Initiation à R

3A Cursus Intégré - 2015
ENSAE ParisTech
Thomas Merly-Alpa (`thomas.merly-alpa@insee.fr`) -
Antoine Rebecq (`antoine.rebecq@insee.fr`)

« The best thing about R is that it was developed by statisticians. The worst thing about R is that... it was developed by statisticians. » - *Bow Cowgill*

1 Introduction

Ce premier TP a pour but de vous familiariser avec l'environnement R. Nous n'aborderons pas de fonctions compliquées mais nous allons essayer de faire le tour de différentes utilisations possibles du logiciel. N'hésitez pas à vous référer à votre poly pour vous aider/aller plus loin et à chercher sur le net les informations manquantes, pendant et en dehors du TP.

Note : Comme expliqué plus loin, le moyen privilégié pour rechercher des informations sur une commande, sa syntaxe, etc., est d'utiliser l'aide de R. Toutefois, il est souvent plus rapide de s'aider d'un moteur de recherche web pour trouver le nom du package ou de la fonction qui réalisera ce que vous attendez ("Google is your friend"). Attention toutefois à bien comprendre toute ligne de code que vous exécutez, pour ne pas tomber dans la programmation "cargo cult" [a] (typiquement, copier-coller sans les comprendre des lignes de code qui semblent fonctionner d'après un post sur *StackOverflow*).
Si votre IDE (*Integrated development environment*, le logiciel qui vous permet d'exécuter des commandes R) possède la fonction d'auto-complétion (comme par exemple la dernière version de *RStudio*), celle-ci est souvent églament très utile pour comprendre le mode de fonctionnement d'une commande.

a. `https://en.wikipedia.org/wiki/Cargo_cult_programming`

1.1 R, une calculatrice ?

L'utilisation la plus simple de R est de l'utiliser comme une « simple » calculatrice. On verra que c'est en fait une calculatrice très sophistiquée. Par exemple, tapez la commande suivante :

```
> 2+3
```

R vous répond immédiatement :

```
[1] 5
```

Si on laisse de côté le [1] qui sera expliqué plus loin, on obtient bien le résultat attendu. Naturellement, tous les opérateurs habituels sont disponibles, ainsi que les fonctions logarithmiques, trigonométriques, et bien d'autres.

Question 1. *Remplir le tableau suivant.*

Code en R	Résultat
a + b	Somme de a et b
a - b	
	Produit de a et b
a / b	Division réelle de a et b
a ** b	
a %/% b	
a %% b	
cos(a)	Cosinus de a
sin(a)	
tan(a)	Tangente de a
	Racine carrée de a
abs(a)	
log(a)	Logarithme naturel de a
exp(a)	Exponentielle de a

On peut affecter le résultat d'un calcul à une variable. Une variable est une entité qui porte un nom et qui contient une valeur. Par exemple, si on tape :

```
> a <- 2*sqrt(5) + pi
```

R n'affiche rien, mais il a :
— calculé la valeur de l'expression $2\sqrt{5} + \pi$;
— affecté cette valeur à une variable dont le nom est a.

L'affectation se fait donc par l'opérateur <-. À sa gauche, on indique le nom de la variable à laquelle il faut affecter la valeur qui se trouve à sa droite. En passant, vous aurez noté l'utilisation de *pi* pour obtenir la valeur de π. *pi* est une variable pré-définie dans R dont la valeur est celle de π. On peut afficher la valeur d'une variable en tapant simplement son nom :

```
> a
[1] 7.613729
```

Naturellement, on peut utiliser la valeur d'une variable pour effectuer des calculs, par exemple :

```
> b <- (a - 1)*(a + 1)
```

Question 2. *Combien vaut $b - a$ à 10^{-3} près ?*

Pour simplifier la saisie des commandes, vous pouvez utiliser la touche fléchée ↑ pour retrouvez les commandes que vous avez déjà tapées et pour les modifier. Ainsi, R garde l'historique des commandes que vous tapez. Vous pouvez remonter dans l'historique avec la touche ↑ et redescendre avec la touche ↓. Quand vous remontez dans l'historique, vous pouvez ensuite utiliser les touches fléchées ← et → pour éditer la commande.

