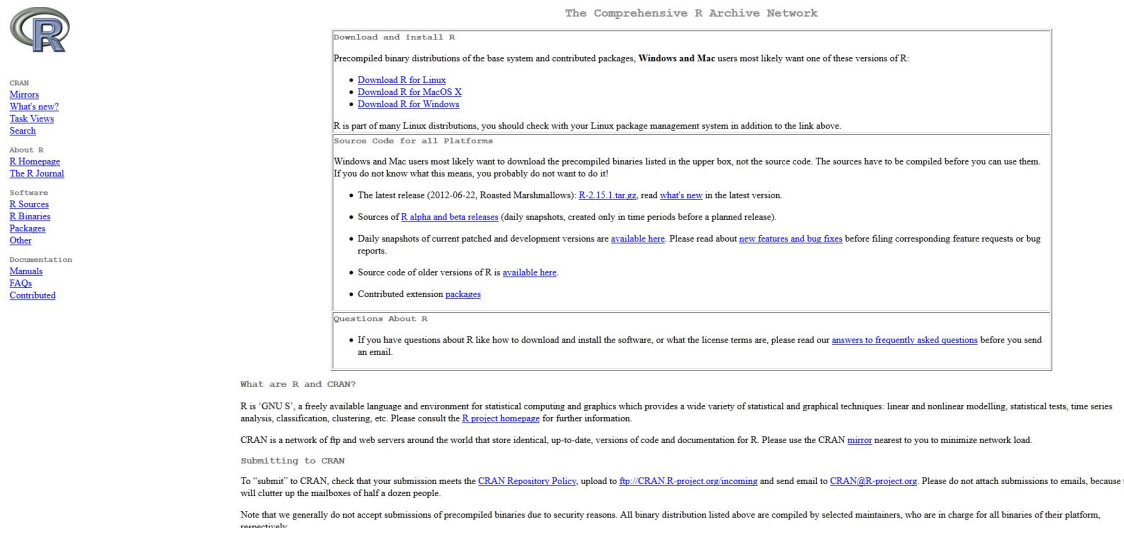
TP1 : INTRODUCTION À R

RÉSUMÉ. R est un logiciel de calcul numérique, utilisé dans de nombreux domaines d'application et open source. Il est basé sur le calcul matriciel. Le but de ce TP est d'introduire les commandes les plus courantes.

1. Installation du logiciel de base

1.1. **Le logiciel R.** Le logiciel R est disponible sur le site internet du CRAN -*Comprehensive R Archive Network*- à l'adresse `http://cran.r-project.org/`. Il s'agit d'un logiciel libre, open source et donc utilisable et modifiable gratuitement. Il existe une version pour les 3 OS : Linux, Windows et Mac OS.

Ce logiciel est massivement utilisé dans la communauté statistique et notamment parmi les chercheurs. Ces derniers sont de plus en plus nombreux à développer et diffuser des packages R pour mettre à disposition de tous leurs travaux de recherche.



Installation sous Windows : télécharger l'exécutable pour la bonne version de windows en cliquant sur l'onglet "Download R for Windows" puis "Base" puis "Download R 2.15.1 for Windows". Lancer ensuite l'exécutable en choisissant les options désirées. Le lgociel est ensuite installé avec possiblement l'interface graphique associée appelée R-Gui (R graphical user interface). Si vous choisissez de configurer les options de démarrage vous pouvez : choisir l'aspect de de Rgui -une ou deux fenêtres-, le format de l'aide -texte ou html-, spécifier l'accès internet -pour le téléchargement de package notamment-, créer une sur le bureau et dans la barre de lancement rapide.

Installation sous Linux : rechercher si il existe un paquet *rpm* adapté à votre distribution de linux à l'adresse `http://cran.cict.fr/bin/linux/`, télécharger et installer avec les droit admin en utilisant la commande "rpm -i R*.rpm". Si il n'existe pas de binaire compilé sur le

site, il faut compiler votre version de R. Pour cela, télécharger le code source R-2.15.1.tar.gz. Ouvrir en tant qu'admin une console à l'endroit où le fichier a été enregistré et lancer la commande :

```
cp R-2.15.1.tar.gz /mon_dossier/
cd /mon_dossier/
tar zxvf R-2.15.1.tar.gz
cd R-2.15.1 ./configure
make
```

1.2. **Chargement des packages.** Dans sa version de base, R permet d'effectuer un certains nombre d'opérations de base : opérations algébriques essentielles (addition, produits, produit scalaire, inversion de matrice, transposé...), opérations logiques, statistiques élémentaires (moyennes, variances, régression...). Toutefois, nous serons souvent amené à utiliser des techniques statistiques élaborées qui nécessiterons le chargement dans R de packages supplémentaires.

