Programmation et Logiciels Statistiques
Introduction à R

Salim Lardjane
Université de Bretagne-Sud
L3 MIS Stat & LP SIS

▶ Lancer R

Aide

`help.start()` : Lance l'aide au format html
`help(fonction ou commande)` ou `?fonction ou commande` : fournit de l'aide sur l'utilisation de la fonction ou de la commande à laquelle on s'intéresse
`help.search("mot-clé")` : fournit une liste des fonctions et commandes pouvant être associées au mot-clé spécifié
`apropos("chaîne de caractères")` : fournit une liste des fonctions et commandes dont le nom contient la chaîne de caractères spécifiée

---

```
» help.start()
» help(integer)
» help.search("integer")
» apropos("help")
```

On peut également utiliser le menu *Aide*.

Quitter

```
» q()
```

On peut également utiliser *Sortir* dans le menu *Fichier*.

Répertoire de travail (répertoire courant de R)

```
» getwd()
» setwd("d:/Work/cours/Tutorial")
» getwd()
```

La liste de commandes R passées peut être obtenue à l'aide de la commande `history()`.

» `history()`

En mode interactif, R peut être utilisé comme une calculatrice scientifique.
On dispose des opérations arithmétiques et d'un ensemble de fonctions de calcul numérique et de visualisation graphique.

Constantes prédéfinies

`pi` : 3.141593
`Inf (Infinite)` : nombre infini
`NaN (Not a Number)` : n'est pas un nombre, exprime une indétermination
`NA(Not Available)` : Non disponible, exprime une valeur manquante.

```
» pi
» 1/0
» 0/0
```

Des commandes peuvent être saisies sur la même ligne à condition d'être séparées par un point-virgule.

» `pi ; 1/0 ; 0/0`

Opérations arithmétiques

+, -, *, /, ^, %%, %/%
/ : division usuelle
^ : élévation à une puissance
%% : modulo
%/% : division entière

```
» 5+7-3+2*5
» 4/3
» 4 %/% 3
» 3^4
» 5 %% 3
```

Les opérations peuvent être composées en respectant les priorités uselles et en utilisant des parenthèses.

» `4*(-5)+12`
» `2.3*(4-6)/(3+15)`

L'élévation à une puissance est prioritaire par rapport aux autres opérations.
Comparer

» `9^1/2` et
» `9^(1/2)`

Saisie

↑	passer à l'instruction précédente
↓	passer à l'instruction suivante
←	aller vers la gauche
→	aller vers la droite
BackSpace	effacer le caractère à gauche
Suppr	effacer le caractère à droite
Début, Fin	se positionner au début, à la fin

Saisir

» `4*(4.5-23+2.5)/(8.5-3.2)`

puis remplacer 23 par 2.3

Vecteurs numériques

Saisie d'un vecteur x

```
» x<-c(6, 4, 7)
» x
```

ou

```
» assign("y",c(6, 4, 7))
» get("y")
» y
```

On peut saisir successivement au clavier les éléments d'un vecteur à l'aide de la commande `scan()`.

```
» x<-scan()
» x
```

La longueur d'un vecteur est sa dimension.

```
» length(x)
```

Lors de la saisie à la console d'un vecteur, on peut spécifier directement le nombre de composantes.

```
» x<-scan(n=3)
» x
```

R fait la distinction entre majuscules et minuscules.

```
» x
» X
```

Construction d'un vecteur à partir d'un autre

```
» v <- c(x,1,2,8)
» w <- c(1,v)
```

```
» w <- c(1,x,1,2,8)
```

Récupération d'une composante d'un vecteur

Dans R, les indices des coordonnées d'un vecteur commencent à 1.

```
» w
» w[3]
» w[0]
```

Récupération de plusieurs composantes d'un vecteur

```
» w<-c(4,2,5)
» w
» w[c(1,3)]
» w[c(3,1)]
» w[0:2]
» w[c(1,4)]
» w[c(0:2,NA)]
```

Récupération des composantes d'un vecteur sauf certaines

```
» w<-c(4,2,5)
» w
» w[-3]
» w[c(-1,-2)]
```

Ajout d'un composante

```
» w<-c(w,7)
» w

» w<-append(w,7)
» w

» w<-append(w,8,after=2)
» w
```

Remplacement de composantes

```
» w[2]<-3
» w

» w<-replace(w,1,7)
» w
» w<-replace(w,c(2,4),c(8,9))
» w
» w[-c(1,4)]<-0
» w
```

Si l'on souhaite définir un vecteur de valeurs comprises entre deux valeurs fixées et espacées d'un pas constant, on peut procéder de l'une des façons suivantes

```
» debut<-0
» fin<-1
» pas<-0.25
» t<-seq(debut,fin,pas)
» t

» t<-seq(-0.5,1,0.25)
» t

» t<-seq(from=-0.5,to=1,by=0.25)
» t
```

Par défaut, le pas est de 1 et la première valeur vaut 1.

```
» seq(10)
» seq(-0.5,10)
```

Cas particulier : la commande `:` permet de générer un vecteur de nombres compris entre la première et la dernière valeur pour un pas de 1.

```
» t<-0:10
» t
» t<-0.5:10.8
» t
```

La longueur du vecteur obtenu peut être obtenue à l'aide de la fonction `length`.

```
» length(seq(0,1,.1))
```

On peut également spécifier directement le nombre d'éléments à obtenir au lieu du pas.

```
» t<-seq(from=-0.5,to=1,length=9)
» t
```

Afin de répéter une même valeur ou un même vecteur un nombre n de fois, on peut utiliser la fonction `rep`.

```
» rep(2,10)
» rep(c(4,2),3)
```

Afin d'inverser l'ordre d'un vecteur, on peut utiliser la fonction `rev`.

```
» rev(1:5)
```

Afin d'extraire des composantes sans répétition :

```
» x <- c(rep(1,3),seq(1,5,by=2),rev(seq(1,5,length=3)),rep(2,3))
» x
» unique(x)
```

Afin d'ordonner les composantes par ordre croissant ou décroissant :

```
» sort(x)
» rev(sort(x))
```

Afin d'obtenir les rangs des composantes (en cas d'ex aequos, le rang moyen est utilisé) :

```
» rank(x)
```

Afin d'obtenir les indices des composantes ordonnées par ordre croissant :

```
» order(x)
```

`sort(x)` est équivalent à `x[order(x)]`.

Opérations arithmétiques sur les vecteurs

Les opérateurs +,-,*, /, ^, %%, %/% permettent de réaliser des opérations sur les vecteurs élément par élément. On dit que R est *vectorisé*.

```
» x<-c(0,6,3)
» y<-c(2,5,7)
» x+y
» x-y
» 3+x
» x-2
» 2*x
» x/4
» x*y
» x^2
» x/y
» x%%y
» x%/%y
```

Fonctions

Exemple explicite de fonction

```
» f<-function(x,y)
{
return(x+y-2)
}
» x
» y
» z<-f(x,y)
» z
```

La définition d'une fonction peut faire intervenir des variables locales.

```
» f<-function(x,y)
{
ans<-x+y-2
return(ans)
}
» f(x,y)
```

Les fonctions mathématiques usuelles sont prédéfinies dans R : *abs, sqrt, sin, cos, exp, tan, asin, acos, atan, log, log10, log2, sinh, cosh, tanh, asinh, acosh, atanh...* ainsi que les fonction spéciales *gamma, digamma, beta, besselI,...*etc. R étant vectorisé, toutes ces fonctions peuvent prendre un vecteur comme argument. Les fonctions sont alors appliquées élément par élément.

```
» x
» y
» z<- -y
» sqrt(x)
» abs(z)
```

Fonctions numériques usuelles

`ceiling(x)`	plus petit entier supérieur à `x` (composante par composante)
`floor(x)`	plus grand entier inférieur - - -
`trunc(x)`	composante entière de `x` - - -
`round(x)`	arrondi à la plus proche valeur entière (valeurs .5 arrondies au plus proche entier pair) - - -
`round(x,n)`	arrondi à `n` chiffres après la virgule - - -

```
» x<- c(-1.9069,0.76018,-0.26556,-1.89828,0.08571,NA)
» ceiling(x)
» floor(x)
» trunc(x)
» round(x)
» round(x,2)
```

Fonctions statistiques usuelles

`sum(x)`	somme des composantes d'un vecteur `x`
`prod(x)`	produit des - - -
`mean(x)`	moyenne des - - -
`mean(x,trim=p)`	moyenne trimmée - - - (p est compris entre 0 et 0.5)
`median(x)`	médiane des - - -
`quantile(x,probs=p)`	quantiles des - - - de niveaux spécifiés dans le vecteur p
`var(x)`	variance des - - -
`sd(x)`	écart-type des - - -
`min(x)`	minimum des - - -
`max(x)`	maximum des - - -
`cumsum(x)`	sommes cumulées des - - -
`cumprod(x)`	produits cumulés des - - -
`rank(x)`	rangs des - - -

```
» x<-scan()
» sum(x)
» prod(x)
» mean(x)
» mean(x,trim=0.2)
» median(x)
» quantile(x,probs=c(0,0.1,0.9)))
» var(x)
» sd(x)
» min(x)
» max(x)
» cumsum(x)
» cumprod(x)
» rank(x)
```

Définir une fonction permettant de centrer et réduire un vecteur donné.

```
» cr<-function(x)
{
ans=(x-mean(x))/sqrt(var(x))
return(ans)
}

» v<-1:10
» cr(v)
```

Autres fonctions statistiques

`which.max(x)`	indice du maximum des composantes de `x`
`which.min(x)`	indice du minimum des - - -
`range(x)`	identique à `c(min(x),max(x))`
`var(x,y)` ou `cov(x,y)`	covariance entre `x` et `y`
`cor(x,y)`	corrélation linéaire entre - - -
`cummin(x)`	minimums cumulés des composantes de `x`
`cummax(x)`	maximums cumulés des - - -
`pmin(x,y,...)`	vecteur dont la i-ème composante est min(x[i],y[i],...)
`pmax(x,y,...)`	vecteur dont la i-ème composante est max(x[i],y[i],...)
`diff(x)`	vecteur dont la i-ème composante est x[i+1]-x[i]
`diff(x,lag=n)`	vecteur dont la i-ème composante est x[i+n]-x[i]

Arguments par défaut

Il est souvent souhaitable de spécifier des valeurs par défaut pour les arguments d'une fonction. Cela peut être fait lors de la déclaration de la fonction. Par exemple, la première ligne de la fonction `mean` est

```
mean <- function(x,trim=0,na.rm=FALSE)
```

Les arguments pour lesquels une valeur par défaut est spécifiée peuvent être omis lors de l'appel de la fonction. Par exemple :

```
» x<-1:10
» mean(x)
```

---

permet d'obtenir la moyenne standard de `x`. Si l'on souhaite effectuer un trimming de 20%, il suffit de spécifier la valeur correspondante pour l'argument `trim` :

```
» mean(x,trim=0.2)
```

Il en va de même pour l'argument `na.rm`, que nous discuterons plus loin.