Question 3. *Utilisez ces touches pour taper la commande suivante, qui diffère peu d'une commande que vous avez tapée plus haut :*

```
> 3 * sqrt (5 + pi)
```

Question 4. *Calculez les racines de l'équation suivante :*

$$2x^2 + 5x - 12 = 0$$

À tout moment une aide en ligne est accessible avec la commande ?topic, où topic peut être remplacé par n'importe-quelle commande. Essayez, par exemple ?log. Vous pouvez aussi utiliser help(topic). Tapez 'q' pour quitter l'aide. Il est très important d'apprendre à lire l'aide. L'aide en ligne contient énormément d'information : elle est là, disponible, c'est à vous d'apprendre à l'utiliser. Les pages d'aide ont toutes la même structure :

- une description rapide de la commande et, éventuellement, de commandes apparentées. Pour log, diverse commandes calculant le logarithme dans différentes bases et les exponentiations sont apparentées ;
- comment on utilise la commande ;
- les paramètres de la commande ;
- une description détaillée de la commande ;
- la valeur de la commande. Là, le comportement de la fonction dans des cas exceptionnels est indiqué ; ainsi, on apprend que log(0) a pour valeur $-\infty$;
- des remarques complémentaires ;
- des références bibliographiques où la manière dont la fonction est calculée est tirée ;
- d'autres commandes à consulter si l'on veut des informations complémentaires, sur d'autres fonctions proches ;
- des exemples : cette section est extrêmement utile car si l'on n'a pas compris les explications, on nous donne des exemples d'utilisation qui peuvent être directement copiées/collées dans R.

Question 5. *Répondre aux questions suivantes en utilisant la documentation :*

1. *Que donne* $\log(-4)$ *?*
2. *Que donne* $\mathrm{sqrt}(-3)$ *?*
3. *Que donne* $1/0$ *?*
4. *Que donne* $0/0$ *?*

5. *Comment calculer le log en base* 5 *de* 23 *?*
6. *Comment obtenir la valeur de e, la base du logarithme naturel ?*
7. *Que vaut* Arcsin(0.52) *?*

Jusqu'à maintenant, on n'a manipulé que des nombres réels. On peut aussi manipuler des nombres complexes directement : si l'on colle un i juste derrière un réel, il devient un nombre imaginaire. Ainsi, 2+3i est un nombre complexe, dont la partie réelle vaut 2, la partie imaginaire vaut 3. **Attention, le i doit être collé au nombre.** Pour savoir si R fait ses calculs dans les réels ou les complexes, la règle est la suivante : si vous n'utilisez que des nombres réels, R fait les calculs dans l'ensemble des réels. Si un complexe intervient, R fait ses calculs dans les complexes. Ainsi, on peut utiliser la notation 0i qui d'un point de vue mathématique n'a pas beaucoup d'intérêt, pour obliger R à faire ses calculs avec des complexes. Il existe des fonctions pré-définies spécifiques pour les complexes (partie réelle, partie imaginaire, module, argument, conjugué). Vous taperez ?complex pour en avoir une liste. Par ailleurs, toutes les fonctions réelles vues plus haut sont étendues aux complexes.

Question 6. *Calculer la somme de 4+2i et de 6+8i. L'affecter à une variable nommée* c. *Calculer l'argument et le module de* c.

Question 7. *Afficher la valeur de la racine carrée de -1, en utilisant la fonction sqrt.*

L'essentiel de ce qui est a connaitre sur la syntaxe des commandes R est d'ores et déjà vu ; on voit ainsi que cette syntaxe est très simple. On la résume ci-dessous :

— pour obtenir la valeur d'une expression : on tape cette expression ;
— pour affecter une valeur a une variable : nom de la variable ← valeur a affecter ;
— pour obtenir la valeur d'une variable : on donne le nom de la variable ;
— le nom d'une fonction est toujours suivie de parentheses entre lesquelles on indique les parametres de la fonction.