Chargement d'un package par internet

Choisir dans le menu déroulant : "Installer le(s) package(s)" :

Choisir ensuite le miroir le plus proche :

Chargement d'un package à partir du disque dur

Cette option est utile si vous travaillez hors ligne et que vous avez préalablement chargé un package sur votre disque dur. Dans ce cas choisissez dans le menu déroulant "Installer le(s) package(s) depuis des fichiers zip...". Indiquez ensuite dans le browser l'emplacement du fichier zip en question.

Sous Linux, télécharger le package sous forme de fichier .tar.gz, puis en tant qu'admin executer :

R CMD INSTALL package.tar.gz

Lorsqu'un package a été installé, pour la charger dans R, il faut taper la commande library(nom du package).

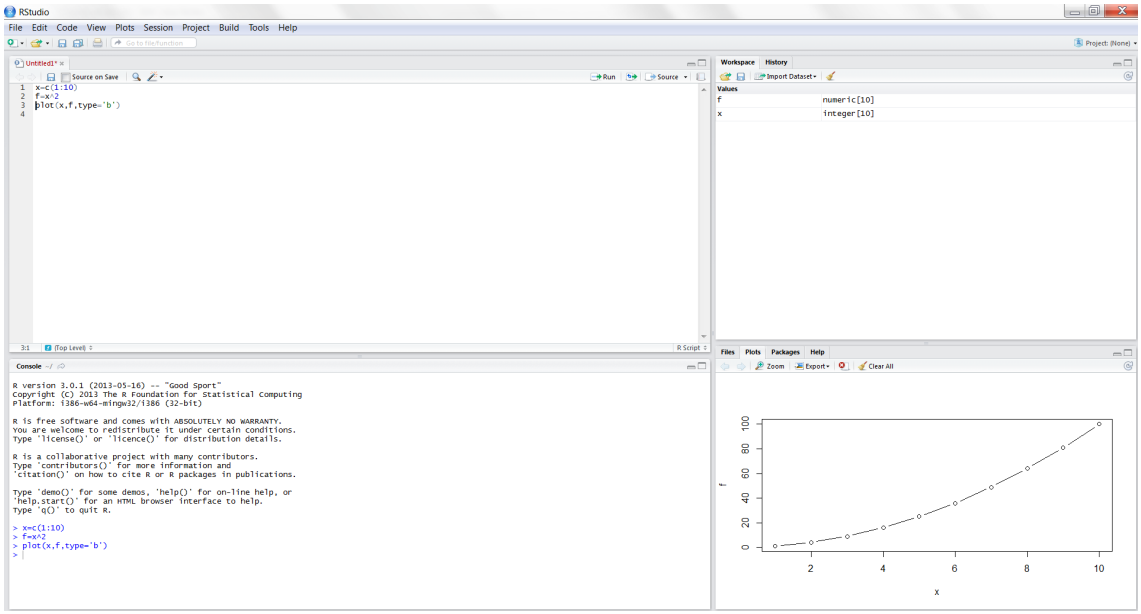
Exemple :

> library(mgcv) This is mgcv 1.7-18. For overview type 'help("mgcv-package")'

1.3. **L'interface RStudio.** Dans ce document nous présentons l'utilisation de R sous l'interface RStudio. Cette interface rend pratique et conviviale l'utilisation de R (assez austère dans sa présentation originale). Pour plus de détails, voir le site `http://www.rstudio.com/`.

2. Accès et commandes générales

2.1. **Lancement.** Pour lancer R, double-cliquer sur l'icône RStudio (ou ouvrir une fenêtre de commande et taper `r &` (le & permettant de garder la main dans la fenêtre de commande, cette solution est moins attractive que RStudio). Un environnement s'affiche à l'écran sous vos yeux emerveillés. Il est composé en général des fenêtres suivantes (de haut en bas, de gauche à droite) :



- Source : fenêtre d'édition du code,
- Workspace : qui vous donne les variables en mémoire, leur type et leur taille,
- Console : la console correspondant à l'environnement `R` en cours
- Help : une fenêtre multiple ou s'affiche l'aide, les graphiques, les fichiers du répertoire de travail.

Il est possible d'organiser cet environnement selon vos préférences.

