Your Paper

You

October 6, 2025

Abstract

Your abstract.

1 Introduction

```
## http://cran.cict.fr/bin/windows/base/R-2.6.1-win32.exe
## ONLINE HELP

help.search("regression")
?lm
help(lm)
help("*")
n = 10
x = 1
x = 10
y = 10 + 2
z = 3 + rnorm(1)
w = 8; name = "Arthur"; dicton = "Vive le logiciel R";
n; x; y; z; w; name; dicton
n = 8 ; n ; mode(n)
A = "bonjour, bienvenue au premier cours de R"
A
mode(A)
length(A)

x = c(FALSE, T); x; mode(x); length(x)

A = c("bonjour, bienvenue au premier cours de R", "vous allez découvrir un univers merveilleux")
A
mode(A)
length(A)

# Creation of a character string chain
nom <- paste("filename", ".", "txt", sep = "")
nom
prefix <- "resultats."
i <- 5
nom <- paste(prefix, i, ".ps", sep = "")
nom
## Generate data
5:10
n <- 10
1:(n - 2)
1:n - 2
rep(1, times = 10)
```

```
rep(1:5, times = 2)
rep(1:5, each = 3)
seq(from = 2, to = 10, by = 2)
seq(1, 5, length = 15)
x = 1:20
x
x[x > 10] <- 20
x
x[x == 20] <- 0
x

x = c(1.1, 5.3, 9, 4.2, 3.6, 7.2, 8.4, 1.6, 8.8, 3.5)
x
x < 5

y = x[x < 5]
y

y = x[c(2, 6)]
y
seq(1, 5, length = 15)
sum(x)  # sum of elements of x
prod(x) # product of elements of x
min(x)  # minimum of elements of x
max(x)  # maximum of elements of x
which.min(x)  # index of the minimum
which.max(x)  # index of the maximum
length(x) # number of elements in x
rev(x)   # reverse the order of elements in x
sort(x)  # sort elements of x in ascending order
matrix(0, 5, 7)
x = 1:20
mat2 = matrix(x, 4, 5, byrow = TRUE)
mat3 = matrix(x, 4, 5)

mat2
mat3
nom_var = paste("V", 1:5, sep = "")
nom_ind = paste("I", 1:4, sep = "")
colnames(mat3) = nom_var
rownames(mat3) = nom_ind
dimnames(mat3) = list(nom_ind, nom_var)

mat3
## Pour les list

liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste
liste[[1]]
liste[[2]]
liste[[1]][c(TRUE, FALSE)]
liste[[2]][ ,3]

##Access the values of an object by name
Lst<-list(name="Fred", wife="Mary", no.children=3, child.ages=c(9,7,4))
Lst
```

```
Lst[[1]]
Lst$name

Lst[[4]]
Lst[[4]][3]
liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste
liste$a;liste$b
liste$b[1, ]

## ------------------------------------------------------



## CONVERSION D'OBJETS

##CONVERSION of MODE

## 1. Towards NUMERIc : as.numeric

## Logic Towards  numeric
logique = c(FALSE, FALSE, TRUE, TRUE, FALSE, TRUE)
conversion_numerique =  as.numeric(logique)

## caracter Towards  numeric
caractere1 = c("1", "2", "3", "4", "5")
conversion_numerique1 =  as.numeric(caractere1)
caractere2 = c("A", "/", "T", "%", "-")
conversion_numerique2 =  as.numeric(caractere2)
caractere1
conversion_numerique1
caractere2
conversion_numerique1
## 2. Towards  LOGIc: as.logical

## Numerique vers logique
numerique = 0:10
conversion_logique1 = as.logical(numerique)
numerique
conversion_logique1
## Caractere vers le logique

caractere = c("FALSE", "TRUE", "F", "T", "false", "t", "A", "(")
conversion_logique2 = as.logical(caractere)
caractere

## 3. Towards character: as.character

## numérique vers caractère
numerique = 1:8
conversion_caractere1 = as.character(numerique)

## logique vers caractère
logique = c(TRUE, FALSE)
conversion_caractere2 = as.character(logique)

## CONVERSION DE TYPE D'OBJETS
```

```
## conversion d'une matrice en une data.frame
a = matrix(1:25, nrow = 5, ncol = 5);a;
b = as.data.frame(a);b;

## LISTER LES OBJETS EN MEMOIRE
ls()
ls(pat = "n")
ls(pat = "^n")
ls.str()

## EFFACER LES OBJETS DE LA MEMOIRE

rm(x)
rm(n, z)
rm(list = ls())
rm(list = ls(pat = "^n"))



##Graphiques
##Fonction plot()
##Linstruction de base pour tracer des graphes est la fonction plot(). Dans sa forme la plus simple,
##de lui donner deux vecteurs de même taille pour tracer le nuage de points correspondant à chaque él
##du premier vecteur contre lélément correspondant du deuxième vecteur :
x<-pi*c(0:200)/50
y<-cos(x)
 plot(x,y)

##On peut aussi donner à la fonction une matrice, et plot() fera alors un graphe de la première colon
##axe des abscisses et de la deuxième colonne en axe des ordonnées :