Question 8. *Tapez ls() et expliquez ce qu'il se passe. Tapez ensuite rm(c), puis c. Que se passe t-il ? Que se passerait-il maintenant si on entrait ls() ? Le vérifier.*

1.2 Les vecteurs

Après les simples nombres, un deuxième type fondamental d'entités dans R est le vecteur. Un vecteur peut-être crée de différentes manières. Par exemple, créons un vecteur dont les éléments sont 0, -1, π et 4,78, et mettons-le dans une variable dont le nom est v :

```
> v <- c(0, -1, pi, 4.78)
```

La fonction c assemble et construit un vecteur avec ces valeurs. On peut ensuite consulter la valeur des composantes du vecteur :

```
> v
[1] 0.000000 -1.000000 3.141593 4.780000
```

On retrouve bien les valeurs avec lesquelles on a initialise le vecteur. On peut également utiliser la fonction c() pour concaténer deux vecteurs.

Question 9. *Qu'obtient-on si l'on écrit c(v,v,v) ? En utilisant la documentation de la fonction rep(), écrire une commande équivalente à c(v,v,v).*

De nombreuses fonctions s'appliquent aux vecteurs. La table ci-dessous en contient un certain nombre.

Question 10. *Remplissez le tableau suivant en donnant les résultats obtenus sur le vecteur v.*

Code en R	Résultat	Valeur obtenue pour v
length(v)	Longueur de v	
min(v)	Minimum de v	
max(v)	Maximum de v	
range(v)	Renvoie le min et le max de v	
sum(v)	Somme de v	
prod(v)	Produit de v	
mean(v)	Moyenne de v	
which.min(v)	Rang du minimum de v	
which.max(v)	Rang du maximum de v	

R peut naturellement effectuer toutes les opérations habituelles sur les vecteurs. Les opérateurs vues plus haut sur les nombres (+, -, ...) et les fonctions (abs, log, ...) s'appliquent sur les vecteurs comme on s'y attend. Toutes ces opérations se font terme à terme : aussi, quand il y a deux vecteurs (comme pour une addition), il faut que les deux vecteurs aient le même nombre d'éléments.

Question 11. *Construisez un vecteur w de telle sorte que v + w renvoie :*

```
> v + w
[1] 1.000000 -2.000000 4.141593 5.780000
```

*Que donne alors v*w ? Et abs(w) ?*

Question 12. *Calculer v %*% w. Quel opérateur classique avons-nous appliqué à ces deux vecteurs ?*

Enfin, on peut ajouter un scalaire aux éléments d'un vecteur, la soustraire, la multiplier, ... par les opérateurs habituels (+, -, *, ...) en combinant un vecteur et un scalaire. Ainsi,

```
v + 3
```

produit un vecteur dont les composantes sont celles de v augmentées de 3.

Question 13. *Que dire du résultat de la commande v - mean(v) ?*

Note : Contrairement ~~aux vrais~~ à beaucoup de langages de programmation, les indices pour les vecteurs ou objets sont **numérotés à partir de 1**, et non pas 0.

On peut aussi demander la valeur d'un élément du vecteur à l'aide de l'opérateur [ ] :

```
v[3]
[1] 3.141593
```

On peut aussi demander la valeur des éléments 1 et 3 du vecteur en fournissant plusieurs indices, comme suit :

```
v[c(1, 3)]
[1] 0.000000 3.141593
```

On voit que pour cela, on a spécifié un vecteur dont les composantes sont les indices qui nous intéressent (c (1, 3) est un vecteur de deux éléments, le premier vaut 1, le second vaut 3). On peut également demander la valeur de tous les élements sauf les 2ème et 4ème par exemple, en utilisant le signe - de la façon suivante :

```
v[-c(2, 4)]
[1] 0.000000 3.141593
```

On obtient bien le même résultat. On vient donc de présenter une opération très utile et très importante en R : accéder aux élements d'un vecteur en utilisant un autre vecteur (dénommé « vecteur d'index ») qui contient le numéro des composantes que l'on veut sélectionner. Une autre manière de spécifier les indices qui nous intéressent est la suivante :

```
> 2:4
```

est un vecteur dont les éléments sont 2, 3 et 4. Une autre notation pour la même chose consiste à utiliser la fonction seq(). Ainsi,

```
> seq(2, 4)
```

ou

```
> seq(from=2, to=4)
```

produit un vecteur dont les éléments sont 2, 3 et 4.