2.2. **Gestion de l'espace de travail.** Pour connaître l'espace de travail dans lequel vous évoluez, utilisez la commande `getwd()`. Vous pouvez définir un espace de travail soit à l'aide de la commande `setwd("C:/Users/MAO/")` soit dans le menu déroulant : Session-Set Working Directory-Choose Directory. Les fichiers de données chargés dans la session ou les données exportées ou enregistrées le seront ensuite sous ce répertoire.

2.3. **Gestion de la fenêtre Source.** Lorsque vous voulez exécuter une commande vous pouvez soit taper cette commande directement dans la console (non recommandé car vous ne garderez pas trace de cette commande) soit taper cette commande dans un fichier source que vous exécuterez ensuite. Positionnez le curseur sur la ligne à exécuter puis tapez ctrl+enter ou cliquez sur Run.

2.4. **Calculs élémentaires.** Dans la partie console de l'interface, taper
> `5+8`
Résultat : > `13`
Pour conserver le résultat, il faut l'assigner dans un objet :
> `a=5+8`
> `a`

Il existe des variables prédéfinies, comme `pi` (3.1415..), `Inf` (nombre infini), `NaN` (n'est pas un nombre, exprime une indétermination, une valeur manquante).

2.5. **L'aide dans R.** Mieux vaut apprendre à se repérer tout seul que de demander en permanence à son voisin comment faire. Ne serait-ce qu'au cas où il faudrait utiliser dans l'examen une fonction dont on ne se souvient que vaguement quelle est sa syntaxe...

Pour accéder à l'aide de R, cliquer sur help dans la fenêtre d'aide de RStudio. L'aide est composer de différents répertoire : An Introduction to R, The R Language Definition ,Writing R Extensions R Installation and Administration, R Data Import/Export... Un façon de se repérer dans l'aide est de la parcourir par package.

En plus de l'aide "officielle" de R, des forums d'échange existes et sont très utiles, par exemple `http://stats.stackexchange.com/` ou `http://www.r-bloggers.com/`.

Vous pouvez accéder directement à l'aide concernant une fonction en tapant (ici avec la fonction plot) :
> `?plot`

ou, si vous chercher un terme sans connaitre le nom exact de la fonction en tapant :
> `help.search("plot")`

Exercice 1. Trouvez la fonction qui donne les valeurs propres d'une matrice. Tapez `?` de cette fonction.

2.6. **Historique.** R conserve l'historique des commandes. Il est donc possible de récupérer des instructions déjà saisies (et ensuite de les modifier dans le but de les réutiliser) :

$\uparrow$, $\downarrow$, $\rightarrow$, $\leftarrow$	permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes

2.7. **Variables d'environnement.** R garde en mémoire les variables qui ont été créées. On les voit en haut, à droite, dans la fenêtre Workspace. Sinon, on peut utiliser les lignes de commandes suivantes.
`objects()` donne la liste des objets présents dans l'espace de travail

`rm(x,y)`	efface les variables x, y de l'espace de travail
`rm(list=objects())`	efface toutes les variables crées dans l'espace de travail

Exemple 1.

(1) Tapez la commande `a=1:7`. Tapez les commandes `a`, `objects()`.

(2) Utilisez $\uparrow$ pour modifier a : `a=1:2;`.

(3) Tapez la commande `b=a+2;`. Réexécutez la commandes `objects()` en utilisant $\uparrow$. Tapez `rm(list=objects())` et `b`.

`print(var)`	affiche le contenu de *var*
`x=as.numeric(readline(prompt="entrez la valeur de x: "))`	affiche la chaîne de caractères `entrez la valeur de x:` et donne la main à l'utilisateur pour qu'il entre la valeur de la variable `x`

2.8. **Dialogue avec l'utilisateur.**

Exemple 2. Essayez les commandes suivantes :
```
x=as.numeric(readline(prompt="entrez le nb de simulation: "))
print(n)
```

3. Les types de données

R est orienté objet. Un objet essentiel est l'objet matrice.

3.1. **Construction explicite.** On peut former des vecteurs et des matrices en entrant leurs coefficients.

- **scalaires**
  > `s=30`

- **vecteurs numériques**
  > `x=c(1,2,3)` (les `,` séparent les éléments d'un vecteur colonne)
  > `t(x)` transposée de x
  > `y = c(x,x,x)` concatène 3 fois x

- **matrices**
  > `M=matrix(c(11,12,13,14,21,22,23,24,31,32,33,34),ncol=3)`
  > `M=matrix(c(11,12,13,14,21,22,23,24,31,32,33,34),nrow=3)`
  Par défaut, R ordonne les éléments d'une matrice par colonne, l'argument `byrow=TRUE` permet de le faire par ligne. > `M=matrix(c(11,12,13,14,21,22,23,24,31,32,33,34),nrow=3,byrow=TRU`