Exemple détaillé

```
» fk<-function(x,y,k=2)
{
return(x+y-k)
}

» x<-1:3
» y<-5:7
» fk(x,y)
» fk(x,y,k=3)
```

Matrices numériques

Matrice nulle

» `matrix(0,nrow=2,ncol=3)`
ou
» `mat.or.vec(2,3)`

Matrice unité

» `matrix(1,nrow=3,ncol=2)`

Matrice identité (matrice carrée)

```
» x<-matrix(0,nrow=2,ncol=2)
» diag(x)<-1
» x
```

ou

» `diag(1,nrow=2)`

ou encore

» `diag(rep(1,2))`

Saisie d'une matrice quelconque

Eléments lus colonne par colonne

```
» x<-matrix(c(0,1,2,3,4,5),nrow=2,ncol=3)
» x
```

Eléments lus ligne par ligne

» `x<-matrix(c(0,1,2,3,4,5),nrow=2,ncol=3,byrow=TRUE)`

```
» x
```

Par redimensionnement ou répétition ("recyclage") des éléments d'un vecteur ou d'un scalaire

---

```
» y<-matrix(c(0,5),nrow=2,ncol=3)
» y
» z<-matrix(5,nrow=2,ncol=3)
» z
```

Par concaténation de lignes

```
» rbind(c(0,2,4),c(1,3,5))
```

Par concaténation de colonnes

```
» cbind(c(0,1),c(2,3),c(4,5))
```

Saisie directe au clavier

```
» x <-matrix(scan(n=2*3),nrow=2,ncol=3,byrow=TRUE)
» x
```

Saisie d'une matrice diagonale

```
» u <- c(10,20)
» v <- diag(u)
» v
```

Saisie d'un vecteur colonne

```
» y<-matrix(c(6,4,7),nrow=3,ncol=1)
```
ou
```
» as.matrix(c(6,4,7))
```

Dimensions d'une matrice

```
» d<-dim(y)
» d
» d[1]
» d[2]
```

Transposée d'une matrice

```
» x
» tx<-t(x)
» tx
```

Accès à un élément d'une matrice

```
» x[2,1]
```

Extraction d'une partie d'une matrice, extension d'une matrice

```
» x<-rbind(c(0,1,2),c(3,4,5))
» x
» x[2,3]

» x1 <- x[,2:3]
» x1

» y <- rbind(c(0,1,2),c(3,4,5),c(6,7,8))
» y

» y1<-y[1:2,c(1,3)]
» y1

» y2<-y[1:2,]
» y2

» y3<-cbind(y[,1],y[,3])
» y3
```

Extraction de la deuxième colonne

```
» x[,2]
```

Extraction de la deuxième ligne

```
» x[2,]
```

Dans les deux cas précédents, le résultat obtenu est un vecteur. Il est posible de spécifier que le résultat doit être une matrice à l'aide des commandes suivantes.

```
» x[2,,drop=FALSE]
» x[,2,drop=FALSE]
```

Extraction de la diagonale d'une matrice

```
» x
» diag(x)
» x<-matrix(1:4,nrow=2)
» x
» diag(x)
```

Opérations arithmétiques sur les matrices

Les opérateurs +,-,*, /, ^, %%, %/% permettent de réaliser des opérations sur les matrices élément par élément. La multiplication matricielle est effectuée par %*%.

```
» x<-rbind(c(1,2,3),c(0,3,1),c(-2,-1,4))
» y<-rbind(c(2,-1,-3), c(1,-1,3), c(4,2,3))
» x
» y
» x+y
» x-y
» x/y
» x*y
» x%%y
» x%/%y
» x%*%x
» x%*%y
» x^2
```

Le produit d'un vecteur ligne par sa transposée donne le carré de la norme de celui-ci.

```
» x<-as.matrix(c(6,4,7))
» x
» tx<-t(x)
» tx
» tx %*% x
```

Le produit d'un vecteur colonne de taille n par un vecteur ligne de taille m donne une matrice de dimensions $n \times m$.

```
» y<-matrix(c(1,2,3),nrow=1,ncol=3)
» x%*%y
```

Le même résultat peut être obtenu à l'aide de la fonction `outer` appliquée aux *vecteurs* .

```
» x<-c(6,4,7)
» y<-c(1,2,3)
» outer(x,y,"*")
```

La fonction `outer` appliquée aux *vecteurs* permet de façon plus générale de calculer la matrice *f(x[i],y[j])* où f est une fonction quelconque de deux vecteurs.

```
» x<-c(6,4,7)
» y<-c(1,2,3)
» outer(x,y,"/")
```

Déterminant d'une matrice

```
» x<-rbind(c(1,2,3),c(0,3,1),c(-2,-1,4))
» det(x)
```

Inverse d'une matrice carrée

```
» solve(x)
```

Rang d'une matrice

» `qr(x)$rank`

Si le rang de la matrice carrée considérée est strictement inférieur à son ordre, la matrice n'est pas inversible. Si on utilise `solve` pour une matrice de rang non plein, R fournit un message d'erreur signalant que la matrice est singulière.

Matrice ordonnée selon une colonne

```
» x<-rbind(c(1,2,3),c(0,3,1),c(-2,-1,4))
» index<-order(x[,3])
» x[index,]
```

Fonctions statistiques

Les fonctions "statistiques" `sum, prod, mean, var, median, cumsum, cumprod, min, max,...etc` peuvent être appliquées directement aux *colonnes* ou aux *lignes* d'une matrice via la fonction `apply`.

```
» x<-rbind(c(1,2,3),c(0,3,1),c(-2,-1,4))
» x
» apply(x,1,"min")   (1 pour lignes)
» apply(x,2,"sum")   (2 pour colonnes)
» apply(x,2,"mean")
```

`colMeans` et `rowMeans` permettent d'obtenir les moyennes colonne par colonne et ligne par ligne respectivement. `colMeans(x)` équivaut à `apply(x,2,"mean")` et `rowMeans(x)` à `apply(x,1,"mean")` mais elles sont plus rapides. Sont définies de même des fonctions `colSums` et `rowSums` pour les sommes colonne par colonne et ligne par ligne.

```
» colMeans(x)
» rowMeans(x)
» colSums(x)
» rowSums(x)
```

Afin de soustraire, aditionner ou diviser les éléments d'une matrice par une même statistique colonne par colonne ou ligne par ligne, on peut utiliser `sweep`.

```
» xc<-sweep(x,2,colMeans(x))
» xc
» xcr<-sweep(xc,2,apply(x,2,sd),FUN="/")
» xcr
```

Le même résultat (colonnes centrées réduites) peut être obtenu à l'aide de `scale`.

```
» scale(x)
```

Vecteurs et matrices logiques

Opérateurs relationnels :

$$<, <=, >, >=, ==, != $$

Ces opérateurs peuvent être utilisés avec des scalaires, des vecteurs ou des matrices.
Le résultat de l'évaluation est TRUE (vrai) ou FALSE (faux).
Quand ces opérateurs sont appliqués à des vecteurs (resp. matrices), le résultat est un vecteur (resp. une matrice) de même longueur (resp. de mêmes dimensions), formé(e) de TRUE et de FALSE, résultats de comparaisons élément à élément.
Opérateurs logiques :

a & b (et/AND), a|b (ou/OR), !a (non/NOT), xor(a,b) (ou exclusif)

& peut être remplacé par &&
| peut être remplacé par ||

```
» x<-rbind(c(1,2,3),c(0,3,1),c(-2,-1,4))
» x
» x<2
» x[1,]<3 & x[1,]>1
» 0/0=="NaN"
» 1/0=="NaN"
» !(x<2)
```

L'égalité à "NaN" peut également être testée à l'aide de la fonction `is.nan()`. L'égalité à "NA" (valeur manquante) peut être testée à l'aide de la fonction `is.na()`.

```
» is.nan(0/0)
» is.nan(1/0)
» x
» is.nan(0/x)
» is.nan(x)
» y<-1:3
» y[1]<-NA
» y
» is.na(y)
```

Extraction d'éléments à l'aide de vecteurs ou matrices logiques

---

```
» x<-1:10
» x
» x[x<=5]<-6
```

```
» x
» x[x==7]<-NA
» x
» is.na(x)
» x[is.na(x)]<-0
» x
```

Comparaisons globales

`identical(x,y)` : compare les *représentations machine* de ses arguments et retourne TRUE si elles sont exactement identiques, FALSE sinon

`all.equal(x,y)` : porte sur l'*égalité approximative* entre ses arguments et retourne TRUE ou un résumé des différences. On peut spécifier l'écart absolu minimum considéré comme significatif à l'aide de l'option `tolerance`. Pour la valeur par défaut de `tolerance`, on pourra consulter l'aide en faisant `?all.equal`.

```
» c(0.9,0.8)==c(1-0.1,0.8)
» identical(c(0.9,0.8),c(1-0.1,0.8))
» all.equal(c(0.9,0.8),c(1-0.1,0.8))

» 0.9==1.1-0.2
» identical(0.9,1.1-0.2)
» all.equal(0.9,1.1-0.2)
» all.equal(0.9,1.1-0.2,tolerance=1e-16)

» s<-0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1
» s==1
» s<1
» s>1
» identical(s,1)
» all.equal(s,1)
» trunc(s)
» round(s)
» ceiling(s)
» floor(s)
```

Conversion en valeurs numériques

Les vecteurs et matrices logiques sont convertis en vecteurs et matrices numériques lorsqu'ils sont utilisés avec des fonctions qui requièrent des valeurs numériques. TRUE est converti en 1 et FALSE en 0.

```
» x<-c(2,1,4,NA)
» sum(is.na(x))>0
```

```
» x<-x[!is.na(x)]
» sum(is.na(x))>0
```

Fonctions statistiques en présence de valeurs manquantes

Lorsqu'un vecteur contient des valeurs manquantes, les fonctions statistiques renvoient NA lorsqu'elles lui sont appliquées. On peut spécifier que les valeurs manquantes ne doivent pas être prises en considération à l'aide de l'option `na.rm=TRUE`.

```
» x<-c(1,2,3,3,3,3,5,10,7,NA)
» max(x)
» max(x,na.rm=TRUE)
» min(x,na.rm=TRUE)
» range(x,na.rm=TRUE)
» mean(x,na.rm=TRUE)
» mean(x,trim=0.2,na.rm=TRUE)
» median(x,na.rm=TRUE)
» quantile(x,probs=c(0,0.1,0.9),na.rm=TRUE)
» var(x,na.rm=TRUE)

» y<-1:10
» cor(x[!is.na(x)],y[!is.na(x)])

» z<-c(12,2,3,5,14,7,9,8,2,3)
» pmax(x,y,z)
» pmin(x,y,z)
```