L'intérêt de cette fonction sequence et que l'on peut lui indiquer plus généralement la valeur du premier indice, du dernier indice et de l'incrément pour passer de l'un à l'autre. Ainsi :

```
> seq(from = 1, to = 10, by = 2)
[1] 1 3 5 7 9
```

l'incrément peut aussi être négatif :

```
> seq(from = 10, to = -5, by = -2)
[1] 10 8 6 4 2 0 -2 -4
```

Question 14. *Afficher de deux façons différentes, en utilisant seq() et sans l'utiliser, les éléments d'indice pair du vecteur v.*

Question 15. *Comparez rev(v) et v[seq(4,1,-1)]. Que pouvez-vous en déduire sur la fonction rev() ?*

La fonction runif() génère des nombres pseudo-aléatoires depuis une distribution de probabilité uniforme. On va l'utiliser pour générer des vecteurs aussi grands que l'on veut. Ainsi, tapez :

```
> vu <- runif(100)
```

et la variable *vu* contient un vecteur de 100 composantes dont la valeur a été tirée au hasard entre 0 et 1.

Question 16. *En utilisant la documentation, construire un vecteur wu de taille 100 dont les composantes sont tirées au hasard entre 1 et 2.*

Question 17. *Répondre aux questions suivantes :*

1. *Quelle est la somme des composantes du vecteur vu ?*
2. *Quel est l'écart entre le maximum de vu et le minimum de wu ?*
3. *Quelle est la somme des composantes d'indice pair de vu ?*
4. *On veut obtenir le vecteur vs dont la valeur de la composante k est la somme des composantes $2k-1$ et $2k$ de vu. Comment faire ?*
5. *Calculer la deuxième plus petite valeur - celle juste supérieure au minimum - de wu.*

En statistique et exploration de données, un vecteur va typiquement contenir des données sur lesquelles on voudra effectuer des traitements statistiques. Les plus simples sont mean(), qui calcule la moyenne, var(), la variance, sd(), l'écart-type et summary().

Question 18. *Que se passe t-il quand on entre summary(vu) ? median(vu) ? quantile(vu) ?*

Question 19. *Donner le 97ème centile de vu.*

Outre les nombres, R manipule des booléens, qui prennent soit la valeur TRUE (que l'on peut abréger en T), soit la valeur FALSE (que l'on peut abréger en F). Les booléens apparaissent typiquement à l'occasion du test de la valeur d'une variable : est-elle égale à une autre valeur, plus petite, plus grande, ... La table ci-dessous indique ces opérateurs :

Code en R	Résultat
a == 1	Teste si a vaut 1
a != 1	Teste si a différent de 1
a < 1	Teste si a strictement plus petit que 1
a <= 1	Teste si a plus petit ou égal que 1

Bien sûr, on peut comparer la valeur de deux variables (donc, écrire a == b qui donne FALSE ici). De la même manière, on peut effectuer ces opérations sur les éléments d'un vecteur. Ainsi,

```
> v < w
[1] TRUE FALSE FALSE FALSE
```

Tout comme plus haut on a sélectionné les éléments d'un vecteur avec un vecteur d'indices, on peut sélectionner les éléments d'un vecteur avec un vecteur de booléens. Entrer

```
> v[c(TRUE,TRUE,FALSE,TRUE)]
```

revient à écrire

```
> v[c(1,2,4)]
```

Ainsi, si l'on veut affecter à 100 les éléments de v qui sont plus petits que ceux de w, on pourra écrire :

```
> v[v < w] <- 100
```

Cette opération ne modifiera que le premier élément de v, qui correspond à la seule valeur TRUE du vecteur de booléens utilisé pour l'indexer.

Question 20. *Combien le vecteur vu a t-il de composantes dont la valeur est plus grande que 0,7 ?*

Il existe des opérations sur les booléens, celles de l'algèbre de Boole : ET, OU, NON, ... On suppose que l'on a défini deux vecteurs booléens comme suit :

```
> bool1 <- c (T, T, F)
> bool2 <- c (F, T, F)
```

Les principales opérations logiques sont symbolisées par &, | et !.