  Construction à partir de plusieurs vecteurs de même longueur :
  > `x=c(1,2,3); y=c(2,3,4)`
  > `mat1=cbind(x,y)` concaténation en colonne
  > `mat1=rbind(x,y)` concaténation en ligne

- **vecteurs de chaîne de caractères**
  Pour créer une chaîne de caractères, on entre les caractères en commençant et en terminant par `'` (*quote*).
  > `ch='test'`

- **les nombres complexes**
  Dans r, un nombre complexe est de la forme : $z = a + ib$.
  > `x=c(1:3); y=c(2:4); z <- complex(real = x, imaginary = y)`

- **Les list**

  Les listes sont des objets très génériques. Elles sont utiles car elles permettent de stocker des objets hétérogènes. Ainsi, une liste de 3 éléments peut être composée d'un vecteur, d'une matrice et d'une chaîne de caractère.

La création d'une liste se fait via la fonction *list*() :

```
x<-c(1:10)
 M<-diag(1:3)
l<-list("a",x,M)
```

On appelle ensuite les éléments d'une liste à l'aide des doubles crochets pour appeler 1 seul élément de la liste.

```
> l[[1]]
[1] "a"
> l[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10
> l[[3]]
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

Il peut être utile de nommer les éléments d'une liste pour ensuite pouvoir les appeler par la commande $:

```
> names(l)<-c("Roger","Victor","Francis")
> l$Roger
[1] "a"
> l$Victor
 [1]  1  2  3  4  5  6  7  8  9 10
> l$Francis
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