 pmat<-matrix(c(x,y),ncol=2)
 plot(pmat)

##Options
##Les options les plus importantes de la fonction plot() sont données dans le tableau 3.1 :
##main="Le titre" donner un titre au graphe
##sub="Le sous-titre" donner un sous-titre au graphe (en dessous de laxe des X)
##xlab="Légende X",ylab spécifier la légende des axes
##xlim=c(xa,xb), ylim=c(ya,yb) fixer les limites des axes
##axes=T/F tracer les axes (F : ne pas les tracer)
##type="" par défaut vaut "p" pour points
##(les autres valeurs possibles sont décrites plus loin)
##log="" échelle logarithmique
##("x"=axe des X en log, "y"=axe des Y en log, "xy"=les 2)
##pch=n,lty=n change le symbole ou le type de ligne utilisé
##col=n ajoute de la couleur
##font=n,cex=n change la fonte et la taille du texte et des symboles

##Modifions notre graphe précédent pour ladapter en ajoutant des légendes, en modifiant les limites d
##axes, et en changeant certaines options :

##Graphe de cos(X) en fonction de X

 x<-pi*c(0:200)/50
 y<-cos(x)
 plot(x,y,xlab="Variable X",ylab="Cos(X)",main="Graphe de cos(X) en fonction de X",sub="intervalle 0-
```

```
Comme cest une des options les plus utiles, le graphe ci-dessousmontre les différents types de graphe
que lon peut obtenir avec loption type.
x1<-pi*c(0:20)/5
y1<-cos(x1)
par(mfrow=c(3,2))
plot(x1,y1,type="p",main="type=\"p\"")
plot(x1,y1,type="l",main="type=\"l\"")
plot(x1,y1,type="o",main="type=\"o\"")
plot(x1,y1,type="h",main="type=\"h\"")
plot(x1,y1,type="s",main="type=\"s\"")
plot(x1,y1,type="n",main="type=\"n\"")

##Ajouter des éléments à un graphe
##On peut aussi vouloir rajouter des éléments après avoir avoir tracé un premier graphe. Pour cela il
##existe un certain nombre de fonctions graphiques spécifiques, résumées dans la table 3.2.

##title(),axis() ajouter un titre ou un axe
##legend() ajouter une légende
##text() ajouter du texte
##mtext() ajouter du texte dans la marge
##box() entourer le graphe dun cadre
##segments(a,b,c,d) ajouter un trait entre (a,b) et (c,d)
##lines(x,y) ajouter un trait entre x(=c(a,b)) et y(=c(c,d))
##points(a,b) tracer un point aux coordonnées (a,b)
##arrows(a,b,c,d) tracer une flèche (a,b) et (c,d)
##rect(a,b,c,d) tracer un rectangle
##polygon(x,y) tracer un polygone
##abline() ajouter une ligne de pente et intercept spécifié (voir syntaxe)
##identify() trouver les coordonnées dun point sur une courbe
##locator() trouver les coordonnées dun point en cliquant avec la souris

##Les fonctions points() et lines() sont très utiles pour ajouter une deuxième courbe sur un graphe,
##pour superposer des courbes à des données. Elles peuvent être utilisées avec des vecteurs. Par exem
##pour superposer la courbe de y=sin(x) sur le nuage de points correspondant à y=cos(x), on peut écri

##La fonction legend(), qui permet de rajouter une légende à un graphe, est particulièrement utile. O
##peut ainsi superposer plusieurs courbes sur le même graphe, ou différencier des sous-groupes avec d
##symboles et des couleurs différentes. Le texte de la légende est obligatoire

##les options importantes de cette fonction sont les suivantes :
## lty : donne le type de la ligne à tracer pour chaque élément de la légende, soit sous forme dun se
##chiffre, soit sous forme dun vecteur de la même taille que le texte (ou -1 pour pas de ligne).
## pch : donne le type de symbole à tracer pour chaque élément de la légende (mêmes remarques que
##pour lty).
## col : donne la couleur à utiliser pour chaque élément de la légende (mêmes remarques que pour lty)

plot(x,y,xlab="Variable X",ylab="",xlim=c(0,max(x)),ylim=c(-1.5,1.5),type="o",lty=2,col=2)
lines(x,sin(x),col=1)
legend(3,1.5,c("Sin(X)","Cos(X)"),lty=1:2,col=1:2,pch=c(-1,1))

##On utilise la fonction abline() pour des lignes droites. On verra son usage pour tracer la droite d
##régression dans la section 4.2, pour lheure servons-nous en pour rajouter la ligne des abscisses. P
##on utilise loption h=0 (bien évidemment, v=0 trace une ligne verticale pour laxe des ordonnées) :