Question 21. *Que vaut :*
- *bool1 & bool2 ?*
- *bool1 | bool2 ?*
- *!bool1 ?*

En déduire le sens des trois opérateurs logiques.

Question 22. *Quelle est la somme des composantes de vu qui sont comprises entre 0,20 et 0,60 ? Est-elle plus grande que celle des composantes entre 0,40 et 0,70 ?*

1.3 Conversions

Les objets manipulés par R ont des types. On peut tester le type d'un objet en utilisant les fonctions du type is.numeric() :

```
> is.numeric(4)
[1] TRUE
> is.numeric("A")
[1] FALSE
```

On peut transformer un élement d'un type en un autre en utilisant des fonctions du type as.numeric(), par exemple avec des booléens :

```
> is.numeric(F)
[1] FALSE
> as.numeric(F)
[1] 0
```

Question 23. *Que vont donner les commandes suivantes ?*

```
> is.character(4)
> is.character("F")
> is.character(F)
```

Le vérifier.

2 Création d'objets

2.1 Création de vecteurs

En utilisant les fonctions présentées dans la partie précédente, et sans rentrer les éléments du vecteur à la main, on peut maintenant créer des vecteurs dans R.

Question 24. *Créer le vecteur x0 composé de 500 zéros, le vecteur x1 tel que*

```
> x1
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
[30] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
[56] 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
[82] 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

et le vecteur x2 tel que

```
> x2
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
 [21]  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
 [41]  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
 [61]  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
 [81]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
[101] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83  82  81
[121]  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65  64  63  62  61
[141]  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41
[161]  40  39  38  37  36  35  34  33  32  31  30  29  28  27  26  25  24  23  22  21
[181]  20  19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
```

Que signifient les nombres entre crochets ?

En plus de la fonction runif présentée dans la partie précédente, il existe d'autres fonctions permettant de générer des réalisations de lois de probabilités. Par exemple, on peut utiliser rnorm(), rpois()...

Question 25. *Créer un vecteur z à 20 éléments, composé de nombres aléatoires extraits d'une loi normale de moyenne 3 et de variance 1. Créer un vecteur z1 dont la composante k vaut 0 lorsque la composante k de z est plus petite que 3, et 1 sinon.*

2.2 Création de matrices

Pour construire une matrice dans R, on utilise la fonction matrix(). Celle-ci prend un vecteur en entrée, ainsi que deux valeurs correspondant aux dimensions de la matrice voulue. Par exemple, on a :

```
> matrix(1:6,2,3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Question 26. *Quelque chose devrait vous choquer dans le résultat précédent. En utilisant la documentation de matrix(), résolvez le problème.*

La fonction dim(m) renvoie le vecteur composé des dimensions de la matrice m. La fonction t(m) permet de calculer la transposée de la matrice. Enfin, il existe quelques astuces pour construire des matrices usuelles : n'hésitez pas à consulter help(diag) avant de répondre à la question suivante.

Question 27. *Créer une matrice identité de dimension* 8×8.

Construisons la matrice suivante :

```
> m <- matrix(1:25,5,5,byrow=T)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
```

Question 28. *Donner la commande R renvoyant la troisième colonne de m, puis celle renvoyant la quatrième ligne. Que faut-il écrire à la place des ? pour obtenir :*

```
> m[?,?]
[1] 9
```

Question 29. *Nommer les colonnes de m C1 à C5, et ses lignes L1 à L5. Afficher les valeurs de la première colonne de m en utilisant son nom.*

2.3 Les data.frame

Commençons par écrire :

```
> donnees <- data.frame(m)
```

Si l'on demande à R d'afficher *donnees*, on ne voit aucune différence avec m. Pourtant, nous avons changé la structure des données. En particulier, il est désormais possible d'utiliser l'opérateur $ comme suit :

```
> donnees$C2
[1]  2  7 12 17 22
```

L'autre principal avantage est qu'il est possible d'utiliser des data.frame pour stocker des données qui n'ont pas le même type : par exemple, des nombres et des caractères.