On peut appeler plusieurs éléments à d'une liste à la fois, via la commande :

```
> l[1:2]
[[1]]
[1] "a"
[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

Ce procédé est utiliser dans la plupart des packages et de nombreuses fonctions renvoient des listes dont les éléments sont nommés.

Exercice 2. Entrez les différents vecteurs et matrices et donnez la longueur et la taille de chacun (Utilisez `help` pour trouver les fonctions qui donnent longueur et taille).

3.2. **Création rapide.** Certaines commandes permettent de créer plus rapidement des vecteurs précis :
> `l1=1:10` Un vecteur contenant les entiers de 1 à 10
> `l2=rep(1,10)` Un vecteur contenant une répétition de 1, 10 fois
> `matrix(1,nrow=4,ncol=5)` matrice 4 lignes, 5 colonnes de 1

A noter, **une ligne de commande commençant par le caractère # n'est pas exécutée par R.** Cela permet d'insérer des lignes de commentaires. Et il faut commenter ses programmes... surtout ceux de l'examen.

Exercice 3. Construire :

(1) une suite partant de -8 et allant à -5 par pas de 0.25.

(2) une suite décroissante d'entiers de 15 à 3.

(3) une suite de longueur 100 de $-\pi$ à π.

3.3. **Opérations vectorielles.** Les tableaux suivants résument certaines commandes couramment utilisées.

Vecteurs

`n:m`	nombres de `n` à `m` par pas de 1
`seq(n,m,by=p)`	nombres de `n` à `m` par pas de `p`
`seq(n,m,length=p)`	nombres de `n` à `m` de taille `p`
`length(x)`	longueur de `x`
`x[i]`	i-ème coordonnée de `x`
`x[i1:i2]`	coordonnées `i1` à `i2` de `x`
`x[-(i1:i2)]`	supprimer les coordonnées `i1` à `i2` de `x`
`cbind(x,y)`	concaténer les vecteurs `x` et `y` en colonne
`rbind(x,y)`	concaténer les vecteurs `x` et `y` en ligne

Matrices

`dim(A)`	nombre de lignes et de colonnes de `A`
`A[i,j]`	coefficient d'ordre `i,j` de `A`
`A[i1:i2,]`	lignes `i1` à `i2` de `A`
`A[-(i1:i2),]`	supprimer les lignes `i1` à `i2` de `A`
`A[,j1:j2]`	colonnes `j1` à `j2` de `A`
`diag(A)`	coefficients diagonaux de `A`

Matrices particulières

`matrix(0,m,n)`	matrice nulle de taille `m,n`
`matrix(1,m,n)`	matrice de taille `m,n` dont tous les coefficients valent 1
`diag(n)`	matrice identité de taille `n`
`diag(x)`	matrice diagonale dont la diagonale est le vecteur `x`

Exemple 3. *Extraction de composantes*
Entrez la matrice
> `A=rbind(c(1,2,3),c(2,3,1),c(3,1,2))`
Quels sont les résultats des commandes suivantes ?
> `A[2:3,1:3]`
> `A[2:3,1:2]`
> `A[2:3,]`
> `A[,1]`

Exercice 4. Répliquez le vecteur `c(1,3,6)` pour en faire une matrice 3×19, de deux manières : en utilisant `matrix(1,m,n)` et en effectuant une multiplication matricielle, puis en trouvant la commande *ad hoc* de réplication.

Exercice 5. Ecrire la matrice carrée M d'ordre 12 contenant les entiers de 1 à 144 rangés par ligne. Extraire de cette matrice les matrices suivantes :
- la sous-matrice formée par les coefficients a_{ij} pour $i = 1, ..., 6$ et $j = 7, ..., 12$;
- celles des coefficients a_{ij} pour $(i, j) \in \{1, 2, 5, 6, 9, 10\}^2$;
- celle des coefficients a_{ij} pour $i + j$ pair.

4. Les opérations matricielles et les fonctions

4.1. **Les opérations matricielles.**

`t(A)`	transposée de `A`
`rankMatrix(A)`	rang de `A (package Matrix)`
`solve(A)`	inverse de `A`
`det(A)`	déterminant de `A`
`sum(diag(A))`	trace de `A`
`eigen(A)`	valeurs propres et vecteurs propres de `A`
`+ -`	addition, soustraction
`* ^`	multiplication, puissance terme à terme
`%*%`	multiplication matricielle
`solve(A,b)`	solution de $Ax = b$
`/`	division terme à terme
`apply(X, MARGIN, FUN, ...)`	applique la fonction FUN aux colonnes ou lignes de X

Exercice 6. Reprendre la matrice A de l'exemple 3. A l'aide d'une boucle for, calculer la moyenne et l'écart type de A de chaque colonne de A. Faire le même calcul à l'aide de la fonction apply.

4.2. **Les fonctions.**

Fonctions élémentaires

`sqrt`	`exp`	`log`
`sin`	`cos`	`tan`
`asin`	`acos`	`atan`
`round`	`floor`	`ceiling`
`abs`	`Arg`	`Conj`

Exemple 4. Construire un vecteur quelconque et essayer les fonctions ci-dessus.

Certaines fonctions de R s'appliquent à l'ensemble d'un vecteur.

Le tableau suivant décrit le résultat de quelques unes de ces fonctions lorsqu'elles sont appliquées à un vecteur x :

Fonctions vectorielles

Exemple 5. Regardez l'effet des instructions suivantes.

> `x=runif(5)`
> `mean(x)`
> `sd(x)`
> `median(x)`
> `sort(x)`
> A=runif(5)
> sort(A)
> order(A)
> max(A)
> sum(A)
> cumsum(A)
> prod(A)
> diff(A)

Pour appliquer des fonctions aux colonnes ou lignes de matrice, il faut EVITER LES BOUCLES et privilégier la fonction `apply` qui permet d'appliquer une fonction `f` aux lignes (MARGIN=1) ou colonnes (MARGIN=2) d'une matrice.

`max(x)`	maximum
`min(x)`	minimum
`sort(x)`	tri par ordre croissant