 plot(x,y,xlab="Variable X",ylab="Cos(X)")
```

```
 lines(x,sin(x))
abline(h=0)

##La fonction plot() est la fonction graphique dite "générique" de R, cest-à-dire que cest une foncti
##qui essaie de tracer le graphe le plus adapté avec le type de données quon lui donne. Il existe aus
##fonctions graphiques permettant de faire des graphes particuliers.
##hist() histogramme
##barplot() graphe en barre
##boxplot() boîte à moustache
##qqplot(), qqnorm() graphe quantile-quantile
##pie() camembert
##tsplot() graphe de séries temporelles
##pairs() corrélations entre les colonnes dune matrice
##matplot() corrélations entre les colonnes de deux matrices
##Les fonctions pairs() et matplot() sont très similaires, elles peuvent être utilisées pour détermin
##jeu de données présenté sous forme de matrice les corrélations entre les différentes variables.
##Voici un exemple dutilisation de la fonction pairs() pour tracer les corrélations entre des paramèt
##estimés par régression non linéaire individuelle chez 53 sujets :
##param<-matrix(scan("plots/parfitte.dat"),
##ncol=7,byrow=T)
nampar<-c("kmet","kaft","vdft","kel0","kinh","vdfu")
 pairs(param[,c(5:7)],labels=nampar[c(4:6)])

##Voici un exemple dutilisation de la fonction hist(). Loption breaks permet de contrôler le nombre d
##classes utilisées pour tracer lhistogramme :
xgaus<-rnorm(100)
 hist(xgaus,breaks=20)

##Voici un exemple dutilisation de la fonction boxplot(). On a utilisé loption varwidth=T pour que la
##des boîtes soit fonction du nombre dobservations dans chaque vecteur.
xgaus<-rnorm(500)
 ygaus<-runif(60)
 boxplot(xgaus,ygaus,varwidth=T)

##Montrons maintenant un exemple dutilisation de la fonction pie(). On trace un camembert où chaque
##part est égale. Notez lusage de la fonction rainbow() qui donne un vecteur contenant un nombre donn
##couleurs différentes.
 pie(rep(1,12),col=rainbow(12),radius = 0.9)

##Enfin, utilisons la fonction persp() pour faire un graphe 3D dune fonction trigonométrique.
x<-pi*c(0:20)/5
 y<-x
 f <- function(x,y) {
 r <- sqrt(x^2+y^2)
 10 * sin(r)/r}
 z <- outer(x, y, f)
 persp(x,y,z)

##Environnement graphique
par(pch=2)
par(ask=T)
##Une autre option très utile est la possibilité de superposer deux graphes
 x<-pi*c(0:200)/50
 y<-cos(x)
 z<-2*sin(x)
 plot(x,y,ylim=c(-2,2),type="l")
```

```
 par(new=T)
 plot(x,z,ylim=c(-2,2),type="l")

##Tracer plusieurs graphes différents sur la même page
##On utilise loption par(mfrow=c(m,n)) pour créer une page de m×n figures où m donne le nombre de
##rangées et n le nombre de colonnes. On peut également contrôler lapparence des graphes en demandant
##par exemple une zone de tracé carrée, en utilisant loption pty="s" de la fonction par(). Par exempl
##créer une page de 2 fois 3 graphes carrés :
par(mfrow=c(2,3),pty="s")



##Analyse statistique
#Differentes fonctions permettent de générer des nombres aléatoires suivant une certaine distribution
#de probabilité :
runif(10) # distribution uniforme
rnorm(10) # distribution normale
rnorm(10,mean=100)
rbinom(10,size=1,prob=.5) # distribution binomiale


a=rnorm(20,mean=55,sd=10)
mean(a)
sd(a)
max(a)
summary(a)
hist(a)
boxplot(a)
x1=rnorm(10,mean=100,sd=10)
x2=rnorm(10,mean=110,sd=10)
boxplot(x1,x2)
plot(x1,x2)

#La fonction rnorm génere des nombres aléatoires distribués selon une loi normale. En augmentant
#le nombre d'échantillons générés (de 10 a 10000), on constate que la distribution des valeurs
#obtenues se rapproche de plus en plus d'une distribution normale continue :
s1=rnorm(10,mean=2)
summary(s1)
s2=rnorm(100,mean=2)
summary(s2)
s3=rnorm(10000,mean=2)
summary(s3)
par(mfrow=c(3,3)) # organisation des graphiques selon une matrice 3 x 3
hist(s1) # histogrammes
hist(s2)
hist(s3)
plot(density(s1)) # fonction de densité
x=seq(-5,5,by=.01) # vecteur de coordonnees normales pour les abscisses
lines(x,dnorm(x,mean=2),col=2)
plot(density(s2))
lines(x,dnorm(x,mean=2),col=2)
plot(density(s3))
lines(x,dnorm(x,mean=2),col=2)