Chaînes de caractères

Deux exemples simples

```
» letters
» LETTERS
```

---

paste

```
» ch<-c("R","S")
» ch
» cch<-paste("language",ch,sep='*')
» cch
» cch<-paste("language",ch,sep='*',collapse="=")
» cch
```

---

grep

```
» ch<-c("ab","bc","ac")
» grep("a",ch)
```

Expressions

Une *expression* est une suite de caractères qui a un sens pour R. Toutes les commandes valides de R sont des expressions. Lorsqu'une commande est saisie au clavier, elle est ensuite *évaluée* par R afin d'être exécutée si elle est valide. Une chaîne de caractère peut être convertie en expression à l'aide de la fonction `parse`.

```
» x<-5
» expr<-parse(text="x<-7")
» expr
```

Une expression peut être évaluée à l'aide de la fonction `eval`.

```
» eval(expr)
» x
```

On peut également définir des expressions de façon directe.

```
» expr1<-expression(x/(y+exp(z)))
» expr1
» x<-1;y<-2;z<-3
» eval(expr1)
» expr1
```

Il est parfois utile de construire des expressions sans les évaluer. Par exemple pour le calcul des dérivées partielles d'une fonction.

```
» D(expr1,"x")
» D(expr1,"y")
» D(expr1,"z")
```

Probabilités avec R

Générateurs (pseudo)-aléatoires

R dispose de ne nombreux générateurs de nombres pseudo-aléatoires. Il s'agit de fonctions appelées `runif`, `rnorm`, ...etc. On les assimile généralement à des générateurs de nombres aléatoires en faisant les hypothèses suivantes.

Hypothèse 1 : Tout *appel* à un générateur de nombres pseudo-aléatoire de R associé à une loi $\mathbb{P}^{\otimes n}$ $(n = 1, 2, ...)$ est une variable aléatoire de loi $\mathbb{P}^{\otimes n}$

Hypothèse 2 : Les *appels* successifs à des générateurs de nombres pseudo-aléatoires de R sont des variables aléatoires indépendantes

Une discussion rigoureuse de ces hypothèse sort du cadre de cette introduction.

Sous les hypothèses précédentes, pour obtenir des réalisations de vecteurs de dimension n dont les éléments sont i.i.d. de loi *normale* centrée réduite, on peut utiliser la commande `rnorm(n,mean=0,sd=1)` ou, plus succinctement, `rnorm(n)`. Pour une distribution *uniforme* sur $[0, 1]$, on utilisera la commande `runif(n,min=0,max=1)` ou, plus succinctement, `runif(n)`.

```
» n <-1000
» A <- rnorm(n)
» B <- runif(n)
```

Tracé de l'histogramme de la série obtenue - échantillon N(0,1)

```
» nclasses<-50
» hist(A,breaks=nclasses,main="Histogramme - Echantillon N(0,1)")
```

Si on désire obtenir une distribution normale de moyenne $\mu = 5$ et de variance $\sigma^2 = 4$, on peut opérer la transformation

$$\text{A2}=\text{A}*\sigma+\mu$$

ou utiliser directement la commande `rnorm(n,mean=5,sd=2)`.

```
» A2<-rnorm(1000,mean=5,sd=2) ; nclasses<-50
» hist(A2,breaks=nclasses,main="Histogramme - Echantillon N(5,4)")
```

Tracé de l'histogramme de la série obtenue - échantillon U(0,1)

```
» nclasses<-50
» hist(B,breaks=nclasses,main="Histogramme - Echantillon U(0,1)")
```

Si on désire obtenir une distribution uniforme sur l'intervalle $[a, b] = [-2, 3]$, on peut opérer la transformation

$$\text{B2}=\text{B}^*(b-a)+a$$

ou utiliser directement la commande `runif(n,min=-2,max=3)`.

```
» B2<-runif(1000,min=-2,max=3)
» hist(B2,breaks=50,main="Histogramme - Echantillon U(-2,3)")
```

Générer une matrice aléatoire x de dimensions 20×5, dont les composantes sont indépendantes de loi $\mathcal{N}(2, 9)$.
Générer une matrice y de taille 20×5 dont toutes les composantes sont égales à 30.
Additionner les matrices x et y. On appelera z la matrice résultante.
Calculer les moyennes, les écart-types et les variances des colonnes de z.
Calculer les sommes des carrés des composantes des colonnes de z.

Liste des générateurs pseudo-aléatoires de R avec leur arguments entre parenthèses

---

Loi	*Générateur pseudo-aléatoire R*
Beta	`rbeta(n,shape1,shape2)`
Binomiale	`rbinom(n,size,prob)`
Binomiale Négative	`rnbinom(n,size,prob)`
Cauchy	`rcauchy(n,location,scale)`
Chi-Deux	`rchisq(n,df)`
Exponentielle	`rexp(n,rate)`
Fisher-Snedecor	`rf(n,df1,df2)`
Gamma	`rgamma(n,shape,scale)`
Gaussian	`rnorm(n,mean,sd)`
Géometrique	`rgeom(n,prob)`
Hypergéometrique	`rhyper(nn,m,n,k)`
Logistique	`rlogis(n,location,scale)`
Lognormale	`rlnorm(n,meanlog,sdlog)`
Multinomiale	`rmultinom(n,size,prob)`
Poisson	`rpoiss(n,lambda)`
Student	`rt(n,df)`
Uniforme	`runif(n,min,max)`
Weibull	`rweibull(n,shape,scale)`
Wilcoxon	`rwilcox(nn,m,n)`

Graphiques 2D

La commande `plot` permet de tracer des graphiques en 2D. Avec `plot(x,y)`, on trace y en fonction de x. x et y sont des vecteurs de *même dimension*.

Tracé de courbe

Définition des valeurs de x et calcul des valeurs de $y = f(x)$

```
» x<-seq(-pi,pi,0.1)
» y<-sin(x)
```

Tracé de la fonction

```
» plot(x,y,type="l")
```

L'option `type="l"` permet de spécifier que le graphique doit être une courbe. D'autres possibilités sont `type="p"` (points), `type="h"` (traits verticaux), `type="s"` (escaliers) et `type="n"` (ne rien tracer). Les essayer.

Documentation du graphique

```
» plot(x,y,type="l")
» grid()
» title(main="y = sin(x)",xlab="x",ylab="y")
» text(0,0,"(0,0)")
```

Afin d'imprimer un graphique dans un fichier ou sur papier, on peut utiliser les commandes `pdf, postscript, jpeg,...etc`.

```
» jpeg(file="sin.jpg")   (ouvre un fichier jpeg)
» plot(x,y,type="l")
» grid()
» title(main="y = sin(x)",xlab="x",ylab="y")
» text(0,0,"(0,0)")
» dev.off()   (ferme le fichier jpeg)
```

Lire l'image à l'aide de Mozilla Firefox, la redimensionner à l'aide de GIMP ou PAINT.

Par défaut `plot` ouvre une fenêtre graphique de type `windows`.

```
» windows()  (ouvre une nouvelle fenêtre windows)
» plot(x,y,type="l")
» grid()
» title(main="y = sin(x)",xlab="x",ylab="y")
» text(0,0,"(0,0)")
» dev.off()  (ferme la fenêtre windows)
```

Options de windows()

Les dimensions de la fenêtre graphiques peuvent être spécifiés (en inches) à l'aide des options `length` (hauteur) et `width` (largeur) . Par défaut, elles sont fixées à 7".

```
» windows()
» windows(height=7,width=12)
» windows(height=4,wideth=10)
```

Options de plot

`type` (type de graphique) peut prendre les valeurs "p", "l", "s" et "h"

`col` (couleur) peut prendre les valeurs "green", "blue", "red", "black", ...etc. La liste des couleurs ainsi disponibles peut être obtenues en faisant

```
» colors()
```

Pour les tracés continus (`type="l"`), `lty` (type de courbe) peut prendre les valeurs "solid" (trait plein), "dashed" (tirets), "dotted" (points), "dotdash" (point-tiret), "longdash" (longs tirets), "twodash" (tiret long - tiret court)

Pour les tracés discontinus (`type="p"`), `pch` (symbole de point) peut prendre les valeurs "+", "o", ".", "*",...etc, ainsi que les valeurs 1 à 25 (formes géométriques).

```
» plot(x,y,type="l",col="green",lty="solid")
» windows()
» plot(x,y,type="p",col="blue",pch="+")
» windows()
```

```
» plot(x,y,type="l",col="red",lty="dotdash")
```

On utilise `dev.off()` pour fermer la fenêtre graphique ou le fichier graphique en cours d'utilisation. Pour fermer toutes les fenêtres graphiques ou fichiers graphiques, on utilise `graphics.off()`.

```
» graphics.off()
```

Le tracé d'un nuage de points se fait à l'aide de la commande `plot(...,type="p")`.

Exemple : nuage de points aléatoires

```
» N<-100
» x<-runif(N)
» y<-runif(N)
» plot(x,y,type="p",pch="o")
» grid()
» title(main="Nuage de points aléatoires")
```

Répéter l'exemple avec $N = 10\ 000$
Répéter l'exemple avec `rnorm` au lieu de `runif`

La commande `barplot(y)` dessine un graphique en barres des valeurs de y.

```
» x<-seq(-2*pi,2*pi,pi/10)
» y<-cos(x)
» barplot(y)
» grid()
» title('Graphique en barres des valeurs d'une fonction)
```

Pour plus d'information sur `barplot` (options) faire

```
» ?barplot
```

`pie(x)` : Tacé du diagramme circulaire en 2D correspondant au vecteur ligne x (distribution de fréquences). Les éléments du vecteur sont normalisés par `x/sum(x)`.

```
» x<-c(0.25,0.40,0.35)
» pie(x)
» windows()
» pie(x,labels=c("x1","x2","x3"))
» title(main="Diagramme circulaire")
```

Pour plus d'information sur `pie` (options) faire

```
» ?pie
```

`hist(y,breaks=N)` : trace l'histogramme de la série des éléments de y pour N classes de même amplitude

---

```
» y<-rnorm(1000)
» hist(y,breaks=10,main="Histogramme d'une série normale (10 classes)")
» grid()
```

Au lieu de spécifier directement le nombre de classes, on peut passer en argument un vecteur qui spécifie les centres des classes.

```
» x<-seq(-5,5,0.2)
» hist(y,x,main="Histogramme à intervalle et espacements fixés")
» grid()
```

Pour un histogramme en fréquences *relatives*, utiliser l'option `freq=FALSE`.

```
» windows()
» hist(y,x,main="Histogramme à intervalle et espacements fixés",freq=FALSE)
» grid()
```

On peut surimposer la courbe de la densité normale à l'aide de la fonction `curve`.

```
» curve(exp(-x^2/2)/sqrt(2*pi),from=-5, to =5,add=TRUE)
```

La fonction `curve` permet plus généralement de tracer une courbe de fonction en spécifiant celle-ci soit par une expression en `x`, soit par son nom.

```
» windows()
» curve(sin,from=-2*pi, to =2*pi)
```

`pairs(x)` : Tracé de nuages de points entre les colonnes de la matrice x.

```
» x<-matrix(rnorm(150),nrow=50)
» y<-matrix(runif(150),nrow=50)
» pairs(x)
» x[,2]<-3*x[,1]+y[,1]
» pairs(x)
```

Pour plus d'information sur `pairs` (options) faire

```
» ?pairs
```

Autres commandes graphiques 2D : `boxplot, dotchart, qqplot, cdplot, stem, stars...etc.`

Graphiques 3D

Considérons la surface d'équation

$$z = \frac{\sin(x^2 + y^2)}{x^2 + y^2}$$

pour x et y variant de $-\pi$ à π avec un pas de $\pi/10$.

