elément-clé pour la **reproductibilité** des analyses.

1. Dans RStudio cliquez sur File $\rightarrow$ New File $\rightarrow$ R Markdown.
2. Choissisez pdf comme Default Output Format. Les format HTML ou rshiny (pour les applications interactives) ou encore beamer pour les présentations (comme celle-ci même) sont aussi disponibles.
3. Un nouveau fichier Markdown s'ouvre avec un exemple qui permet de comprendre la syntaxe.
4. Pour produire le pdf, il faut cliquer sur Knit PDF.

Les librairies

R s'appuie sur de nombreuses librairies: les librairies de base sont chargés en mémoire au moment du lancement, certains seront à charger avant utilisation par la commande library() par exemple

```
library(dplyr) # NB: nom du package sans guillemets
```

D'autres peuvent ne pas être encore installées; leur installation pourra se faire grâce à l'onglet Package/Install ou la ligne de commande

```
install.packages("ggplot2") # NB: nom du package entre guillemets
```

Trouver de l'aide

La documentation et l'aide à l'utilisation d'une fonction ou d'une librairie sont accessibles en cliquant sur l'onglet help ou par ligne de commandes:

```
help("mean")
help(lubridate)
```

```
help.search("kolmo") # Mais où le test de Kolmogorov-Smirnov se cache-t-il?
```

- Pour apprendre, imiter! pour chaque commande, on aura intérêt à utiliser la commande `example` qui exécute le paragraphe d'exemple dont le code se trouve à la fin du fichier d'aide de la commande en question.

```
example(plot)
```

NB: chercher dans l'aide locale avant de demander à Google, StackOverflow ou ChatGPT, tant que c'est plus efficace.

Assigner et afficher

Pour assigner une valeur à une variable :

```
x<-20
x=20 #autre commande
```

Vous pouvez ensuite afficher la valeur d'une variable :

```
x
print(x) #autre commande
```

Vous pouvez afficher l'ensemble des variables du répertoire courant contenant des informations :

```
objects()
ls() # autre commande
```

Les informations précédentes sont disponibles dans la fenêtre **Environment**. Pour libérer de la mémoire, vous pouvez effacer des variables:

```
rm(x) # suppression de l'objet x
rm(list=objects()) # suppression de tous les objets en mémoire (utile!)
```

Interaction avec le système de fichiers

- Que l'on travaille en local ou sur le container, il existe un système de fichier pour héberger le code et ses artefacts ainsi que les données.
- Pour connaître ou modifier le répertoire de travail (working directory) :

```
getwd()  # retourne le répertoire courant
```

```
## [1] "/home/fay/Enseignement/StatB/tdtp"
```

```
  setwd("/home/myself/cours3A/ma1300/tpR/") # modifie le repértoire courant
```

- Il devient vite indispensable d'écrire ses morceaux de code dans un fichier (par exemple d'extension `.R`) au moyen d'un éditeur de texte de votre choix ou celui de votre environnement de développement et de les exécuter via votre interface d'IDE ou la commande

```
  source("monpremierprogramme.R")
```

- Utiliser au choix le Terminal (en bas a gauche), l'explorateur de fichier (en bas a droite), ou les commandes R en console (tels *list.dirs()*) pour se repérer.

Fichier de script

Toutes les commandes peuvent être lancées à partir d'un fichier.

1. Créer un fichier de commandes *monTP.R* dans un répertoire *TPx* (où x est le numéro de la séance).
2. Y copier les quelques commandes précédemment exécutées depuis l'onglet **History** de la fenêtre **Environment** (icône **To Source**).
3. Enregistrer le fichier.
4. Pour exécuter le code depuis la fenêtre d'édition, sélectionner les commandes à exécuter, puis cliquer sur l'icône run, ou utiliser le raccourci clavier CTL+enter (command+enter sur Mac). Note: ce raccourci clavier permet également d'exécuter la ligne sur laquelle se trouve le curseur, sans avoir à la sélectionner en entier.
5. Il est possible d'exécuter la totalité fichier (même non ouvert) avec la commande **source('monTP')**.
6. Les fichiers .R sous Edunao contiennnent les commandes présentes sur ces slides et des eléments de réponses pour les exercices (à utiliser en après avoir cherché la résolution) Vous pouvez le télécharger dans votre répertoire de TP, puis l'éditer dans R pour vous faire gagner du temps.