sink(file="C:\\Users\\Privé\\Desktop\\tp1.txt")
```

```
#Bernoulli
n=100
n
#Tirage avec remise
x=sample(c(-1,1), n, replace=T)
plot(x, type="h", main="Variable de Bernoulli")
#Avec des probabilités différentes :
x=sample(c(-1,1), n, replace=T, prob=c(.2,.8))
plot(x, type=h,main="Bernoulli, cas général")
x=rbinom(1000,10,0.5)
x
parmfow=c(2,2)
hist(x,xlim=c(min(x),max(x)), col="blue",nclass=max(x)-min(x))
#Ajouter une estimation nom-paramétrique de la densité
lines(density(x), col="red", lwd=2)
#Un noyau gaussien est utilisé par défaut pour cette estimation avec une optimisation du paramètre de
#fenétre
#Loi de Poisson
#Les memes instructions sont reprises avec la génération dune variable de Poisson de paramètre 1.
x=rpois(1000,1)
hist(x,xlim=c(min(x),max(x)), probability=T, col="blue",nclass=max(x)-min(x))#Changer le paramètre de
x=rpois(1000,20)
hist(x,xlim=c(min(x),max(x)), probability=T, col="blue",nclass=max(x)-min(x))
sink()

#Lois théoriques
#Les expressions théoriques des densités sont aussi connues de R. Voici les tracés des fonctions pour
curve(dchisq(x,1), xlim=c(0,10), ylim=c(0,.6), col="red", lwd=2)
curve(dchisq(x,2), add=T, col="green", lwd=3)
curve(dchisq(x,3), add=T, col="blue", lwd=3)
curve(dchisq(x,5), add=T, col="orange", lwd=3)
abline(h=0,lty=6)
abline(v=0,lty=3)
sink()


#Loi des grands nombres
#La moyenne empirique de n variables suivant une loi uniforme est calculée. Comparer la précision obt
mean(runif(10))
mean(runif(1000))
#Meme chose avec la loi de Cauchy pathologique :
mean(rcauchy(10))
mean(rcauchy(1000))
mean(rcauchy(100000))
# Limite centrale et loi gaussienne
#Estimer par un histogramme la densité dune variable aléatoire, somme de 12 variables uniformes sur [
x=rep(0,1000)
for (i in 1 :1000) x[i]=sum(runif(12))
hist(x, col="blue", probability=T) #Ajouter une estimation non paramétrique de la densité
lines(density(x), col="red", lwd=2) #Comparer avec la densité théorique dune loi N(6, 1)
curve(dnorm(x,mean=6,sd=1), add=T, col="green", lwd=2) #Commentaire sur la vitesse de convergence en
#moyenne empirique de n variables aléatoires i.i.d. despérance m et de variance 2 vers une gaussienne
#variance 2/n

sample(1 :20,20)
sample(1 :20, 20, replace=TRUE)
```

```
## ------------------------------------------------------

# IMPORTATION DE DONNEES

#Premier exemple : Auto original



f=file.choose()
A1 = read.table(f)

A2 = read.table(f, header = FALSE)

A4 = read.table(f, header = TRUE)
g=file.choose()

write.table(A1, g)
d=file.choose()
B = read.csv(d)

#Lecture de fichier Excel
library(xlsReadWrite)
l=file.choose()

C = read.xls(l, colNames = TRUE, sheet = 1)

write.xls(C, "D:/Documents and Settings/at215484/Bureau/Arthur/Bibliotheque/Cours R/donnees R (import
#Pour sauver les données au format R

o=file.choose()

save(A1, B, C, file = "o.Rdata")

save(list = ls(), file = "o.Rdata")




#Régression simple
# Exemple de régression linéaire avec la commande lm
# ================================================
```

```
# on stocke les données de data.txt dans lobjet « klein »

o=file.choose()

klein = read.table(o, header=TRUE)
klein                           # pour voir

summary(klein)                  # pour des détails

summary(wtot)                   # appel nominatif impossible
attach(klein)
summary(wtot)                   # wtot est à présent connu

# la commande lm

help(lm)                        # si on veut lire la documentation
lm(cons~p+plag)            # la régression

lm(cons~p+plag+wtot)            # la régression
reg = lm(cons~p+plag+wtot)
reg                             # reproduit laffichage minimal des résultats
summary(reg)                    # affiche les résultats détaillés
plot(reg)
reg$residuals                   # les résidus
reg$fitted                      # la série ajustée
plot(reg$residuals)             # on figure les résidus
lines(reg$residuals)            # idem avec la ligne qui les joint
plot(wtot, reg$residuals)       # un graphique croisé
```