Définition de la fonction à représenter

```
» f<-function(x,y)
{
ans<- sin(x^2+y^2)/(x^2+y^2)
return(ans)
}
```

Les vecteurs x et y définissent le domaine de calcul de z.
Evaluation des valeurs de z

```
» x<- seq(-pi,pi,pi/10)
» y<-x
» z<-outer(x,y,f)
» z[is.na(z)]<-1
```
(traite le cas $x, y = 0$)

Tracé de la fonction

```
» persp(x,y,z,theta=30,phi=30)
```

Avec documentation du graphique

```
» persp(x,y,z,theta=30,phi=30,main="Tracé d'une surface",
xlab="x",ylab="y",zlab="z")
```

Avec davantage d'options

```
» persp(x,y,z,theta=30,phi=30,expand=0.5,col="lightblue",
ltheta=120,shade=0.75,ticktype="detailed", main="Tracé d'une surface",
xlab="x",ylab="y",zlab="z")
```

Pour plus d'information sur `persp` (options) faire

```
» ?persp
```

Autres commandes graphiques en 3D : image, contour, ...
Pour avoir un aperçu des autres possibilités graphiques de R, saisir les commandes

---

```
» demo(persp)
» demo(image)
» demo(graphics)
```

Partitionnement d'une fenêtre graphique

Il arrive fréquemment qu'on souhaite afficher plusieurs graphiques dans une même fenêtre. Ceci peut être fait de plusieurs façons avec R. On peut en effet utiliser les fonctions `split.screen`, `layout` ou `par`.

---

par

Un partitionnement rectangulaire de la fenêtre graphique peut facilement être réalisé à l'aide des options `mfrow` et `mfcol` de la fonction `par`.

```
» x<-rnorm(150)
» y<-runif(150)

» windows()
» par(mfrow=c(2,2))
» plot(x,y)
» hist(x)
» hist(y)
» plot(y,x)

» windows()
» par(mfcol=c(2,2))
» plot(x,y)
» hist(x)
» hist(y)
» plot(y,x)
```

---

split.screen

Pour des arrangements plus complexes que des arrangements rectangulaires, on peut utiliser `split.screen`. Cette fonction n'est toutefois pas compatible avec toutes les fonctions graphiques de R.

```
» x<-rnorm(150)
» y<-runif(150)

» windows()
» split.screen(c(1,2))
» screen(1)
» hist(x)
» close.screen(1)
» screen(2)
```

```
» hist(y)
» close.screen(2)

» windows()
» split.screen(c(2,1))
» screen(1)
» hist(x)
» close.screen(1)
» screen(2)
» hist(y)
» close.screen(2)
```

Une partie obtenue par `split.screen` peut également être divisée à l'aide de `split.screen`.

```
» windows()
» split.screen(c(2,1))
» screen(1)
» plot(x,y)
» close.screen(1)
» screen(2)
» split.screen(c(1,2))
» close.screen(2)
» screen(3)
» hist(x)
» close.screen(3)
» screen(4)
» hist(y)
» close.screen(4)
```

Il est recommandé, lorsqu'on utilise `split.screen` de s'occuper de chaque graphique de façon complète avant de passer au suivant.

Layout

La fonction `layout` partitionne la fenêtre graphique en plusieurs sous-fenêtres sur lesquelles sont affichés les graphiques successivement. Elle permet d'obtenir des arrangements complexes.

```
» windows()
» m<-matrix(1:4,nrow=2,ncol=2)
» m
» layout(m)
```

Un arrangement spécifié dans `layout` peut être visualisé à l'aide de la fonction `layout.show` qui prend en argument le nombre de sous-fenêtres.

```
» layout.show(n=4)
» plot(x,y)
» hist(x)
» hist(y)
» plot(y,x)
```

Les exemples qui suivent illustrent quelques uns des arrangements susceptibles d'être obtenus à l'aide de `layout`.

```
» m<-matrix(1:6,nrow=3,ncol=2)
» layout(m)
» layout.show(n=6)

» m<-matrix(1:6,nrow=2,ncol=3)
» layout(m)
» layout.show(n=6)

» m<-matrix(c(1:3,3),nrow=2,ncol=2)
» layout(m)
» layout.show(n=3)

» m<-matrix(c(1:3,3),nrow=2,ncol=2,byrow=TRUE)
» layout(m)
» layout.show(n=3)

» m<-matrix(c(2,1,4,3),nrow=2,ncol=2,byrow=TRUE)
» layout(m)
» layout.show(n=4)

» m<-matrix(scan(n=25),nrow=5,ncol=5)
1:0
2:0
3:3
4:3
5:3
6:1
7:1
8:3
9:3
10:3
11:0
```

```
12:0
13:3
14:3
15:3
16:0
17:2
18:2
19:0
20:5
21:4
22:2
23:2
24:0
25:5
» layout(m)
» layout.show(n=5)
```

Par défaut, `layout` partitionne la fenêtre graphiques de façon proportionnelle. On peut spécifier des dimensions relatives pour les différentes sous-fenêtres à l'aide des options `widths` et `heights`.

```
» m<-matrix(1:4,nrow=2,ncol=2)
» layout(m,widths=c(1,3),heights=c(3,1))
» layout.show(n=4)

» m<-matrix(c(1,1,2,1),nrow=2,ncol=2)
» layout(m,widths=c(2,1),heights=c(1,2))
» layout.show(n=2)

» m<-matrix(0:3,nrow=2,ncol=2)
» layout(m,widths=c(1,3),heights=c(1,3))
» layout.show(n=3)
```

Objets

Les vecteurs, matrices, fonctions et expressions vues jusqu'ici sont traitées par R commes des *objets*. Un objet R est caractérisé par son nom et son contenu mais également par ses *attributs*. On distingue des *attributs intrinsèques* : *type*, *mode* et *length* (longueur), liés à la façon dont R stocke l'objet sur machine, et des éventuels *attributs extrinsèques* : *class, comment, dim, dimnames, names, row.names, rownames, colnames, tsp* et *levels* qui vont déterminer la façon dont les fonctions agissent sur l'objet.

Vecteurs

```
» x <- 1:10
```

Le *type* d'un objet détermine la façon dont R le stocke sur machine. Le *mode* d'un objet est fonction de son type et en donne une version moins détaillée. Ainsi, pour un vecteur, les types `integer` et `double` seront tous deux résumés par le mode `numeric`.

```
» x
» typeof(x)
» mode(x)
» x<-x+0.5
» x
» typeof(x)
» mode(x)
```

Un vecteur logique est de type et de mode `logical`. Il existe également un type et mode `complex`, mais il ne sera pas discuté ici.

```
» t<-(x<5)
» typeof(t)
» mode(t)
```

L'attribut *class* d'un objet détermine la façon dont il est traité par différentes fonction de R. La fonction `class` permet d'obtenir la classe d'un objet. Pour un vecteur, la classe est implicitement identique au mode mais peut être modifiée par l'utilisateur.

```
» x
» class(x)
» class(x)<-"a"
» x
» class(x)<-"numeric"
```

```
» x
```

L'attribut *length* peut être utilisé pour obtenir ou modifier la longueur d'un vecteur.

```
» x
» length(x)
» length(x)<-3
» x
» length(x)<-5
» x
```

L'attribut *names* peut être utilisé pour affecter des labels ("noms") aux différentes composantes d'un vecteur.

```
» names(x)
» names(x)<-c("A","B","C","D","E")
» x
» names(x)
```

Extraction d'un élément d'un vecteur à l'aide de son label

```
» x["B"]
» x[c("A","B")]
» x[c("F","A","B")]
```

L'attribut *comment* peut être utilisé pour associer un commentaire à un vecteur.

```
» comment(x)<-c("Un exemple simple","08.12.2010")
» comment(x)
» x
```

Matrices

```
» y <- matrix(1:9,nrow=3,ncol=3,byrow=TRUE)
» y
» typeof(y)
» mode(y)
» length(y)
» class(y)
```

```
» z <- matrix(seq(from=1,to=2,length=9),nrow=3,ncol=3,byrow=TRUE)
» z
» typeof(z)
» mode(z)
» length(z)
» class(z)

» m <- (z<1.5)
» m
» typeof(m)
» mode(m)
» length(m)
» class(m)
```

Ainsi qu'on l'a déjà vu, l'attribut `dim` correspond aux dimensions d'une matrice. Il peut être obtenu ou modifié à l'aide de la fonction `dim`. La longueur d'une matrice correspond à la somme de ses dimensions.

```
» y <- matrix(1:6,nrow=2,ncol=3,byrow=TRUE)
» y
» dim(y)
» dim(y)<-c(3,2)
» y
```

L'attribut `rownames` permet d'affecter des labels ("noms") aux lignes d'une matrice. La fonction `rownames` correspondante permet d'obtenir la valeur de cet attribut ou d'en modifier la valeur.

```
» rownames(y)
» rownames(y)<-c("a","b","c")
» y
```

Extraction d'une ligne d'une matrice à l'aide de son label

```
» y["a",]
```

L'attribut `colnames` permet d'affecter des labels ("noms") aux colonnes d'une matrice. La fonction `colnames` correspondante permet d'obtenir la valeur de cet attribut ou d'en modifier la valeur.

```
» colnames(y)
» colnames(y)<-c("x1","x2")
» y
```