2. Bases de R

Quelques objets

Les objets de R se différencient par

- leur mode
  - **logical**, **numeric**, **character**
- leur classe (structure)
  - **vector**, **matrix**, **array**, **factor**, **data.frame**, **list**, **function**, ... et toute classe programmée)

NB: *pandas* de python est une librairie s'inspirant des data.frame natifs dans R.

```
mode(2)  # mode ou type plus général
```

```
## [1] "numeric"
```

```
typeof(2) # mode de stockage interne un peu plus fin
```

```
## [1] "double"
```

```
mode("Le dernier jour du disco")
```

```
## [1] "character"
```

```
mode("Juliette"=="Michel")
```

```
## [1] "logical"
```

```
typeof("Juliette"=="Michel")
```

```
## [1] "logical"
```

Création et indexation de vecteurs

On peut créer un vecteur par plusieurs opérations

```
x = c(1,4,9,16,25,36) # c() est l'opérateur de concaténation
y = c(NA)
z = c(T,F,F) # vecteur logique
```

```
1:10 # suite d'entiers entre limites comprises
```

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

```
seq(2,100,by=7) # suite arithmétique
```

```
##  [1]   2   9  16  23  30  37  44  51  58  65  72  79  86  93 100
```

```
rep(c(T,F),5)
```

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

```
x[c(1,3,5)] # Indices positifs
x[-3] # Indice négatif, exclut le 3ème élément de x
x[c(T,T,F,F,T,T)] # Indice valeurs logiques
x[x > 5] # extraction conditionnelle
```

Création de matrices

```r
n=4 # nombre de lignes
p=3 # nombre de colonnes
mymat = matrix(runif(12),nrow=n,ncol=p) #runif : nombre aléatoire sur [0,1]
print(mymat)
```

```
##           [,1]      [,2]       [,3]
## [1,] 0.7458866 0.0227076 0.05691019
## [2,] 0.1663810 0.9012152 0.63207013
## [3,] 0.8885313 0.5083908 0.93691426
## [4,] 0.6350475 0.7230272 0.42403733
```

```r
cbind(mymat,c(1,2,3,4)) # acolle des matrices selon les colonnes (Columns)
```

```
##           [,1]      [,2]       [,3] [,4]
## [1,] 0.7458866 0.0227076 0.05691019    1
## [2,] 0.1663810 0.9012152 0.63207013    2
## [3,] 0.8885313 0.5083908 0.93691426    3
## [4,] 0.6350475 0.7230272 0.42403733    4
```

rbind similairement empile des matrices (ayant le même nombre de colonnes) selon des lignes (Rows)

Indexation de matrices

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

```
## [1] 0.05691019 0.63207013
```

```
mymat[,1] # toutes les lignes de la colonne 1
```

```
## [1] 0.7458866 0.1663810 0.8885313 0.6350475
```

```
mymat[2,] # toutes les colonnes de la ligne 2
```

```
## [1] 0.1663810 0.9012152 0.6320701
```

```
mymat[1:2,2:3] # tranches d'indices -> sous-matrice
```

```
##            [,1]       [,2]
## [1,] 0.0227076 0.05691019
## [2,] 0.9012152 0.63207013
```

```
mymat[8] # un élement avec un sens de lecture conventionnel
```

```
## [1] 0.7230272
```

Les tables de données (*data.frames*)

Un data.frame possède des lignes et des colonnes comme une matrice, mais :