`o = order(x)`	retourne les permutations des élements de `x pour le tri par ordre croissant (par défaut) ou décroissant`
`which(x==0)`	retourne les indices non nuls de `x`
`sum(x)`	somme des éléments de `x`
`cumsum(x)`	vecteur contenant la somme cumulée des éléments de `x`
`prod(x)`	produit des éléments de `x`
`cumprod(x)`	vecteur contenant le produit cumulé des éléments de `x`
`diff(x)`	vecteur des différences entre deux éléments consécutifs de `x`
`mean(x)`	moyenne des éléments de `x`
`var(x)`	variance
`sd(x)`	écart type (voir aussi `std(x,1)`)

> X=cbind(c(1 :3),c(4 :6),c(10 :12))
> apply(X,1,mean) # moyenne par ligne
> apply(X,1,mean) # moyenne par colonne

Exercice 7. Soit X une matrice $2 \times n$ contenant les coordonnées de n points du plan. Comment faire pour obtenir une matrice où les points sont ordonnés par ordre croissant des abscisses ?

Exercice 8.

(1) Soit le vecteur de dimension 8 de composantes : 3.2, 4.8, 3.3, 3.2, 3.1, 4.2, 3.2, 3.3. Entrez le vecteur $\mathbf{y} = (y_i)_{i=1,\dots,8}$ correspondant.

(2) Construisez à l'aide des fonctions précédentes la suite des moyennes

$$\bar{y}_n = \frac{1}{n} \sum_{i=1}^{n} y_i \ .$$

Extrayez $\bar{y}_8$. Donnez une fonction qui calcule directement $\bar{y}_8$ à partir de $\mathbf{y}$.

Exercice 9. Tirez 20 nombres aléatoirement dans l'intervalle $[0, 1]$. Quelle est la valeur minimale du vecteur et la position du coefficient qui la réalise ? Vérifiez.

5. Opérateurs relationnels et logiques

Ils permettent de relier logiquement deux *matrices*.

`Opérateurs relationnels`	`<`, `<=`, `>=`, `==` (égalité), `!=` (différent)
`Opérateurs logiques`	& (et), | (ou)

Attention de ne pas confondre `=` qui sert à affecter une valeur à une variable et `==` qui sert à tester l'égalité. Les opérateurs relationnels peuvent être utilisés avec des scalaires, des vecteurs ou des matrices. Le résultat d'évaluation d'une expression relationnelle est `1` (vrai) ou `0` (faux). Appliqués à une matrices, ils rendent une matrice de même dimension, formée de 1 et de 0.

Exemple 6.
`u=4`
`u==4`
`u<=12`
Reprenez la matrice `A=rbind(c(1,2,3),c(2,3,1),c(3,1,2))`.
`Ar=(A<=2)`

Exercice 10. Reprenons le vecteur **y**. Pour rappel, tapez `y`.

(1) Faites répondre R à la question suivante : existe-t-il une coordonnée du vecteur **y** inférieure à 3.3 ?

(2) Construisez un vecteur logique `z` tel que la i-ème coordonnée de ce vecteur sera 1 si la $i$ème coordonnée du vecteur y est à l'extérieur de l'intervalle $[\bar{y}_8 - \sigma_8, \bar{y}_8 + \sigma_8]$ où σ_8 est la racine carrée de la variance d'échantillonnage.

Exercice 11.

(1) Tirez 100 nombres aléatoirement (et uniformément) dans l'intervalle $[0, 1]$ et groupez-les dans un vecteur $\mathbf{x} = (x_i)_{i=1,\ldots,100}$.

(2) Prenez $y_i = 2 * x_i$ pour tout $i = 1 \ldots, 100$.

(3) Prenez la partie entière de ces nombres (à l'aide de la fonction `floor`) : $z_i = [y_i]$. Ceci définit un vecteur **z**. (Au passage, notez qu'il existe plusieurs fonctions parties entières, avec des comportements différents, `ceiling, floor` par exemple...)

(4) Donnez la fréquence de 1 sur l'échantillon **z**. Pouvait-on s'attendre à ce résultat ?

Réinitialisez l'espace de travail en tapant `rm(list=objects())`.

6. Graphiques

6.1. **Représentations de points dans le plan.** Il existe plusieurs possibilités pour représenter un ensemble de points $(x(i), y(i))$. Les plus utilisées sont énumérées ci-dessous.

`plot(x,y)`	tracé d'un nuage de points
`plot(x,y,type='l')`	tracé d'une courbe
`barplot(x)`	tracé sous forme de diagramme en barres
`plot(x,y,type='h')`	diagramme en bâtons
`hist`	trace des histogrammes
`points(x,y); lines(x,y)`	superpose un nuage de point ou une courbe au graphique précédemment tracé avec plot

Chacune de ces fonctions présente un grand nombre d'options, consultez l'aide ?plot pour avoir des détails ainsi que ?par pour la liste des différents paramètres graphiques.

6.2. **Gestion de la fenêtre graphique.** Les paramètres des fenêtres graphiques sont gérés dans R via la fonction `par`.

`par(new=TRUE)`	les prochains tracés se superposeront aux tracés déjà effectués
`dev.off()`	fermeture de la fenêtre graphique active
`par(mfrow=c(n,m)`	partage la fenêtre graphique active en une fenêtre de n lignes et m colonnes
`postscript("file.txt"); pdf("file.txt")`	écris les commandes graphiques qui suivent dans un fichier postcript ou pdf, ne pas oublier de terminer ces instruction par `dev.off()`

6.3. **Axes et légendes.**

`plot(...,xlim=c(-10,10), ylim=c(-5,5))`	pour définir les échelles des axes
`plot(...,main="titre")`	titre pour le graphique
`plot(...,xlab='x',ylab='y')`	légende pour l'axe des abscisses et des ordonnées
`legend("top",col=c("blue","red"), c("nom1","nom2"))`	légende pour chaque courbe du graphique

Exemple 7.