Question 30. *Après avoir consulté help(LETTERS), créer un data.frame d dont la première colonne est composé des chiffres de 1 à 6 et la deuxième colonne des 6 premières lettres de l'alphabet, de telle sorte que l'on ait :*

```
> d$id
[1] 1 2 3 4 5 6
> d$initiales
[1] A B C D E F
Levels: A B C D E F
```

On tire ensuite un vecteur note dont les 6 éléments sont des réalisations d'une loi normale de moyenne 12 et de variance 1.

Question 31. *Rajouter au data.frame d la colonne note. Quelle est la note obtenue par l'individu dont l'initiale est D ?*

2.4 Les listes

Les listes sont des outils de R qui permettent de stocker ensemble plusieurs éléments de natures différentes. Pour construire une liste, on utilise la fonction list(). Pour accéder aux éléments d'une liste, on utilisera l'opérateur [[ ]].

Question 32. *Construire une liste l comportant les éléments x0, x1, x2 et d. Comment accéder à x2 à partir de l ? Construire une liste lx contenant les éléments x0, x1 et x2.*

Lorsqu'une liste contient des éléments qui sont tous de même nature, on peut appliquer à tous ses élements une même fonction. On utilise pour cela la fonction lapply().

Question 33. *À l'aide de la documentation de lapply, renvoyer une liste contenant les moyennes de x0, x1 et x2.*
Question subsidiaire pour utilisateurs avancés : Changer la valeur de x2 après avoir créé l, et observer l'effet sur l. L'objet list semble-t-il utiliser des références ou des copies ?

Il existe des variantes de lapply() telles que sapply() qui permettent de varier les formats d'entrée et de sortie de la fonction : vous pouvez vous reporter à l'aide lorsque vous avez besoin de les utiliser.

2.5 Création de fonctions

Une fonction en R correspond à une application qui prend en entrée un ou plusieurs objets et renvoie un objet (ou non) en sortie. La syntaxe d'une fonction est la suivante :

```
> f <- function(x) {
   return(2*x+4)
 }
> f(4)
[1] 12
```

On peut déclarer pour certaines entrées des valeurs par défaut, qui sont choisies si rien d'autre n'est spécifié :

```
> f <- function(x,a=4)  {
   return(2*x+a)
 }
> f(4)
[1] 12
> f(4,3)
[1] 11
```

Les fonctions peuvent prendre en argument et renvoyer en sortie différents types d'objets R.

Question 34. *Écrire une fonction g qui prend un nombre positif en entrée, et renvoie le vecteur constitué de sa racine carrée et de son carré. Que se passe t-il si l'on entre g(-1) ?*

Nous aimerions bien éviter le problème précédent. Pour cela, il nous faut utiliser l'instruction if/else. Celle-ci permet de tester une condition avant de réaliser la fonction. Par exemple :

```
> f <- function(x) {
   if (x == 2) {return(2)}
   else {return(1)}
 }
> f(1)
[1] 1
> f(2)
[1] 2
> f(3)
[1] 1
```

Question 35. *Améliorer la fonction g pour que quand l'on donne un nombre négatif en entrée, celle-ci ne renvoie que le carré du nombre.*

La fonction sapply() évoquée précédemment prend ici tout son intérêt. Par exemple, on peut faire :

```
> sapply(1:5,f)
[1] 1 2 1 1 1
```

Question 36. *Construire une fonction h qui à un nombre naturel n renvoie le déterminant d'une matrice de taille $n \times n$ dont les composantes sont des réalisations d'une loi uniforme sur [0,1]. Que se passe t-il si l'on appelle plusieurs fois h(3) ?*

Question 37. *Construire une fonction hvect qui, prenant en entrée n, renvoie le vecteur h(1), . . . , h(n).*

Lorsque vous souhaitez connaître le type d'un objet de votre namespace, vous pouvez utiliser la commande *str*. Par exemple :

```
> str(d)
'data.frame': 6 obs. of  3 variables:
 $ id       : int  1 2 3 4 5 6
 $ initiales: Factor w/ 6 levels "A","B","C","D",..: 1 2 3 4 5 6
 $ note     : num  12.4 13 12.5 11 12.5 ...
```

Optionnel : installation de RStudio

Rstudio est un IDE libre et gratuit pour R qui présente de nombreuses fonctionnalités très pratiques : visualisation des graphiques, des dataframe, gestion des objets du namespace, auto-complétion, etc. Pour les utilisateurs avancés, RStudio intègre quelques fonctionnalités incontournables, notamment en ce qui concerne la création de *packages*.