- lignes = individus statistiques
- colonnes = variables statistiques, de type non nécessairement homogène
- classe construite sur la notion de liste

```
df= data.frame(a=c(9,8,5),b=c(3,3,1),c=c(T,F,F),dernier=c("toto",NA,NA))
print(df)
```

```
##   a b     c dernier
## 1 9 3  TRUE    toto
## 2 8 3 FALSE    <NA>
## 3 5 1 FALSE    <NA>
```

```
dim(df)
```

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

```
nrow(df)
```

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

```
names(df)
```

```
## [1] "a"       "b"       "c"       "dernier"
```

```
class(df)
```

```
## [1] "data.frame"
```

```
mode(df)
```

Résumés de tables de données

```
head(df)
```

```
##   a b     c dernier
## 1 9 3  TRUE    toto
## 2 8 3 FALSE    <NA>
## 3 5 1 FALSE    <NA>
```

```
str(df)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ a      : num  9 8 5
##  $ b      : num  3 3 1
##  $ c      : logi  TRUE FALSE FALSE
##  $ dernier: chr  "toto" NA NA
```

```
summary(df)
```

```
##        a               b               c              dernier
##  Min.   :5.000   Min.   :1.000   Mode :logical   Length:3
##  1st Qu.:6.500   1st Qu.:2.000   FALSE:2         Class :character
##  Median :8.000   Median :3.000   TRUE :1         Mode  :character
##  Mean   :7.333   Mean   :2.333
##  3rd Qu.:8.500   3rd Qu.:3.000
##  Max.   :9.000   Max.   :3.000
```

Adressage de tables de données

```
df$dernier
```

```
## [1] "toto" NA     NA
```

```
df["dernier"] # différence avec le précédent ?
```

```
##   dernier
## 1    toto
## 2    <NA>
## 3    <NA>
```

```
max(df$a)
```

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

```
df[1:2,2:3] # nature de cet objet
```

```
##   b     c
## 1 3  TRUE
## 2 3 FALSE
```

```
identical(df$c,df[[3]])
```

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

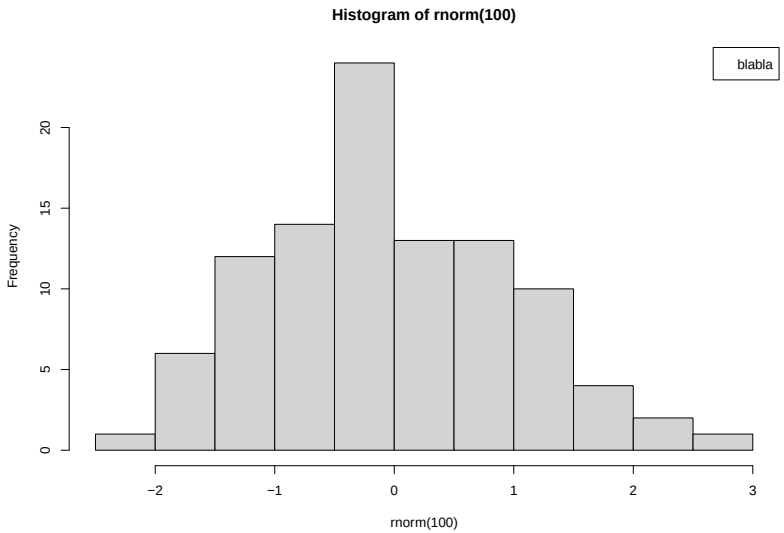
Graphiques, itérations, données, entrées/sorties, fonctions