Distributions conjointes Si l'on reprend l'exemple précédent des tailles de la population française masculine (20-25 ans), on a une distribution similaire (i.e. suivant une loi normale de moyenne 70 et d'écart-type 7) pour les poids. On peut bien évidemment se poser les mêmes questions que précédemment, mais on peut également s'intéresser à la relation entre ces deux variables quantitatives. En représentant le poids en fonction de la taille, on peut évaluer la liaison linéaire entre ces deux variables à l'aide du coefficient de corrélation de Bravais-Pearson.

Pour illustrer cela, nous allons utiliser les données issues d'une population d'enfants de sexe masculin âgés de 11 à 16 ans.

```
taille<-scan('')        # saisie manuelle des données
 172 155 160 142 157 142 148 180 167 165
11:
Read 10 items           # indicateur de fin d'entrée-sortie généré par R
poids<-scan('')
50.5 38.1 57.3 39.3 46.1 37.1 45.9 66.3 60 50.5
11:
Read 10 items
plot(poids~taille)
r<-lm(poids~taille)     # modèle linéaire (x,y)
summary(r)                      # diagnostic de la régression
abline(r)                       # tracé de la droite de régression
-55.1963626 + 175 * 0.6568411   # "prédiction" pour taille=175 cm
predict(r,list(taille=c(175)))
predict(r,list(taille=c(175,198)))
```