Pour la suite de ce TD (ainsi que pour vos travaux futurs de statistiques sous R), il peut être intéressant de prendre le temps d'installer RStudio. Le logiciel peut être téléchargé depuis le lien suivant : `https://www.rstudio.com/products/rstudio/download/`

3 Fonctionnalités de R

3.1 Lecture et écriture de données

Pour commencer, nous allons spécifier le dossier dans lequel R doit lire et enregistrer les données que nous allons manipuler. Pour cela, nous utilisons la commande suivante (après avoir remplacé le chemin par un chemin valide, sans oublier d'utiliser des / comme barres obliques).

```
> setwd("C:/Users/XXXXX/Desktop/TD R")
```

R peut lire les données stockées dans des fichiers texte (ASCII) grâce, entre autres, aux fonctions suivantes : read.table(), scan(), read.csv().

```
> meubles <- read.csv("meubles.csv")
> meubles
  Taille Poids Prix
1      5     6    4
2      4     7    6
3      8     8   12
4      5     4    4
5      5     6    5
6      6     4    7
7      7     8    9
```

Lorsque les données ont été sauvegardées sous le format propriétaire d'un logiciel statistique tiers, il est nécessaire de disposer d'outils permettant leur transfert vers le système. La librairie foreign offre ces outils pour une sélection des logiciels statistiques les plus courants, tels SPSS et Stata. Par exemple, la fonction read.spss prend en charge les données enregistrées au moyen des commandes save et export de SPSS. Par ailleurs, la fonction read.xls du package gdata fournit les outils pour lire des fichiers au format Excel. Pour plus d'informations,le document proposé par le « R development core team » est disponible

gratuitement sur internet à l'adresse suivante : `http://www.r-project.org`.

Pour enregistrer des données, on utilise les fonctions write.table, write.csv, par exemple.

Question 38. *Le responsable du jeu de données précédent souhaite avoir les poids en grammes plutôt qu'en kilogrammes. Modifier la table meubles comme souhaité, puis enregistrez-là sous le nom "meubles2.csv".*

3.2 Utilisation de packages

R est principalement utile car il permet d'utiliser de nombreux packages qui permettent de réaliser de nombreuses choses : de l'analyse de données, de l'économétrie, de l'actuariat... La liste des packages disponibles se trouve à l'adresse suivante : `https://cran.r-project.org/web/packages/available_packages_by_name.html`. On peut y consulter la documentation des packages, qui résume les fonctions disponibles et donne des exemples de leur utilisation. Pour installer un package sur sa machine, on a besoin de réaliser une fois l'opération suivante :

```
install.packages("FactoMineR")
```

R demande alors de choisir un site miroir de téléchargement, puis installe le package. Pour l'utiliser, il faut ensuite à chaque fois que l'on lance la session R utiliser la commande :

```
library(FactoMineR)
```

Il est recommandé d'écrire en haut de son fichier .R toutes les commandes library qui sont utilisées par le programme, afin de pouvoir relancer le programme à la prochaine ouverture de session.

3.3 Module graphique

R permet d'afficher des graphiques de différentes sortes. Nous allons commencer par nous intéresser à la fonction plot().

Question 39. *Créer une matrice, mat1, composée de 100 lignes et deux colonnes ; chacune des colonnes des vecteurs aléatoires de loi normale centrée et de variance 1.*

Créer une matrice, mat2, composée de 100 lignes et deux colonnes ; chacune des colonnes étant des vecteurs aléatoires de loi normale de moyenne 10 et de variance 1.