Graphiques

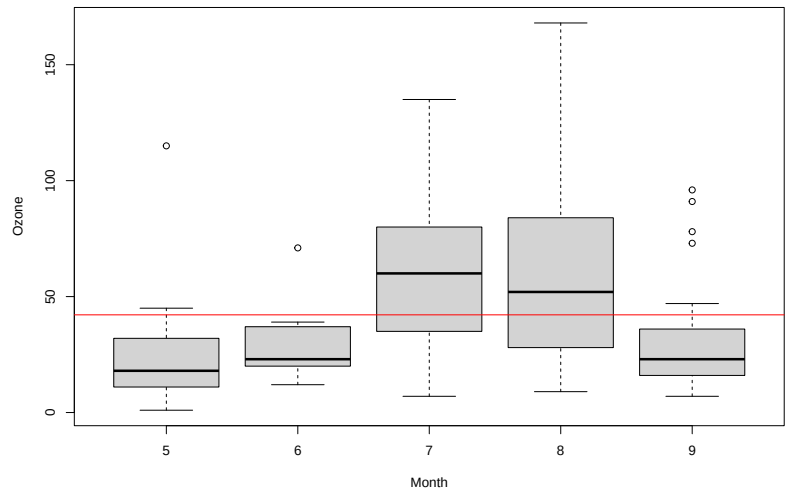
Le logiciel R propose de nombreuses fonctionnalités graphiques, dont un aperçu peut-être visualisé en jouant la commande **demo(graphics)**.

- **plot**, **boxplot**, **hist**, etc. différents graphiques primaires, axes, titres etc,
- **points**, **lines**, **text**, **legend**,... : enrichissement de ces graphiques.
- **ggplot2** : librairie avec grammaire différentes, graphiques de grande qualité.

```
hist(rnorm(100))
legend("topright",legend = "blabla")
```




```
boxplot(data=airquality,Ozone~Month)
abline(h=mean(airquality$Ozone,na.rm=T),
```



Boucles

Pour des raisons de performance, on les *évite* autant que possible en utilisant des opérations vectorisées déjà codées telles que les opérations matricielles (*,.,/,%%*) **apply**, **lapply**, etc mais parfois on ne peut pas y couper.

```
week <- c("Lundi","Mardi","Mercredi","Jeudi","Vendredi")
for (day in week)
{
    print(day)
}
```

```
## [1] "Lundi"
## [1] "Mardi"
## [1] "Mercredi"
## [1] "Jeudi"
## [1] "Vendredi"
```

```
x = 1
while (x<5) {print(x); x = x + 1}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

Voir aussi *repeat*, *break*

Boucles (suite)

Exemple de renormalisation de matrice en utilisant plusieurs CPU

```
library(foreach)
foreach(i=1:nrow(m), .combine=rbind) %dopar%
  (m[i,] / mean(m[i,]))
```

Import et export de données

- *read.csv*, *write.csv* pour les fichiers csv
- *read.table*, *write.table* pour les data.frame au format texte de R
- *load*, *save*, pour le format binaire
- *read_parquet*, *write_parquet* pour le format parquet (bibliothèque *arrow*)
-

Extension des commandes disponibles

La commande *function* permet de se créer un objet fonction sur mesure.

```
hingeloss <- function(x) {
  return(pmax(1-x , 0))
  # NB: pmax prend le max de deux vecteurs (après recyclage éventuel)
  # vectorisation toujours!
}

hingeloss(seq(-4,4))
```

```
## [1] 5 4 3 2 1 0 0 0 0
```

Une telle fonction pourra être passée en argument à d'autres fonctions.