Extraction d'une colonne d'une matrice à l'aide de son label

```
» y[,"x1"]
```

Extraction d'un élément d'une matrice à l'aide des labels de ligne et de colonne

---

```
» y["a","x1"]
```

De même que pour les vecteurs, l'attribut *comment* peut être utilisé pour associer un commentaire à une matrice.

```
» comment(y)<-c("Un exemple simple de matrice","09.12.2010")
» comment(y)
» y
```

Fonctions

Les fonctions R sont de mode `function`. Celui-ci correspond à trois types : `special`, `builtin` et `closure`, les deux premiers types correspondant aux fonctions et operateurs de base disponibles sous R.

```
» f <- function(x,y) {return(x+y-2)}
» f
» typeof(f)
» mode(f)
```

La longueur ("length") d'une fonction est toujours égale à 1. Sa classe est par défaut identique à son mode. Il est possible d'associer un commentaire à une fonction à l'aide de l'attribut `comment`.

```
» length(f)
» class(f)
» comment(f)<- c("Un exemple simple de fonction", "09.12.2010")
» comment(f)

» typeof(sin)
» mode(sin)
» length(sin)
» class(sin)
```

Expressions

Les expressions R sont de type, mode et par défaut de classe `expression`. Leur longueur est fixée à 1. De même que les vecteurs, matrices et fonction, il est possible d'associer un commentaire à une expression à l'aide de l'attribut `comment`.

```
» expr <- expression(x/(y+exp(z)))
» expr
» typeof(expr)
» mode(expr)
» length(expr)
» class(expr)
» comment(expr)<- c("Un exemple d'expression R", "09.12.2010")
» comment(expr)
```

Sauvegarde d'objet

```
» x<-c(1,2,3)
» save(x,file="fichier.Rdata")
```

Le fichier obtenu aura pour extension `.Rdata`, ce qui permettra de la reconnaître comme fichier de données R, et sera sauvegardé dans le répertoire de travail sous le nom `fichier.Rdata`.

On peut effacer toutes les objets de la mémoire :

```
» rm(list=ls())
» x
```

Si l'on charge le fichier `fichier.Rdata`, l'objet x est de nouveau présent dans l'espace de travail :

```
» load("fichier.Rdata")
» x
```

Pour afficher (presque) *tous* les objets en mémoire, on peut utiliser les instructions `ls(), objects()`

```
» x<-c(1,2,3)
» y<-1
» z<-2
» ls()
» objects()
```

Pour effacer les variables x et y :

```
» rm(x,y)
» ls()
```

Pour afficher un résumé succinct du contenu d'un objet R : `str`

```
» x<-c(1,2,3)
» y<-1
» f <- function(x,y) {return(x+y-2)}
» expr <- expression(x/(y+exp(z)))

» str(x)
» str(y)
» str(f)
» str(expr)
```

Pour afficher un résumé succinct de tous les objets présents dans l'espace de travail :

```
» ls.str()
```

Fonctions génériques

Certaines fonctions R agissent différemment sur un objet en fonction de sa classe. De telles fonctions sont dites *génériques*. Elles ne font rien en elles-mêmes si ce n'est appeler la fonction (appelée *méthode*) adaptée à la classe de l'objet en question parmi un ensemble de fonctions associées.

Exemple : `summary`

```
» x<-c(1,2,3)
» y<-matrix(1:6,nrow=2)
» expr <- expression(x/(y+exp(z)))
```

```
» summary(x)
» summary(y)
» summary(expr)
```

Pour afficher l'ensemble des méthodes associées à une fonction générique : `methods`

```
» methods("summary")
```

On voit ainsi les différentes classes des objets qu'on peut résumer à l'aide de `summary`.

Deux autres exemples de fonctions génériques : `print` et `plot`. Afficher les méthodes correspondantes.

Fichiers et programmation

R peut exécuter des séquences d'instructions stockées dans des fichier. Ces fichiers sont appelés "scripts". Leur extension est `.R`.
Un **script** est une séquence d'instructions R.
Les variables d'un fichier script sont généralement globales (à moins qu'elles apparaîssent dans une définition de fonction).
Les valeurs des variables de l'espace de travail peuvent être modifiées par un fichier script.
Une première utilisation des fichiers scripts est la *lecture et mise en forme de données.*
Editeur de scripts : menu Files/New Script.

Sauvegarder le script

```
A<-matrix(1:6,nrow=2)
```

---

sous le nom donnees.R puis l'executer par Ctrl a/Ctrl r ou en utilisant le menu Edit/Run All.

Examiner les variables présentes dans l'environnement de travail.

```
» ls()
```

Au lieu d'éxécuter le script en le sélectionnant, on peut utiliser le fonction `source`.

```
» rm(A)
» source("donnees.R")
» ls()
```

Script courbe1.R (récupérer dans le presse-papier Acrobat Reader)

```
x<-seq(-5,5,0.1)
y<-x^2+5
plot(x,y,type="l")
grid()
title(main="Tracé de y=x^2 + 5",xlab="x",ylab="y")
```

Pour lancer un script, il suffit d'ouvrir le fichier correspondant (Fichier/Ouvrir Un Script) puis de l'éxécuter

entièrement (Ctrl a/Ctrl r) ou partiellement.

`Ctrl a/Ctrl r`

On peut également utiliser directement la fonction `source`.

» `source("courbe1.R")`

Pour insérer une ligne de commentaire dans un programme : utiliser le symbole #.

Les scripts peuvent être utilisés pour définir des fonctions R.

Exemple

Nous allons écrire une fonction générant un tableau de n nombres aléatoires entiers compris entre 0 et une valeur maximale contenue dans une variable notée `max`.

Fichier de fonction randint.R (récupérer dans le presse-papier Acrobat Reader)

```
randint <- function(n,max)
# res : vecteur de n entiers compris entre 0 et max
# runif : génère un nombre aléatoire entre 0 et 1
# floor : renvoie la partie entière d'un nombre
{ temp<-runif(n)
res<-floor((max+1)*temp)
return(res)}
```

Il est préférable de donner à ce type de fichier un nom identique à celui de la fonction.
Ainsi, l'exemple sera sauvegardé sous le nom `randint.R`.

La première ligne déclare le nom de la fonction et les arguments d'entrée.
Pour appeler une fonction, il suffit d'exécuter le script correspondant puis de procéder selon la syntaxe suivante :

```
resultat <- nom_fonction(liste des arguments d'appel)
```

Exemple

```
» nb_alea<-randint(10,50)
» nb_alea
```

On peut également appeler la fonction sous la forme

```
» randint(n=10,max=50)
```

Il est souhaitable de faire suivre la première ligne d'un fichier de fonction par des lignes de commentaire dans lesquels on décrit son but et ses arguments.

On peut éviter d'ouvrir le fichier de fonction en utilisant la commande `source`. Elle permet également à un fichier de fonction ou un script quelconque de charger une fonction R.

Exemple

```
» rm(list=ls())
» randint(10,50)
» source("randint.R")
» randint(10,50)
```

R dispose des instructions de contrôle : `for, while, if, else, repeat, break, switch`

if, else

Exemple : Script flog.R

```
flog <- function(x)
{
# Une fonction simple
if (x<0) return(x) else return(log(x))
}

» flog(2)
» flog(-2)
```

---

for

Exemple : Script ncarres.R

```
ncarres <- function()
{
# tableau des carrés des 10 premiers entiers naturels
n <- 10
x <- NULL #vecteur vide
for (i in 1:n)
{x<-c(x,i^2)}
cat(x,"\n")# Affiche le résultat obtenu
}
```

```
Ctrl a/Ctrl r
» ncarres()
```

Des boucles `for` peuvent notamment être utilisées lors de la création d'objets R. Dans ce cas l'objet doit être créé en dehors de la boucle.

```
» x<-NULL
» for (i in 1:10) x[i]<-i
» x

» y<-matrix(0,nrow=3,ncol=4)
» for (i in 3) { for (j in 1:4) { x[i,j]<-i+j}}
» y
```

---

while, repeat

Exemple : Script nx.R

```
nx <- function()
{
# plus petit entier n tq 2^n >= 15
x<-15
n<-0
while (2^n < x)
{ n <- n+1 }
```

```
cat(n,"\n")# Affiche le résultat obtenu
}
```

Ctrl a/Ctrl r
» nx()

Au lieu de `while`, on aurait pu utiliser un `repeat` de la façon suivante :

```
nx2 <- function()
{
# plus petit entier n tq 2^n >= 15 (version 2)
x<-15
n<-0
repeat{
n <- n+1
if (2^n >= x) break }
cat(n,"\n")# Affiche le résultat obtenu
}
```

Ctrl a/Ctrl r
» nx2()

---

stop, warning

`stop` renvoie un message d'erreur et interrompt l'exécution de la fonction ou du script en cours.

Exemple : Script sf.R

```
sf <- function(x)
{
# Une fonction simple
if (x>0) return(log(x)) else stop("x est inférieur ou égal à 0")
}
```

» sf(2)
» sf(-2)

`warning` renvoie un message mais n'interrompt pas l'exécution de la fonction ou du script en cours.

Exemple : Script narm.R

```
narm <- function(x)
{
# Supprime les valeurs manquantes
if (sum(is.na(x))>0) {
warning("Les valeurs manquantes ont été supprimées")
return(x[!is.na(x)])
}
}
```

```
» x<-c(1:5,NA)
» x<-narm(x)
» x
```

switch

Exemple : script centre.R

```
centre <- function(x, type) {
switch(type,
mean = mean(x),
median = median(x),
trimmed = mean(x, trim = .2),
stop("Choix non valide"))
}
```

```
» x <- rcauchy(10)
» centre(x, "mean")
» centre(x, "median")
» centre(x,"trimmed")
» centre(x,"trim")
```

dump, dput, dget

Les scripts peuvent également être utilisés pour sauvegarder des objets R. Cela évite d'avoir à distinguer fichiers de code et fichiers de données.

```
» x<-1:10
» y<-matrix(seq(0,1,length=6),nrow=2,byrow=TRUE)

» dump("x","x.R")
```

```
» rm(x)
» x
» source("x.R")
» x

» dput(y,"y.R")
» rm(y)
» y
» dget("y.R")
» y
» y<-dget("y.R")
» y
```

On pourra ouvrir les scripts `x.R` et `y.R` avec l'éditeur de scripts afin de visualiser le code obtenu.

Entrées/Sorties Console et sorties fichiers

Affichage à la console

```
» x<-1:10
» x
» print(x)
» cat(x,"\n")

» x<-"Bonjour"
» x
» print(x)
» cat(x,"\n")
```

Remarque : `"\n"` signifie un retour à la ligne.