TP 1 Simulation 2

2 TP1

The ts Object

The R language includes many other objects. For example, a time series ts is an object in R. To define a univariate time series in R, four objects are necessary:

- `ts(data = NA, start = 1, end = numeric(0), frequency = 1)`
  - **Data object**: This object contains the actual data. These data can be structured as a vector, and this object is mandatory to define a time series.
  - **Start object**: This indicates the date at which the series starts. For a monthly series, `start` could be, for example, equal to "February 1990".
  - **Frequency object**: This specifies the number of observations in a unit of time. For instance, in the case of monthly data collected over several years, the unit of time is the year and the frequency of observations is 12 per year.
  - **End object**: This determines the date of the last realization of the process (start and end can be vectors of two elements).

Examples

Example 1: Simulating White Noise

```
a = ts(rnorm(100, mean = 0, sd = 1), 1980, f = 4)
plot(a)
```

Example 2

Now suppose we want to create a "time series" object that contains the data of a vector named

```
monvecteur = c(1, 2, 3, 4, 5, 6, 7, 2, 3, 4, 5, 6, 7, 8, 3, 4, 5, 6, 7, 8, 9, 4, 5, 6, 7, 8, 9, 10, 5
```

which, by construction, comprises six periods of length seven. Thus, the time series we will constitute is such that:

```
frequency = 7
```

Let's imagine that this series represents the evolution of a variable day after day, starting, say, on Wednesday of the fifth week. We would then write:

```
start = 5 + (3/7)
```

to specify the date of the first realization of the series (with 0/7 = Sunday, 1/7 = Monday, etc.). Consequently, we no longer have the choice to specify the last object, which must be:

```
end = 11 + (1/7)
```

To encode all these objects into a "time series" object called `maserie`, we use the notation:

```
maserie = ts(monvecteur, start = 5 + (2/7), frequency = 7, end = 11 + (1/7))
```

The series is then presented as follows:

```
maserie
```

Nothing has significantly changed. However, if we take a period = 12, R recognizes the data as monthly and renders

```
maserie = ts(monvecteur, start = 5 + (2/12), frequency = 12)
```

R thus arranges the ts object in the form of a table where the rows represent the unit of time (here, the week number) and the number of columns is determined by the frequency of the series. It should be noted that, although R presents the data in the form of a table, the ts object is not a matrix or table in the strict sense: the series is univariate and the reading of its data must be done from left to right, then top to bottom. Thus, if we want to know the 20th realization of `maserie`, we need to write `maserie[20]`. The object `maserie` can thus be manipulated like a vector. To define a time series, only the data object is mandatory. We could, in fact, define a time series `maserie2` as:

```
maserie2 = ts(monvecteur)
```

If nothing is specified, the other series objects take a default value as given in the following table:

Object	Default Value
start	1
frequency	1
end	length of the series

We already noted that the data and start objects determine the value of end. To avoid errors, only one of the two objects, start or end, should be fixed in the series. Thus, we could have defined

```
maserie = ts(monvecteur, start = 5 + (2/7), frequency = 7)
```

with the same result. An interesting simplification is to specify, for the start, a vector of two elements containing the time unit (in this case, the number of the first week considered) and the position of the first observation in the period. The time series can thus be defined in a new way:

```
maserie = ts(monvecteur, start = c(5, 3), frequency = 7)
```

To obtain the values of data, start, and end for a given series, one must use the `tsp` function. Thus, for `maserie`, we get:

```
tsp(maserie)
```

The last value of this vector is the number of observations made during a unit of time of the series. The first element is the start and corresponds to the Wednesday of the fifth week (with a frequency of 7 observations per year, the time between two observations is thus $1/7 = 0.1428$ weeks). Finally, the central component of `tsp(maserie)` indicates that the last observation took place on Tuesday of the eleventh week.

Autocorrelation Function

The autocorrelation function can be defined as follows:

```
acf(x, lag.max = 50, plot = TRUE, main = "Autocorrelation Function", xlab = "Time Lag")
```

The temporal autocorrelation of the series for a time lag h measures the degree of linearity of the relationship that links observations at times t and $t - h$. A positive correlation between two variables indicates a tendency for these variables to vary in the same direction (and conversely for a negative correlation). To visualize the correlogram, one can use:

```
acf(maserie) # to plot the correlogram
acf(maserie, lag.max = 10, type=c("correlation", "covariance"), plot = TRUE) # to plot the correlogra
pacf(maserie) # to plot the partial correlogram
```

3 TP 1.1

Run and Explain the Following Codes

1) Code Execution and Explanation

The following R code manipulates time series data. Let's go through each line of code step by step:

```
x = c(4, 3, 7, 1.7, 3, 8.4, 5, 12, 9, 12)
```

This creates a numeric vector x containing the values {4, 3, 7, 1.7, 3, 8.4, 5, 12, 9, 12}.

```
is.ts(x)
```

This checks whether x is a time series object. The expected output is `FALSE` since x is a regular numeric vector, not a time series.

```
x
```

This outputs the original vector x.

```
y = as.ts(x)
```

This converts the vector x into a time series object y.

```
y
```

This displays the time series object y.

```
time(y)
```

This returns the time indices of the time series y.

```
is.ts(y)
```

This checks whether y is a time series object, which will return `TRUE`.

```
t = tsp(y)
```

This retrieves the time series properties of y that indicate the start time, end time, and frequency.

```
plot.ts(y)
```

This visualizes the time series y in a default plot.

```
X11()
```

This opens a new graphics window (on systems that support X11).

```
plot.ts(y, type = "b")
```