```
maliste = list(3,-2,NA,c(-5,0,5,NA,Inf,-Inf))
lapply(maliste, hingeloss) # lapply applique une même fonction à chaque élement
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] NA
##
## [[4]]
## [1]   6   1   0  NA   0 Inf
```

3. TD : Introduction à la simulation numérique

Évaluation numérique d'intégrale par Monte-Carlo

Supposons qu'on souhaite évaluer une intégrale $I = \int_D f(u)du$ où D est un domaine de $\mathbb{R}^d$. On peut utiliser une méthode numérique par discrétisation, mais son écriture peut être difficile en fonction de la forme de D et la convergence est lente avec la dimension d. Les méthodes de Monte-Carlo sont très simples à mettre en oeuvre et permettent une approximation rapide. Mais l'erreur commise n'est contrôlée qu'en probabilité. Pour notre exemple, supposons qu'on sache facilement calculer si un point u est dans D, la valeur $f(u)$ et que l'on dispose d'une densité g non nulle sur D selon laquelle on sache simuler. Soit donc X une variable aléatoire de densité g. Alors

$$I = \int_D g(u)du = \int_D \frac{f(u)}{g(u)} f(u)du = \mathbb{E}\left(\mathbf{1}_D(X)\frac{f(X)}{g(X)}\right) .$$

Cette dernière espérance peut être approchée, en vertu de la loi des grands nombres, par

$$\hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_D(X_i)\frac{f(X_i)}{g(X_i)}$$

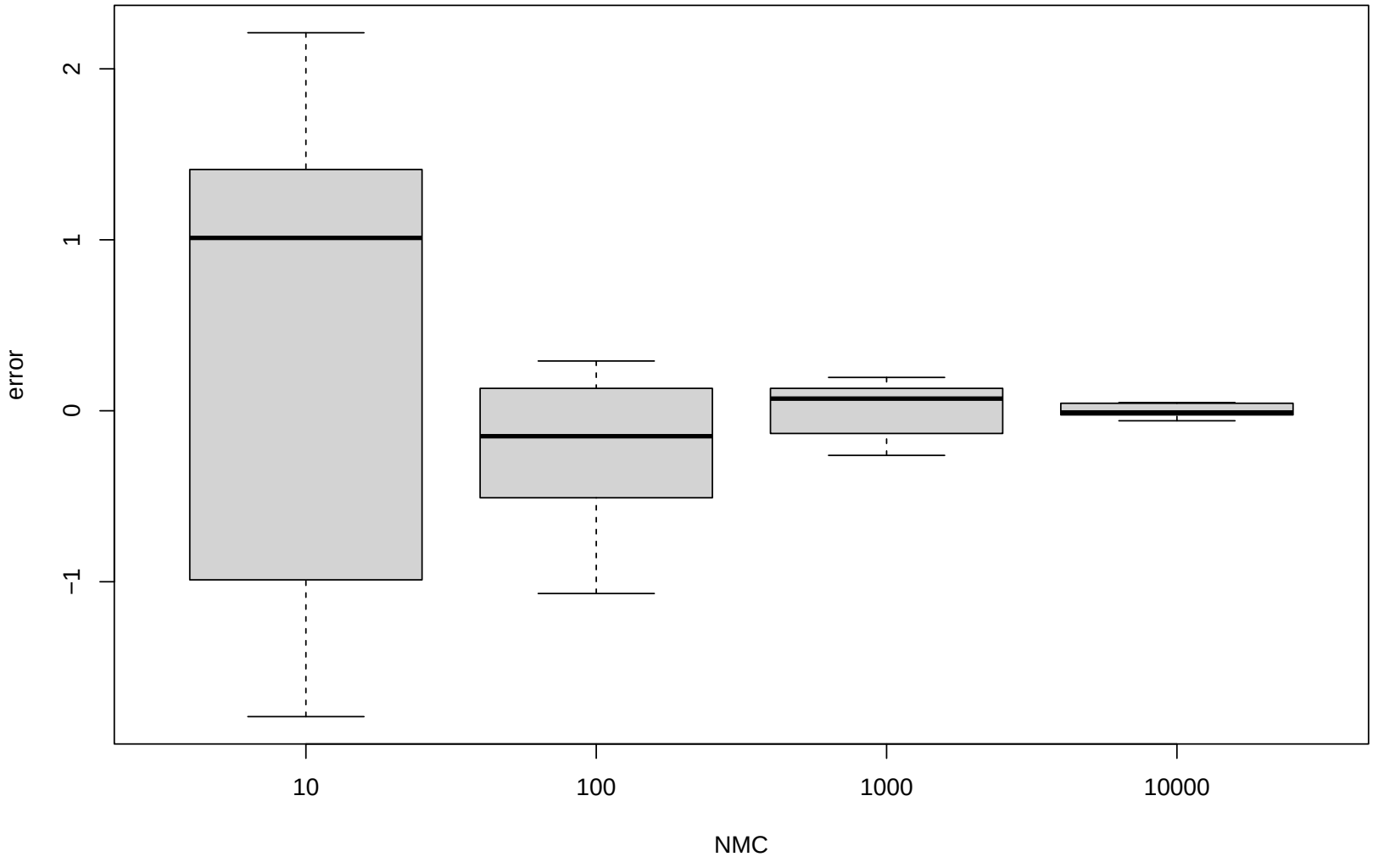
où $X_1, \ldots, X_n$ est un échantillon i.i.d. de densité g.

Exercice 1

Utiliser une simulation de Monte-Carlo pour évaluer le volume de la sphère unité (dont on sait, en fait qu'il vaut $4\pi/3$).

1. Simuler un échantillon i.i.d. sur le cube unité. On stockera ce vecteur dans une matrice de `NMC` lignes et trois colonnes contenant les coordonnées.
2. Le transformer pour obtenir un échantillon i.i.d. sur le cube $([-1,1])^3$ (on conservera ce stockage matriciel).
3. Ecrire une fonction qui renvoie la norme auclidienne d'un vecteur.
4. Au moyen de la commande *apply* appliquer cette fonction à toute les lignes de la matrice, pour récupérer le vecteur des normes de chaque point.