Lecture de vecteurs à la console

```
exemple1 <- function()
{
# Lecture de 10 chiffres à la console et calcul de leur somme
cat("Saisissez 10 chiffres :","\n")
x <- scan(n=10)
total <- sum(x)
cat("Le total des 10 chiffres est :",total,"\n")
}

» exemple1()
```

Lecture de chaînes de caractères à la console

```
exemple2 <- function()
{
# Simulation de 10 réalisation issues de la loi normale ou uniforme, au choix
cat("Nom de la loi des 10 réalisations ?","\n")
cat("normale ou uniforme : ")
loi <- readline()
switch(loi,normale=rnorm(10),uniforme=runif(10),stop("Choix non valide"))
}

» exemple2()
```

Ecriture simple dans un fichier

```
» x<-1:10
» cat(x,"\n",file="afile.txt")

» x<-"Bonjour"
» cat(x,"\n",file="afile.txt",append=TRUE)
```

Le fichier "afile.txt" est créé par cat s'il n'existe pas. L'option `append=TRUE` spécifie que l'écriture doit se faire en fin de fichier, c'est-à-dire à la suite du contenu existant. On pourra ouvrir le fichier "afile.txt" pour visualiser le résultat obtenu.

Redirections

La fonction `sink` permet de rediriger les sorties R vers un fichier – en créant celui-ci si nécessaire – tout en conservant la possibilité de les afficher également à la console (option `split=TRUE`). L'utilisation de `sink` est particulièrement intéressante pour des sorties longues ou des sorties qui ne sont pas effectuées à l'aide de `cat`.

```
» sink("myfile.txt",split=FALSE)
» cat("Un exemple simple :","\n")
» cat(letters,"\n")
» sink()

» sink("myfile.txt",append=TRUE,split=FALSE)
» cat("En majuscules :","\n")
» cat(LETTERS,"\n")
» sink()

» sink("myfile.txt",append=TRUE,split=FALSE)
» cat("Et pourquoi pas du numérique :","\n")
» log(10)
» sin(10)
» A <- matrix(1:6,nrow=2)
» print(A)
» sink()
```

L'option `append=TRUE` spécifie que l'on doit écrire en fin de fichier et non au début (valeur par défaut). On pourra ouvrir le fichier "myfile.txt" pour visualiser le résultat obtenu.

Connexions

Il est possible d'ouvrir un fichier en écriture sous R à l'aide de la commande `file`. On dit qu'on ouvre une connexion. Après usage, une connexion doit être fermée à l'aide de l'instruction `close`.
Lors d'un recours à `sink("fichier.txt")`, une connexion est en fait automatiquement ouverte vers "fichier.txt" ; elle est refermée par la commande `sink()`. L'intérêt de `file` est que la connexion est ouverte indépendamment de `sink` et peut donc être utilisée par d'autres fonctions.

```
» f<-file("newfile.txt",open="w")
» f
```

L'option `open="w"` spécifie que le fichier est ouvert en écriture (le fichier est créé s'il n'existe pas).

```
» sink(f,split=TRUE)
» cat("Un exemple simple","\n")
» cat("sur deux lignes","\n")
» cat("ou plutôt trois")
» sink()

» A<-matrix(1:6,nrow=2)
» write(A,file=f,ncolumns=3,append=TRUE)
```

La fonction `sink` permet de réorienter les sorties à suivre de la console vers la connexion `f`, c'est-à-dire vers le fichier "newfile.txt". L'option `split=TRUE` spécifie que les sorties doivent également apparaître à la console. La fonction `write` est spécifiquement adaptée à la sauvegarde de matrices au format texte (consulter l'aide). On pourra ouvrir le fichier "newfile.txt" avec l'éditeur de scripts pour visualiser la sortie affichée.

```
» close(f)
» f

» unlink("newfile.txt")
» unlink("myfile.txt")
» unlink("afile.txt")
```

La commande `close(f)` ferme la connexion. La commande `unlink("fichier.txt")` permet d'effacer le fichier "fichier.txt".

Listes

La fonction `list` peut être utilisée pour combiner des objets de différents types ou modes en un seul objet de type, de mode et par défaut de classe `list`. La longueur d'une liste correspond au nombre d'objets qui y sont stockés.

```
» rm(list=ls())
» L <- list(1:3,"hello",f)
» L
» typeof(L)
» mode(L)
» class(L)
» length(L)
```

On peut assigner des labels ("noms") aux différents éléments d'une liste à l'aide de l'attribut `names`. Ceci peut être fait soit lors de la création de la liste, soit à l'aide de la fonction `names`.

```
» names(L)
» names(L)<-c("A","B","C")
» L

» L2 <- list(A2=1:3,B2="hello",C2=f)
» L2
» names(L2)
```

Extraction d'élements d'une liste

```
» L[[1]]
» L[[1]][2]
» L[[2]]
» L[[3]]
» L[[3]](1,2)

» L$A
» L$A[2]
» L$B
» L$C
» L$C(1,2)

» L[["A"]]
» L[["C"]]
» L[["C"]](1,2)
```

Distinction entre `L[1]` *et* `L[[1]]`

```
» L[1]
» mode(L[1])

» L[[1]]
» mode(L[[1]])
```

Concaténation de listes

```
» LL <- c(L,L2)
» mode(LL)
```

De même que pour les vecteurs, matrices, fonctions et expressions, on peut utiliser l'attribut `comment` pour associer un commentaire à une liste.

```
» comment(LL)<- c("Liste obtenue par concaténation","09.12.2010")
» comment(LL)
```

Attacher/Detacher une liste

Afin de faciliter l'accès aux éléments d'une liste et pour en alléger la notation, il est possible d'*attacher* une liste. On peut alors accéder aux composantes de la liste directement avec leurs noms. Une fois cette utilisation terminée, il est nécessaire de *détacher* la liste. On ne peut alors accéder aux composantes de la liste qu'avec la syntaxe usuelle ($ ou [[ ]]).

```
» LL
» attach(LL)
» A
» B
» detach(LL)
» A
```

Attribut `dimnames` des objets de classe `matrix`

```
» y <- matrix(1:6,nrow=3,ncol=2,byrow=TRUE)
```

```
» rownames(y)<-c("a","b","c")
» colnames(y)<-c("x1","x2)
» y

» dimnames(y)
» dimnames(y)<-list(LIGNES=c("i1","i2","i3"),COLONNES=c("V1","V2"))
» y
» dimnames(y)
```

Return

Lorsque le résultat d'une fonction est multiple, on peut utiliser la fonction `return` pour le renvoyer sous la forme d'une liste.

Exemple

```
» mvs<-function(x)
{
x.mean <- mean(x)
x.var <- var(x)
x.sum <- sum(x)
return(value=list(MOY=x.mean,VAR=x.var,SOM=x.sum))
}
» x<-1:10
» mvs(x)
» class(mvs(x))
```

Quelques fonctions de matrices dont le résultat est une liste

---

`qr(x)`	décomposition QR de la matrice `x`
`chol(x)`	décomposition de Cholesky - - -
`svd(x)`	décomposition en valeurs singulières - - -
`eigen(x)`	valeurs et vecteurs propres - - -

Facteurs

La fonction `factor` peut être utilisée pour définir des objets de classe *factor*. La notion de facteur correspond en Statistique à celle de variable catégorielle. Des facteurs peuvent également être définis par conversion directe de vecteurs ou par regroupements d'éléments d'un vecteur.

```
» rm(list=ls())
» age <- factor(c(1,1,2,2,1,3,1,2),labels=c("20-35ans","35-55ans","+55ans"))
» age
» typeof(age)
» mode(age)
» length(age)
» class(age)
```

L'attribut `levels` correspond aux valeurs possibles des éléments d'un facteur. Il peut être obtenu ou modifié à l'aide de la fonction `levels`.

```
» levels(age)
» levels(age)<-c("20-35ans","35-50ans","+50ans")
» age
```

Conversion d'un vecteur de mode numérique

```
» age <- c(1,1,2,2,1,3,1,2)
» age <- factor(age,labels=c("20-35ans","35-50ans","+50ans"))
» age

» age <- c(1,1,2,2,1,3,1,2)
» age <- as.factor(age)
» age
» levels(age)<-c("20-35ans","35-50ans","+50ans")
» age
```

Conversion d'un vecteur de mode caractère

```
» age <- c("20-35ans","20-35ans","35-50ans","35-50ans","20-35ans",
"+50ans","20-35ans","35-50ans")
» age
» age <- as.factor(age)
» age
```

Regroupement de valeurs d'un vecteur de mode numérique

```
» age <- c(22,31,37,49,27,60,34,47)
» age <- cut(age,breaks=c(20,35,50,80),labels=c("20-35ans","35-50ans","+50ans"))
» age
```

`cut` peut également être utilisée pour obtenir des regroupements en classes d'amplitudes égales.

```
» age <- c(22,31,37,49,27,60,34,47)
» age <- cut(age,breaks=3)
» age
```

La fonction `pretty` permet d'obtenir des limites de classes plus raisonnables.

```
» age <- c(22,31,37,49,27,60,34,47)
» age <- cut(age,breaks=pretty(age))
» age
```

Facteurs ordonnés

```
» age <- ordered(c(1,1,2,2,1,3,1,2),levels=c(1,2,3),
labels=c("20-35ans","35-50ans","+50ans"))
» age
» typeof(age)
» mode(age)
» length(age)
» class(age)
```

La classe d'un facteur ordonné est `c("ordered","factor")`. Cela signifie qu'outre les fonctions spécifiques à la classe `ordered`, les fonctions spécifiques à la classe `factor` et sans équivalent pour la classe `ordered` pourront être appliquées à un tel objet.

Définition d'un facteur ordonné par conversion

```
» age <- c(1,1,2,2,1,3,1,2)
» age <- ordered(age,labels=c("20-35ans","35-50ans","+50ans"))
```

```
» age

» age <- c("20-35ans","20-35ans","35-50ans","35-50ans","20-35ans",
"+50ans","20-35ans","35-50ans")
» age
» age <- as.ordered(age)
» age
» levels(age) <- c("20-35ans","35-50ans","+50ans")
» age

» age2 <- c(22,31,37,49,27,60,34,47)
» age2 <- cut(age2,breaks=pretty(age))
» age2 <- as.ordered(age2)
» age2
```

Tableaux de contingence

```
» table(age)

» sexe <- factor(c(1,2,2,1,2,1,2,2),labels=c("F","H"))
» table(sexe,age)
```

---

tapply

Afin d'appliquer une même fonctions aux cellules d'un tableau de contingence, on peut utiliser la fonction R `tapply`. Supposons qu'on souhaite obtenir la pression artérielle moyenne pour chacun des groupes sexe/age.

```
» press <- c(118,125,128,127,110,140,130,120)
» tapply(press,list(sexe,age),mean)
```

---

grep

De façon analogue à son utilisation pour les chaînes de caractères, la fonction `grep` peut être utilisée pour obtenir les indices correspondant à un niveau donné d'un facteur.

```
» grep("20-35ans",age)
```

Tableaux de Données (Data Frames)

Les *tableaux de données* R (data frames) sont des objets permettant de représenter des données de type "feuille de calcul". Chaque ligne correspond à une unité statistique (individu) et chaque colonne à une variable. L'avantage des tableaux de données par rapport aux matrices est que les colonnes peuvent être des vecteurs de classes différentes : numérique, logique, facteur...etc.