This plots the time series y with both points and lines connecting the data points.

```
z = ts(y, freq = 4)
```

This creates a new time series object z from y with a frequency of 4 (for quarterly data).

```
z
```

This outputs the time series object z.

```
time(z)
```

This returns the time indices of the time series z.

```
z = ts(x, freq = 4, 1991 + 1/4, 1993)
```

This creates another time series object z from the original vector x with a frequency of 4, starting from the first quarter of 1992 (1991 + 1/4), and ending in 1993.

```
time(z)
```

This retrieves the time indices for the newly created time series z.

```
plot.ts(z)
```

This visualizes the new time series z.

```
frequency(z)
```

This returns the frequency of the time series z, which should be 4.

```
length(z)
```

This returns the number of observations in the time series z.

2) Code Execution and Explanation

The following R code involves simulation and manipulation of time series data. Let's go through each part step by step.

```
x = rnorm(100)
```

This generates a vector x of 100 random numbers drawn from a standard normal distribution.

```
a = ts(x, 1980, f = 4)
```

This creates a time series object a starting in the year 1980 with a frequency of 4 (indicating quarterly data).

```
a
```

This outputs the time series object a.

```
plot(a)
```

This visualizes the time series a.

```
a = ts(x, 1980, 2004.75, 4)
```

This creates a new time series object a starting in 1980 and ending in the third quarter of 2004.

```
a = ts(x, f = 4, start = c(1980, 3))
```

This creates a time series object a with frequency 4, starting in the third quarter of 1980.

```
x = rnorm(30)
```

This generates a new vector x of 30 random numbers from a standard normal distribution.

```
par(mfrow = c(2, 2))
```

This sets up a plotting area for a 2x2 grid of plots.

```
plot.ts(x)
```

This plots the time series data from vector x.

```
hist(x, nclass = 6)
```

This creates a histogram of x with 6 classes.

```
qqnorm(x)
```

This produces a quantile-quantile plot for x to assess its normality.

```
x = rbinom(30, 10, .3)
```

This generates a vector x of 30 random numbers drawn from a binomial distribution with size 10 and probability 0.3.

```
y = as.ts(x)
```

This converts the vector x into a time series object y.

```
sort(y)
```

This sorts the time series y in ascending order.

```
z = order(y)
```

This determines the order of indices that would sort the time series y.

```
plot.ts(x, z)
```

This attempts to plot the time series x against the sorted indices z.

```
data = round(rnorm(24), 4)
```

This generates 24 random normal values and rounds them to four decimal places.

```
ts(data, start = c(2008, 3), frequency = 12)
```

This creates a monthly time series from *data* starting in March 2008.

```
ts(data, start = c(2008, 3), frequency = 4)
```

This creates a quarterly time series from *data* starting in March 2008.

3) Further Time Series Examples

Observations taken every 30 seconds over 5 minutes

```
s1 <- ts(data = rnorm(11), start = 0, end = 5 * 60, frequency = 30)
```

This creates a time series $s1$ with 11 random normal observations, starting at 0 seconds and ending at 5 minutes, with observations taken every 30 seconds.

```
plot(s1)
```

This visualizes the time series $s1$.

Monthly observations starting in February 1990

```
s2 <- ts(matrix(rt(204, df = 3)), start = c(1990, 2), frequency = 12)
```

This creates a monthly time series $s2$ with 204 random observations from a t-distribution with 3 degrees of freedom.

```
s2
```

This outputs the time series object $s2$.

```
plot(s2)
```

This visualizes the time series $s2$.

Information about the Time Series

To retrieve information regarding the time series $s2$:

```
start(s2)
end(s2)
frequency(s2)
```

Extracting a Portion of the Series

There are two possibilities for subsetting a part of the series:

```
s4 <- ts(s2, start = c(1990, 4), end = c(1991, 2), freq = 12)
```

This creates a new time series $s4$, from April 1990 to February 1991.

```
s4 <- window(s2, start = c(1990, 4), end = c(1991, 2))
```

This uses the window function to extract the portion of $s2$ from April 1990 to February 1991.

```
plot(s4)
```

This visualizes the subset time series $s4$.

Data Import and Transformation

a) Data Processing Steps

1) Retrieve the data `USAccDeaths`

The file contains the monthly number of accidental deaths from 1973 to 1978. It is assumed that this file has already been imported into the R environment.