5. Evaluer la proportion de points de norme plus petite que 1 et en déduire une estimation du volume de la sphère unité de $\mathbb{R}^3$.
6. Calculer l'erreur pour différentes valeurs de `NMC`.
7. Représenter l'évolution de l'erreur

Exercice 1 (suite)



Exercice 2: Simulation de variables aléatoires par acceptation-rejet

Soit D le disque inscrit dans le carré $[0,1] \times [0,1]$

$$D = \left\{(x,y) \in \mathbb{R}^2 | (x-\tfrac{1}{2})^2 + (y-\tfrac{1}{2})^2 \leq \tfrac{1}{4}\right\} \ .$$

On souhaite simuler un vecteur aléatoire bidimensionnel de densité f sur $\mathbb{R}^2$

$$f : (x,y) \mapsto \frac{8}{\pi} y \mathbf{1}_D(x,y) \ .$$

On propose d'utiliser la méthode d'acceptation rejet vue en cours.

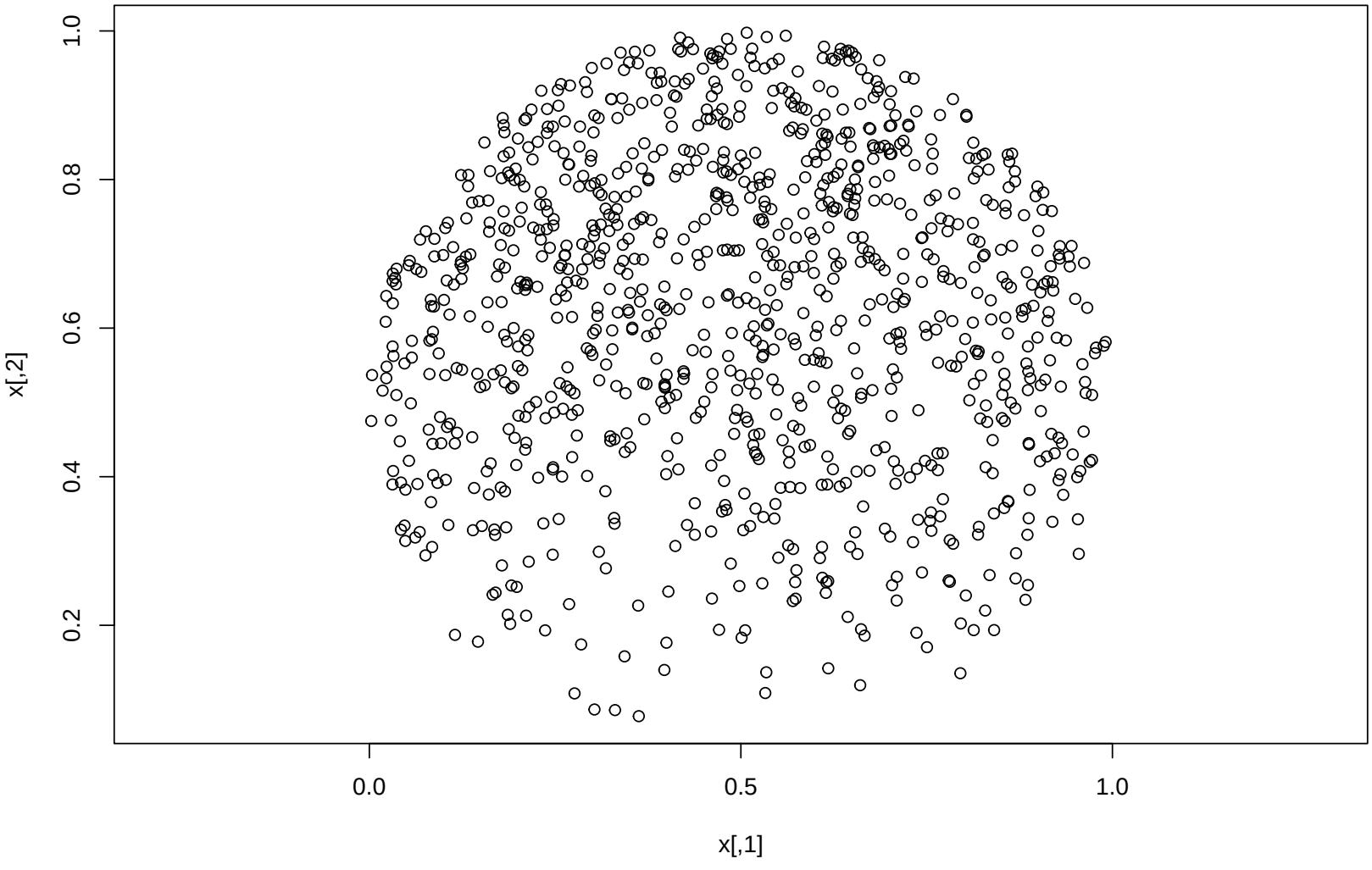
- Montrer que la fonction g uniforme sur le carré circonscrit au cercle est une densité dominante satisfaisante.On donnere une valeur M telle que $f(x)/g(x) \leq M$.
- Coder l'algorithme d'acceptation rejet.
- Afficher le nuage de points des tirages.