Tracer le graphe bivarié de la première colonne de mat1 sur la deuxième colonne de mat1.

Lorsque l'on fait apparaître plusieurs jeux de données sur le même graphique, on peut utiliser des couleurs différentes pour faciliter la lecture. Pour connaître la liste des couleurs disponible sur R, tapez la commande colors().

Question 40. *Ajouter au graphe précédent le graphe bivarié de la première colonne de mat2 sur la deuxième colonne de mat2.*
Indication : Avec le module graphique standard de R, chaque instruction plot *ouvre une nouvelle fenêtre graphique. Pour combiner un nouveau nuage de points sur une fenêtre graphique déjà ouverte, vous pouvez par exemple utiliser l'instruction* lines*, en spécifiant l'argument* type="p".
Que constatez-vous ? Utiliser les arguments xlim et ylim pour contourner cette problématique.

Intéressons-nous maintenant à la fonction hist().

Question 41. *Générer un vecteur poisson de taille 1000 dont les composantes sont des réalisations d'une loi de Poisson de paramètre 4. Qu'observez-vous en faisant hist(poisson) ?*

Question 42. *Comment fixer le pas de l'histogramme obtenu à 2 ?*

De meilleurs modules graphique sont disponibles, par exemple dans les librairies lattice ou ggplot2.

Question 43. *Refaire les questions du module graphique en utilisant le package* ggplot2 [1].

1. `http://docs.ggplot2.org/current/`

4 Quelques questions plus avancées

R et vitesse de calcul

R est avant tout pensé pour le calcul vectoriel, ce qui peut donner lieu à des temps de calcul curieusement longs pour certaines opérations lorsque le code n'a pas été écrit en "pensant R". Cette façon de coder est parfois contre-intuitive lorsque l'on vient d'autres langages de programmation (C, Java, etc.). Nous étudions ici deux des sources les plus fréquentes de frustration.

La première est relative à l'utilisation des boucles for. Une boucle for consiste en la réalisation d'une opération un nombre donné de fois, avec un compteur, souvent appelé i, évoluant dans un jeu de valeurs spécifié par un vecteur, souvent de la forme (1,...,n). En R, par exemple, attribuer à un vecteur préexistant v des valeurs allant de 1 à n se fait de la façon suivante :

```
for (i in 1:n) {
    v[i] <- i
  }
```

Question 44. *On cherche à calculer la somme des logarithmes des nombres entiers de 2 à n. Écrire une fonction réalisant ce calcul à l'aide d'une boucle* for. *Écrire une autre fonction réalisant ce calcul à l'aide d'une opération vectorisée. Tracer un plot en échelle logarithmique (ou semi-logarithmique) indiquant les temps d'exécution des deux fonctions pour n variant de* 10^4 *à* 10^8 *(avec le pas de votre choix).*

Question 45. *Écrivez deux fonctions calculant à l'aide d'une boucle* for *les termes de la suite de Fibonacci de 1 à n. Chaque fonction doit retourner un vecteur contenant les n premiers termes de la suite.*
Dans la première fonction, le vecteur retourné sera initialisé par les seules deux premières valeurs, et augmenté par concaténation à chaque passage de boucle. Dans la seconde, vous initialiserez le vecteur retourné dès l'appel de la fonction par un vecteur de taille n contenant les deux premières valeurs de la suite et des zéros de la position 3 à la position n.
Comparer les temps d'exécution des deux fonctions de la même manière qu'à la question précédente. Qu'en déduisez-vous : en R, vaut-il mieux pré-allouer toute la mémoire dont vous avez besoin ou augmenter petit à petit la taille des objets ? Souvenez-vous que ceci restera vrai pour les objets plus complexes (et plus utiles en statistique) tels que les dataframe, les matrices, etc.
(Question subsidiaire : pensez-vous qu'il est possible de calculer les termes de la suite sans utiliser une boucle for *?)*

Si vous souhaitez tout connaître en optimisation de code R, l'e-book "R-inferno", disponible gratuitement à l'adresse `www.burns-stat.com/pages/Tutor/R_inferno.pdf` regorge d'informations utiles.