```
x=seq(-5,5,length=100)
y=x^2
z=1/x

plot(x,y,type='l')
lines(x,z,col='red')
legend("top",col=c("black","red"),c("x^2","1/x"),lty='solid')

plot(x,y,type='l')
par(new=TRUE)
plot(x,z,col='red',type='l')
```

Exercice 12. Soit f et g les fonctions définies sur l'intervalle $[0, 10]$ par :

$$f(x) = \exp\left(-\frac{x}{25}\right), \quad g(x) = \cos\left(\frac{x}{10}\right) .$$

Tracez ces deux fonctions (à l'aide de `plot`) d'abord dans une même fenêtre graphique mais sur des graphes différents, puis dans une même fenêtre graphique et sur le même graphe.

6.4. **La sauvegarde d'une figure.** Pour sauvegarder une figure, dans RStudio, dans la fenêtre graphique utiliser le menu déroulant Export/Save Plot as Image...Vous pouvez également utiliser les fonctions `postscript()` et `pdf` décrites précédemment.

Exercice 13.

(1) Tirer 100 couples de points (x, y) aléatoirement dans le carré $[0, 1] \times [0, 1]$.

(2) Représenter le nuage de points obtenus dans une fenêtre graphique.

(3) Calculer le centre de gravité G du nuage de points.

(4) Ajouter en rouge au nuage de points le centre de gravité.
(5) Sauvegarder la figure sous le nom `nuage.pdf`.
(6) Faire afficher dans une même fenêtre graphique deux histogrammes, un pour les abscisses et l'autre pour les ordonnées des points tirés.

7. Utilisation de fichiers.

7.1. **Les fichiers de sauvegarde.** Avant d'aborder les commandes de sauvegarde, un point important est qu'il est essentiel lorsque vous codez en R d'effectuer régulièrement des sauvegarde des instructions que vous tapez via l'éditeur de commande.

Voici d'autres façons de sauvegarder son travail.

`save(objet, file="fichier.RData")`	sauve la variable (matrice, vecteur, liste...) dans le fichier spécifié
`save.image("fichier.RData")`	sauve l'ensemble des variables existant dans l'espace de travail de R dans un fichier binaire
`write.table(M,"fichier.txt")`	sauve la table M dans un fichier texte

Exemple 8. Créez une liste incluant `x`, `y` et le centre de gravité `G` et sauvegardez la dans un fichier binaire. Dans R, avec la commande `rm()`, effacez `x`, `y` et `G`. Puis, chargez le fichier sauvegardé afin de les récupérer. Vérifiez la récupération.

7.2. **Les fonctions.** Une fonction est un objet R qui prend en entrée des arguments et renvoie un résultat en sortie. Les arguments d'une fonction sont soit obligatoires soit optionnels (dans ce cas ils ont une valeur par défaut).

Exemple 9.
densite de la loi $N(\mu, \sigma)$:

```
densnorm=function(x,mu=0,sigma=1)
{
  d=(1/(sigma*sqrt(2*pi)))*exp(-(x-mu)^2/(2*sigma^2))
  return(d)
}
```

Si le fichier contenant la fonction est sourcé en utilisant la commande `source("fichier.r")` ou si la fonction est simplement exécuter dans la console (run dans RStudio) la fonction `densnorm` devient une fonction comme toutes les autres.

Exercice 14. Utilisez la fonction ci-dessus pour tracer (avec `plot`) la densité de la loi normale centrée réduite entre -5 et 5 en faisant calculer 100 points. Calculez $\mathbb{P}(-5 \leq X \leq 1.96)$ où X est une variable aléatoire de loi $\mathcal{N}(0, 1)$.

Exercice 15. Ecrire une fonction `simule` qui prend `n` en argument et :

(1) construit un vecteur de n nombres aléatoires $(y_i)_{i=1,\dots,n}$ issus d'une loi normale de moyenne 2 et de variance 1,
(2) renvoie la moyenne et la variance de l'échantillon observé,
(3) sépare la fenêtre graphique en deux espaces graphiques et superpose respectivement sur chacun d'eux :
   – l'histogramme de $(y_i)_{i=1,\dots,n}$ (on pourra utiliser la fonction `hist`) et la densité de la loi normale de moyenne 2 et de variance 1, donne un titre à la figure et affiche la moyenne et la variance,
   – la fonction de répartition empirique (utiliser la fonctions `sort`) et la fonction de répartition théorique (utiliser la fonction `pnorm` et donne un titre à la figure.

Faites tourner le programme pour d'autres valeurs de n, plus grandes ou plus petites.
Indication : la première ligne du fichier de fonction est `simule=function(n)`.

7.3. **Les programmes *script* (ou fichiers d'instructions).** Ce sont des fichiers texte avec une extension `.r`. Ils contiennent des suites d'instructions R qui sont exécutées les unes après les autres. On peut créer un fichier en cliquant sur `File` puis `Save as`, `Script.r`.

Exemple 10. Sauvez dans le répertoire courant les lignes suivantes sous le nom `losange.r` :