Définition d'un tableau de données

```
» rm(list=ls())
» td <- data.frame(sexe=rep(c("H","F"),c(2,3)),temps=round(rexp(5),2))
» td
» typeof(td)
» mode(td)
» length(td)
» class(td)
```

Les tableaux de données sont de mode `list` et de ce fait, des éléments peuvent en être extraits de la même façon que pour les listes.

```
» td$temps
» td$sexe
» td[[1]]
» td[[2]]
```

De même que les matrices, les tableaux de données possèdent les attributs `dim`, `rownames`, `colnames`, `dimnames` et `comment`.

```
» dim(td)
» rownames(td)
» colnames(td)
» dimnames(td)
» comment(td)<-c("Un exemple simple de data frame", "11.12.2010")
» comment(td)

» dimnames(td)<- list(letters[1:5],c("Sexe","Temps"))
» td
```

Ajout d'une ligne

```
» f<-data.frame(Sexe="H",Temps=round(rexp(1),2))
» td<-rbind(td,f)
» rownames(td)[6]<-"f"
» td
```

Ajout d'une colonne

```
» Age<-c(28,21,35,22,22,26)
» td<-cbind(td,Age)
» td
```

Attacher/Détacher un tableau de données

De même que pour les listes, il est possible d'attacher un tableau de données de façon à appeler les colonnes du tableau simplement par leur nom.

```
» attach(td)
» Sexe
» Age
» detach(td)
» Sexe
» Age
» td$Sexe
```

Remarque importante : La commande `attach` crée en fait une copie du tableau de données. Aucun changement effectué sur une des colonnes alors que le tableau est attaché n'est sauvegardé dans le tableau initial.

Merge de tableaux de données

```
» td2 <- data.frame(Temps2=round(rexp(6),2))
» rownames(td2) <- c("e","f","a","d","b","c")
» td2
```

Les individus ne sont pas dans le même ordre dans td2. On ne peut donc recoller les tableaux de données directement.

```
» Noms<-rownames(td2)
» td2 <- cbind(Noms,td2)
```

```
» index <- order(td2$Noms)
» td2 <- td2[index,]
» td2
```

Les individus sont à présent dans le même ordre dans td2 et td. On peut effectuer le "merge" par simple concaténation de colonnes.

```
» td3 <- cbind(td,td2)
```

Lorsqu'il n'y a pas d'ordre évident sur les identifiants des individus, on peut utiliser la fonction `match`.

```
» match(c("B","O","N","J","O","U","R"),LETTERS)
» letters[match(c("B","O","N","J","O","U","R"),LETTERS)]
```

A présent, supposons qu'on dispose d'un tableau de données td4 où tous les individus ne sont pas représentés et où un individu supplémentaire apparaît.

```
» td4 <- data.frame(Age=c(21,26,28,37),Temps3=round(rexp(4),2))
» rownames(td4) <- c("b","f","a","g")
» td4
```

Nous pouvons effectuer un "merge" à l'aide des identifiants des individus ("Noms").

```
» Noms<-rownames(td4)
» td4 <- cbind(Noms,td4)

» td5 <- merge(td3,td4,by="Noms",all=TRUE)
» td5
```

L'option `by` permet de spécifier la variable selon laquelle le recollement est effectué ; l'option `all=TRUE` spécifie que tous les individus apparaissant dans au moins l'un des deux tableaux de données doivent être inclus dans le tableau résultant.

Sauvegarde de tableaux de données au format texte

```
» write.table(td5,file="td5.txt")
```

Par défaut, le séparateur de champs est " ". Si l'on souhaite que ce soit ",", on peut procéder de la façon suivante :

```
» write.table(td5,file="td5.csv",sep=",")
```

Lecture de tableaux de données au format texte

```
» td6 <- read.table(file="td5.txt",header=TRUE)
```

Si le séparateur de champs est "," (fichiers .csv ou .CSV) au lieu d'être " ", on pourra procéder de la façon suivante :

```
» read.table(file="td5.csv",header=TRUE,sep=",")
```

ou

```
» read.csv(file="td5.csv",header=TRUE)
```

Pour d'autres options et variantes de `write.table` et `read.table`, on pourra consulter l'aide en ligne. Pour l'importation de données aux formats Stata, SPSS, SAS, Fortran, Epiinfo, on consultera le menu menu Aide/Manuels (en PDF)/R data Import/Export.

Editeur de tableaux de données

Le sexe de l'individu `g` est en fait `H`. Nous pouvons apporter cette correction au tableau de données à l'aide de l'éditeur R.

```
» td6 <-edit(td6)
» td6
```

Résumés d'un tableau de donnée

Les fonctions `str` et `summary` sont bien définies pour les tableaux de données et permettent d'en donner un aperçu succinct ; du point de vue structurel pour `str` et du point de vue statistique pour `summary`.

```
» td6
» str(td6)
» summary(td6)
```

Séries Temporelles

Les objets R de classe `ts` ou `mts` peuvent être utilisés pour représenter des séries temporelles. La fonction `ts` permet de créer un objet de classe `ts` à partir d'un vecteur (série temporelle simple) ou de classe `mts` à partir d'une matrice (série temporelle multiple).

```
» s<-ts(data=1:20, start = 1959)
» s
» typeof(s)
» mode(s)
» length(s)
» class(s)

» s<-ts(data=1:10, frequency=4, start = c(1959,2))
» s
» typeof(s)
» mode(s)
» length(s)
» class(s)

» s<-ts(data=cumsum(rexp(25)), frequency=12, start = c(1959,2))
» s
» typeof(s)
» mode(s)
» length(s)
» class(s)
```

L'option `frequency` spécifie le nombre d'observations par année, `start` le temps de la première observation sous la forme `start=année` ou `start=c(année,numéro de période)`.

```
» s2<-ts(data=matrix(abs(cumsum(rnorm(26))),nrow=13),frequency=12,start=c(1971,3))
» s2
» typeof(s2)
» mode(s2)
» length(s2)
» class(s2)
```

Une série temporelle multiple est en fait de classe `c("mts","ts")`. Cela signifie qu'outre les fonctions spécifiques à la casse `mts`, les fonctions spécifiques à la classe `ts` et sans équivalent pour la classe `mts` pourront être appliquées à une série temporelle multiple sous R.

On peut spécifier le nom des séries à l'aide de l'option `names` de `ts` (voir l'aide) ou de l'attribut `colnames` de la

série obtenue.

```
» colnames(s2)<-c("X1","X2")
» s2
```

Représentations graphiques

```
» plot(s)
» windows()
» plot(s2)
```

Packages

Un des grands avantages de R est qu'il est enrichi en permanence de nouveaux *packages*, c'est-à-dire de "paquets" de programmes spécifiquement conçus pour traiter un certain type de problèmes ou dédiés à un certain type d'applications. Il existe par exemple des packages dédiés à la Statistique non paramétrique, au Traitement du signal, aux Statistiques pour l'environnement, aux représentations graphiques...etc.
Sous Windows, l'installation d'un nouveau package se fait via le menu "Packages". La plupart des packages peuvent être téléchargés directement à partir du site cran.r-project.org. Une fois un package installé, il doit être chargé sous R, à l'aide de la commande `library`. La commande `library()` donne la liste des packages installés et pouvant être chargés.

```
» library()
» library(graphics)
» library(datasets)
```

Certains packages sont automatiquement chargés au démarrage de R. La liste de ces packages peut être obtenue de la façon suivante :

```
» search()
```

Pour obtenir de l'aide à propos d'un package donné (notamment un descriptif et une liste des principales fonctions définies dans le package) :

```
» library(help=graphics)
```

Le package `datasets` est spécifiquement dédié à la mise à disposition de jeux de données pour les différentes applications de R. Pour obtenir la liste des jeux de données disponible dans le package `dataset` :

```
» data()
```

Pour obtenir la liste des jeux de données disponibles dans un package installé :

```
» data(package="cluster")
```

Pour obtenir la liste des jeux de données disponibles dans tous les package installés :

```
» data(package = .packages(all.available = TRUE))
```

Pour charger un jeux de données :

```
» data(LakeHuron)
» LakeHuron

» data(flower,package="cluster")
» flower
```

Pour obtenir un descriptif du jeux de données chargé :

```
» help(LakeHuron)
» help(flower,package="cluster")
```

Probabilités avec R

Loi normale standard

Sous R, les valeurs de la fonction de répartition associée à la loi normale standard peuvent être obtenues à l'aide de la fonction `pnorm` :

$$\begin{aligned}\Phi(z) &= \mathbb{P}(Z \leq z) \\ &= \text{pnorm}(z)\end{aligned}$$

```
» p<-pnorm(0.5)
» p
```

Le fractile d'ordre α de la loi normale centrée réduite peut être obtenu à l'aide de la fonction `qnorm` :

$$z_\alpha = \text{qnorm}(\alpha).$$

```
» qnorm(0.6915)
» qnorm(0.5)
» qnorm(1)
```

Plus généralement, R propose les fonctions de répartition, les fonction de densité ou de masse et les inverses des fonctions de répartition pour diverses lois. Entre autres,

---

Loi	*rép.*	*inv. rép.*	*dens. ou mas.*
$\mathcal{N}(0,1)$	`pnorm`	`qnorm`	`dnorm`
$St(n)$	`pt`	`qt`	`dt`
$\chi^2(n)$	`pchisq`	`qchisq`	`dchisq`
$\mathcal{B}(n,p)$	`pbinom`	`qbinom`	`dbinom`

Ainsi qu'on l'a déjà vu, les générateurs de nombres pseudo-aléatoires correspondants sont : `rnorm`, `rt`, `rchisq`, `rbinom`,...etc.

```
» x <- 0:10
» y <- dbinom(x,10,0.5)
» plot(x,y,type="p",pch="+")

» x <- seq(0,15,0.2)
» y <- dchisq(x,4)
```

» `plot(x,y,type="l")`

ou, plus directement,

» `curve(dchisq(x,4),from=0,to=15)`

Pour d'autres possibilités, faire

» `help(stats)`

puis cliquer sur "Index".