2) Re-write this file to the .tex format (deaths.dat)

We can use the command below to write the data to a new .dat file.

```
write.table(USAccDeaths, file = "deaths.dat", sep = "\t", row.names = FALSE, col.names = TRUE)
```

3) Import this file into R

To import the .dat file into R, use the following code:

```
f = file.choose()
deaths = read.table(f, header = TRUE)
```

b) Transformation and Analysis of the Data

1) Transform the object `'deaths'` into a time series

The object `deaths` can be transformed into a time series as follows:

```
deaths_ts = ts(deaths$V1, start = c(1973, 1), frequency = 12)
```

Where $\texttt{deaths}V1 correspondstothefirstcolumnofthedeathtable.$

2) Display the mode and class of this object

To display the mode and class of the object $\texttt{deaths}_{ts}$:

```
mode(deaths_ts)
class(deaths_ts)
```

3) The same with the object `'strikes'`

For the object strikes:

```
strikes_ts = ts(strikes, start = 1951)
```

Seasonal series do not require the frequency argument since they are annual by default.

4) Extract the data `'deaths'` from February 1974 to October 1974

To extract this period:

```
deaths_sub = window(deaths_ts, start = c(1974, 2), end = c(1974, 10))
```

5) Plot the graphs of `deaths` and `strikes`

To plot the graphs:

```
par(mfrow = c(2, 1))
plot(deaths_ts, main = "Accidental Deaths", ylab = "Number of Deaths")
plot(strikes_ts, main = "Strikes in the United States", ylab = "Number of Strikes")
```

6) Plot the correlogram of `deaths`

To plot the correlogram, one uses acf:

```
acf(deaths_ts)
```

7) Explain `acf(deaths, plot = FALSE)`

The command below computes the autocorrelation coefficients for the lags without plotting:

```
result = acf(deaths_ts, plot = FALSE)
```

You can then inspect result to observe the autocorrelation values for each lag.

8) Try applying the previous codes to the data `AirPassengers`

To apply the same steps:

```
data(AirPassengers)
airpassengers_ts = ts(AirPassengers, frequency = 12, start = c(1949, 1))
plot(airpassengers_ts)
```

Conclusion

Ces étapes illustrent le processus d'importation, de transformation et d'analyse des séries chronologiques en R. L'utilisation des fonctionnalités de base de R pour manipuler des données temporelles est cruciale pour toute analyse statistiques des séries chronologiques.

3.1 Exercice 1

We call Gaussian white noise a sequence of independent and identically distributed random variables with mean zero and unit variance.

1. What is the autocorrelation function of white noise?
2. Simulate a Gaussian white noise of length 100, and plot it.
3. Plot the autocorrelation function.
4. Repeat the two previous questions and observe the variability of the results. Vary the length of the series.

4 TP 1.2

Exercice 2

Simulate over 400 time steps the following series, where ϵ_t is a mean-zero Gaussian random noise with variance 1, and where ($\S_t$is a seasonal series with period 12, whose first 12 terms are drawn independently from the uniform distribution on $[0; 5]$:

1. $y_t = e_t$;
2. $y_t = S_t + e_t$;
3. $y_t = 0.3t + S_t + e_t$;
4. $y_t = t \times S_t \times e_t$;
5. $y_t = \cos\left(\frac{t}{30}\right) \times e_t$;
6. $y_t = \cos\left(\frac{t}{30}\right) + e_t$;
7. $y_t = 0.3y_{t-1} + \epsilon_t \quad (y_0 = 0)$;
8. $y_t = \epsilon_t + 0.7\epsilon_{t-1}$;
9. $y_t = 0.6y_{t-1} + 0.3y_{t-2} + \epsilon_t + 0.2\epsilon_{t-1}$.

Exercice 3

1. Varicella data Retrieve the file containing the number of varicella cases recorded in New York from January 1931 to June 1972.

   (a) Create a time series object containing this series. Graphically display the series.

   (b) Qualitatively analyze this series, i.e., identify any trends and/or seasonality.

   (c) What is the mean monthly number of varicella cases?

   (d) Plot the first 25 autocorrelations. Interpret these results. What do the horizontal dashed lines on the ACF plot represent?

   (e) On a single plot, display the monthly evolutions of varicella cases for each year.

   (f) Plot the yearly evolution of varicella cases.

   (g) Do these last two questions help you improve your conclusions from question 2?