```
losange=function()  x=rbind(c(0,-1,0,1), c(-1,0,1,0))
y=rbind(c(-1,0,1,0),c(0,1,0,-1))
plot(x,y,type='l')
```

La commande `losange` affichera `x`, puis `y`, puis tracera un losange. Si ce fichier est "sourcé" en utilisant la commande `source("losange.r")`.

8. Les commandes structurées

8.1. **L'instruction `for`.** Bien que peu recommandé -il faut privilégier les opérations vectorielles à l'utilisation de boucles-, les boucles for existent en R et peuvent être utilisées si nécessaire. La syntaxe en est la suivante :

```
for( i in x)
{
faire ceci ou cela
}
```

Exemple 11. Ces quelques lignes calculent $n!$ pour $n = 1$ à 100.

```
n=100
fact=array(0,dim=n)
fact[1]=1
nfact=1
for(i in c(2:n))
{
  nfact=nfact*i
  fact[i]=nfact
}
```

Mais on peut aussi taper simplement : `n=100; fact=cumprod(1:n);` (ou même `gamma(n+1)`) L'exécution du premier programme prend significativement plus de temps que l'exécution du deuxième programme. Pour le voir, et comparer l'efficacité des algorithmes, utilisez `proc.time()`.

```
ptm <- proc.time()
gamma(n+1)
proc.time() - ptm
```

8.2. **Les intructions conditionnelles.** La manière la plus brute de procéder est d'utiliser un bloc `if .. ... else ...` . La syntaxe en est la suivante :

```
if(x==0)
{
  faire ceci
}
else
{
  faire cela
}
```

'faire ceci' n'est exécuté ici que si $x = 0$, sinon 'faire cela' est exécuté. Bien entendu, il s'agit ici d'un exemple et la condition 'x==0' peut être remplacée par n'importe quel booléen.

Il est possible d'emboiter des conditions les une dans les autres.

```
x=runif(20)
y=array(0,dim=20)
for(i in c(1:length(x)))
{
  if(x[i]<0.5)
    y[i]=1
  else
    if(x[i]>0.8)
    y[i]=1
}
```

Exemple 12. Le programme suivant simule le lancer d'une pièce 10 fois de suite.

```
p=0.5
u=runif(10)
for(i in c(1:10))
{
  if(u[i]<p)
  {
    print('pile')
  }
  else
  {
    print('face')
  }
}
```

Rappelons qu'une proposition logique (par exemple, `(x>0 & x<10)`) a pour valeur 1 si elle est vraie et 0 sinon. Dans les calculs, on peut donc éviter d'utiliser l'instruction `if` en introduisant des indicatrices d'ensembles.

Exercice 16. En tenant compte de cette remarque, simuler la réalisation d'une variable aléatoire de loi de Bernoulli de paramètre p. Puis, après cet échauffement, écrire un programme *script* `EstimeParam.r` qui :

(1) simule un 1000-échantillon de variables aléatoires de Bernoulli de paramètre 0.2,

(2) affiche le nombre de succès contenus dans ce 1000-échantillon,

(3) et trace la courbe des moyennes en fonction de la longueur de l'échantillon.

Exécuter le programme. Que remarque-t-on ? Quel est le théorème qui explique le phénomène ?

Exercice 17. Nous allons censurer des variables. Soit `u` une réalisation d'un 15-échantillon $X_1, \ldots, X_{15}$ de loi uniforme sur $[-5, 1]$. Mettez les observations plus petites que -2 à -10. Ceci forme `v`, réalisation d'un 15-échantillon $Y_1, \ldots, Y_{15}$. Quelle est la loi commune aux Y_i ?

8.3. **L'instruction `while`.** Ce format de boucle permet de s'arrêter conditionnellement (et non plus à rang fixé, comme dans une boucle `for`). La syntaxe en est la suivante :

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

Les instructions sont exécutées tant que les conditions sont vérifiées.

Exercice 18. Ecrire une fonction `Puiss(n,M)` qui étant donné un entier n et une valeur maximale $M > n$, calcule les puissances entières de ce nombre, les agrège dans un vecteur `y` et s'arrête lorsque le résultat dépasse la valeur maximale.