Probabilités avec R

Evaluation d'une densité gaussienne de dimension d

La fonction de densité d'un ve.a.r. X de loi $\mathcal{N}_d(\mu, V)$ tel que $\det V \neq 0$ s'écrit

$$f(x) = \frac{1}{(2\pi)^{d/2}\sqrt{\det V}} \exp\left(-\frac{1}{2} \cdot (x-\mu)' V^{-1}(x-\mu)\right)$$

Ecrire une fonction permettant de calculer la valeur de cette fonction en n points de $\mathbb{R}^d$.

Script pdfgaussd.R

```
pdfgaussd <- function(x,mu,covmat)
# densité de la loi normale sur Rd
# x : points où l'évaluation est faite
# dim(x) : d*n
# mu : espérance
# dim(mu) : d*1
# covmat : matrice de variance-covariance
# dim(covmat) : d*d
{
d<-dim(x)[1]
n<-dim(x)[2]
y<-x-as.matrix(mu)%*%t(rep(1,n))
a<-(2*pi)^(d/2)*sqrt(det(covmat))
arg<- diag(t(y) %*% solve(covmat) %*% (y))
dens<-(1/a)*exp((-0.5)*arg)
return(as.numeric(dens)) }
```

Probabilités avec R

Visualisation d'une densité gaussienne bivariée

Utiliser *pdfgaussd.R* pour représenter la densité d'un ve.a.r. de loi $\mathcal{N}_2(\mu, V)$ en 3D, ainsi que ses courbes de niveau.
Le faire d'abord pour une matrice de variance-covariance diagonale , puis pour une matrice de variance-covariance quelconque.

Script visugaussd.R

```
# Visualisation de la densité d'une loi gaussienne bivariée

# Charge la fonction pdfgaussd()

source("pdfgaussd.R")

# Matrice de variance-covariance diagonale

pdfgauss2 <- function (u,v)
{
mu<-rep(0,2) # espérance nulle
covmat<-diag(1,nrow=2) # matrice identité
x<- rbind(u,v)
res<- pdfgaussd(x,mu,covmat)
return(res)
}
r <- seq(-4,4,0.2)
s <- r
z <- outer(r,s,pdfgauss2)

windows()#ouvre une nouvelle fenêtre graphique
split.screen(c(1,2))#divise la fenêtre en deux parties
screen(1)
persp(r,s,z,theta=30,phi=30,main="Densité")
close.screen(1)
screen(2)
image(r,s,z)
contour(r,s,z,add=TRUE)
title("Intensités et Courbes de niveau")
close.screen(2)

# Matrice de variance-covariance non diagonale

pdfgauss3 <- function (u,v)
{
mu<-rep(0,2) # espérance nulle
covmat<-cbind(c(1,0.7),c(0.7,1)) # matrice non diagonale
x<- rbind(u,v)
```

```
res<- pdfgaussd(x,mu,covmat)
return(res)
}
r <- seq(-4,4,0.2)
s <- r
z <- outer(r,s,pdfgauss3)

windows()#ouvre une nouvelle fenêtre graphique
split.screen(c(1,2))#divise la fenêtre en deux parties
screen(1)
persp(r,s,z,theta=30,phi=30,main="Densité")
close.screen(1)
screen(2)
image(r,s,z)
contour(r,s,z,add=TRUE)
title("Intensités et Courbes de niveau")
close.screen(2)
```

Probabilités avec R

Illustration du Théorème Central Limite

Théorème (Théorème Central Limite)
Soit $(X_n)_{n\geq 1}$ une suite de variables aléatoires réelles i. i. d., d'espérance μ et de variance $\sigma^2 > 0$. Définissons, pour tout $n \geq 1$, la variable centrée réduite

$$Z_n = \frac{\overline{X}_n - \mu}{\sigma/\sqrt{n}}.$$

Alors, pour tout $x \in \mathbb{R}$,

$$\mathbb{P}(Z_n \leq x) \underset{n\to\infty}{\longrightarrow} \Phi(x)$$

i.e.

$$\mathbb{P}\left(\frac{\overline{X}_n - \mu}{\sigma/\sqrt{n}} \leq x\right) \to \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-u^2/2} du,$$

où Φ désigne la fonction de répartition associée à la loi normale standard $\mathcal{N}(0,1)$.

▶ Illustrer le TCL en étudiant la loi de $\overline{X}_{30}$ lorsque $X_1, \ldots, X_{30}$ sont des v.a.r. i.i.d. de loi uniforme sur $[0,1]$.
Pour cela, simuler indépendamment n échantillons i.i.d. de taille 30 associés à la loi $\mathcal{U}([0,1])$
Calculer la moyenne empirique de chacun des échantillons

$$\overline{x}_{30,1}, \ldots \overline{x}_{30,n}$$

et les valeurs

$$z_{30,1}, \ldots, z_{30,n}$$

associées, telles que

$$z_{30,i} = \frac{\overline{x}_{30,i} - \mu}{\sigma/\sqrt{n}}$$

où $\mu = \mathbb{E}(X_1) = 0.5$, $\sigma^2 = var(X_1) = 1/12$.

Représenter l'histogramme des valeurs ainsi obtenues.

Comparer graphiquement cet histogramme à la fonction de densité d'une loi gaussienne standard.
On pourra faire en sorte que l'utilisateur puisse spécifier le nombre d'échantillons n directement à la console. Pour cela, faire en sorte que la simulation et la représentation graphique soient réalisées par une fonction. On pourra lire n à la console à l'aide de l'instruction `scan(n=1)` (ne pas faire la confusion entre `n` paramètre de `scan` et le nombre d'échantillons n).

Script tcl.R

```
# Illustration du Théorème Central Limite
tcl <- function(){
rm(list=ls())
cat("Nombre d'échantillons n?","\n")
n<-scan(n=1) # Un seul nombre à lire à la console
i<-1
x<-NULL
for (i in 1:n)
```

```
{ x<- cbind(x,runif(30))
i<-i+1 }
# n échantillons indépendants de taille 30 réal. de v.a.r. i.i.d. de loi U[0,1]
z <- (apply(x,2,mean)-0.5)*(sqrt(12*30))
# moyenne empirique centrée réduite de chaque échantillon
windows()
hist(z,freq=FALSE,col="green",xlim=c(-4,4),ylim=c(0,0.5))
curve(dnorm,from=-4,to=4,add=TRUE)
# comparaison avec densité N(0,1)
}
```

Probabilités avec R

Illustration de la Loi forte des Grands Nombres

Théorème (Loi Forte des Grands Nombres)
Soit $(X_n)_{n\geq 1}$ une suite de variables aléatoires réelles i. i. d. d'espérance finie μ, alors

$$\mathbb{P}(\overline{X}_n \to \mu) = 1.$$

▶ Illustrer la LGN Forte en étudiant la convergence simple de $\overline{X}_n$ lorsque $X_1, \ldots, X_{10000}$ est un échantillon i.i.d. associé à la loi uniforme sur $[0, 1]$.

Pour cela, simuler en une fois un échantillon i.i.d. de taille 10000 associé à la loi $\mathcal{U}([0,1])$: $x_1, \ldots, x_{10000}$.

Calculer la moyenne empirique de chacun des échantillons partiels de tailles 30 à 10000,

$$\overline{x}_{30}, \ldots, \overline{x}_{10000}$$

et les comparer graphiquement à la valeur de l'espérance $\mu = 0.5$.

Script lgn-forte.R

```
# Illustration de la Loi forte des Grands Nombres
rm(list=ls())
mx <- NULL
x <- runif(10000) # un échantillon de taille 10000 de loi uniforme sur [0,1]
for (n in 30:10000)
{ mx <- c(mx,mean(x[1:i]))# moyenne des i premières réalisations
}

windows()
plot((30 :10000),mx,pch=".",col="blue",
main="Loi forte des Grands Nombres",xlab="n", ylab="Moyenne empirique Xbar(n,omega)")
lines(c(30,10000),c(0.5,0.5),col="red")# trace une ligne horizontale entre (30,0.5) et (10000,0.5)
```

Probabilités avec R

Illustration de la Loi faible des Grands Nombres

Théorème (Loi Faible des Grands Nombres)
Soit $(X_n)_{n\geq 1}$ une suite de variables aléatoires réelles i. i. d. d'espérance finie μ, alors pour tout $\varepsilon > 0$,

$$\mathbb{P}(|\overline{X}_n - \mu| \leq \varepsilon) \to 1.$$

▶ Illustrer la LGN Faible en étudiant la variabilité de $\overline{X}_{n,n}$ lorsque $X_{1,n}, \ldots, X_{n,n}$ sont des échantillons i.i.d. indépendants associés à la loi uniforme sur $[0, 1]$ pour n variant de 30 à 10000.

Pour cela, simuler indépendamment pour n variant de 30 à 10000, des échantillons i.i.d. de taille n associé à la loi $\mathcal{U}([0, 1])$.

Calculer la moyenne empirique de chacun des échantillons

$$\overline{x}_{30,30}, \ldots, \overline{x}_{n,n}, \ldots, \overline{x}_{10000,10000}$$

et les comparer graphiquement à la valeur de l'espérance $\mu = 0.5$.

Script lgn-faible.R

```
# Illustration de la Loi faible des Grands Nombres
rm(list=ls())
mx <-NULL
for (n in 30:10000)
{ x<-runif(n) # échantillon de taille n de loi uniforme sur [0,1]
mx<-c(mx,mean(x))} # moyenne de l'échantillon x

windows()
plot(log10(30:10000),mx,pch=".",col="blue",
main="Loi faible des Grands Nombres",
xlab="n",ylab="Moyenne empirique Xbar(n,omega_n)")
lines(c(log10(30),4),c(0.5,0.5),col="red") # trace une ligne horizontale entre
# (log10(30),0.5) et (4,0.5)
```

Références

[1] Dalhousie University Dept. of Statistics & Mathematics. *Getting started using S+. 1993.*

[2] S. Lardjane. *TP de Data Mining. IUP Génie des Systèmes Industriels, Université de Bretagne Sud, 2003.*

[3] S. Lardjane. *TP de Probabilités. ENSAI, 2006.*

[4] T. Lumley. *R Fundamentals and Programming Techniques. R Core Development Team and UW Dept. of Biostatistics, 2006.*

[5] W. L. Martinez, A. R. Martinez. *Computational Statistics Handbook with MATLAB. Chapman & Hall/CRC, 2002.*

[6] M. Mokhtari, A. Mesbah. *Apprendre et Maîtriser MATLAB. Springer, 1997.*

[7] E. Paradis. *R pour les débutants. Université de Montpellier II, 2005.*

[8] P. S. Toulouse. *Thèmes de Probabilités et Statistique. Dunod, 1999.*

[9] W. N. Venables, D. M. Smith. *An Introduction to R. V. 2.7.0, 2008.*