- Observer numériquement que le nombre moyen de tirages nécessaires est de l'ordre de $M+1$ fois la taille de l'échantillon visée.
- On peut précisément montrer que le nombre de tirages T nécessaires pour obtenir un échantillon selon f est distribué selon une loi géométrique de paramètre $p = 1/M$, au sens où

  $$\forall t \in \mathbb{N}^*, \mathbb{P}(T = t) = p(1-p)^{t-1}.$$

  On demande ici de le vérifier statistiquement en stockant le nombre d'essais nécessaires pour chaque acceptation et en réalisant un test d'ajustement du χ^2 sur les 5 premières modalités de T: $T = 1, T = 2, T = 3, T = 4$ et $T \geq 5$.

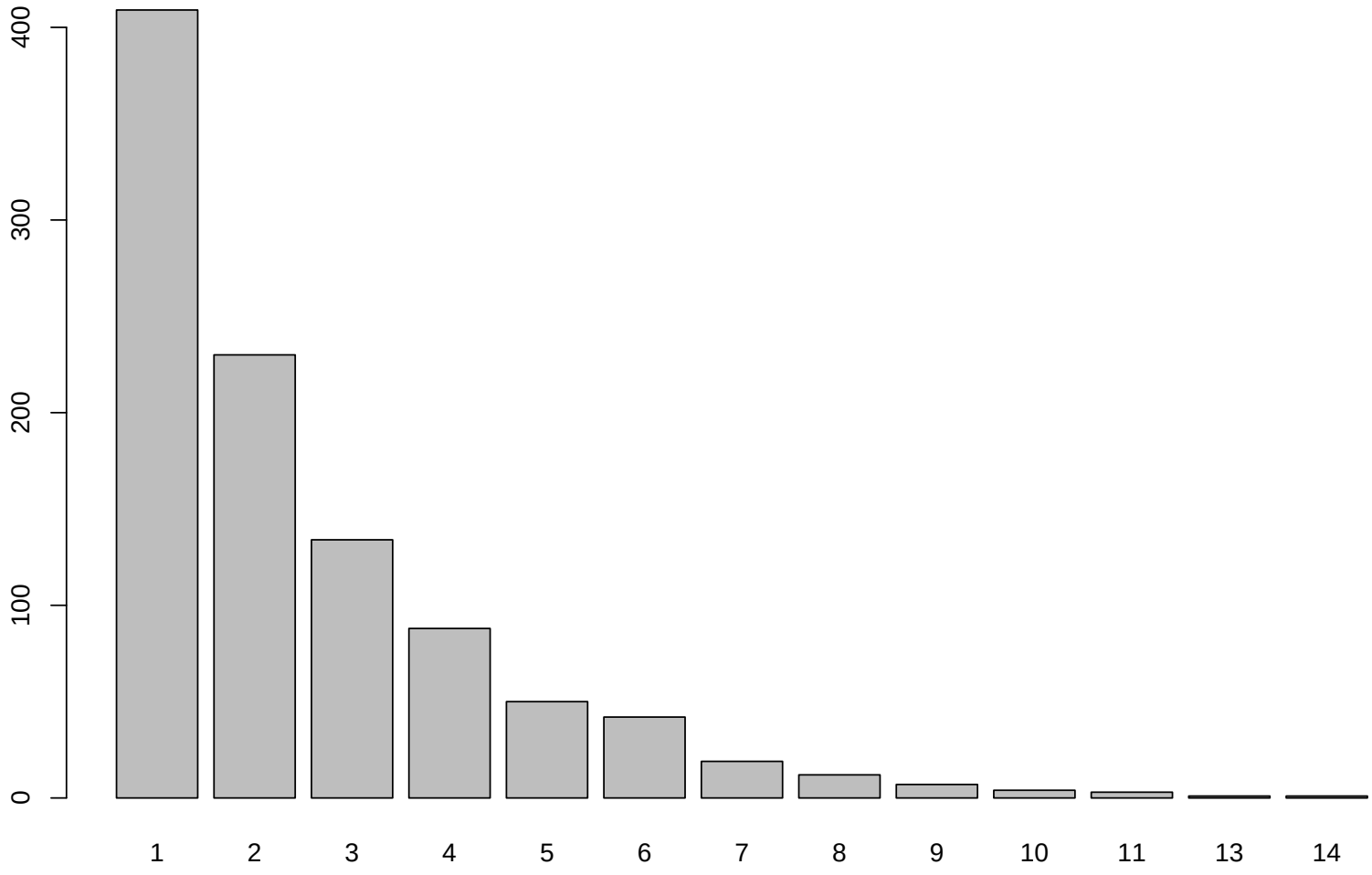
NB: les fonctions *pgeom*, *rgeom*, *dgeom* utilisent la convention qu'une loi géométrique décrit le nombre d'échecs avant le succès. Elle conviennent donc à décrire ou simuler $T' = T - 1$, à valeur dans $\mathbb{N}$.

Exercice 2 (suite)



Distribution du nombre de tirages

```
## [1] "Nb tirage moyen = 2.517"
```




```
## [1] "p-val = 0.762"
```