M1 ISMAG
MISB40Y - Séries chronologiques
TP 0 - Introduction à R

---

1 Ouvrir une session R sous windows

Ouvrir une session R en cliquant sur l'icône R du bureau. Au démarrage de R, une fenêtre apparaît. La première ligne apparaissant dans la fenêtre de commandes est le répertoire de travail. En dessous de cette ligne figure le symbole >. Ce symbole signifie que R est disponible pour recevoir une commande à écrire au clavier. Ainsi, toutes les commandes devront être introduites devant ce symbole.

La rubrique d'aide fournie avec R constitue la première source de documentation sur le logiciel. Pour appeler cette rubrique, il faut écrire, dans la ligne de commande :

```
> help()
```

puis enfoncer la touche return. Une nouvelle fenêtre apparaît à l'écran contenant la rubrique d'aide. On peut manipuler cette rubrique à l'aide de la souris. Il arrive également que l'on cherche une rubrique d'aide à propos d'un élément précis. Par exemple, si on désire afficher la rubrique d'aide qui traite des séries ARIMA, il faut écrire :

```
> help(arima)
```

L'aide de R est également accessible en cliquant sur Help, puis R Help. Une autre source d'information très utile est l'aide électronique disponible en ligne. Ces ressources sont accessibles en cliquant sur Help puis, dans Online Manuals, en sélectionnant l'aide en ligne désirée.

2 Premières manipulations

Symboles de base

Les manipulations de bases en R concernent les calculs arithmétiques. Pour additionner 12 et 24 par exemple, il suffit de taper, à l'invite de la ligne de commande :

```
> 12+24
[1] 36
```

Les quatre symboles arithmétiques fondamentaux sont repris dans le tableau suivant

Opération	Symbole à utiliser
addition	+
soustraction	-
multiplication	*
division	/

Bien entendu, on peut mélanger plusieurs opérations, en employant les parenthèses () si nécessaire. En écrivant

```
> (12/8+24)*5-2
[1] 125.5
```

R commence par diviser 12 par 8, ajoute 24, multiplie le tout par 5 et soustrait 2. Le résultat

n'est pas le même que

```
> (12/(8+24))*5-2
[1] -0.125
```

Remarquons que R ne considère pas les espaces blancs qui pourraient exister entre les difiérentes opérations :

```
> 2    *    24
[1] 48
```

De plus, R ne commence son calcul que si l'opération demandée est bien complète. Ainsi, si on écrit 2*, on obtient :

```
> 2*
+
```

où le nouveau + qui apparaît à la ligne signifie que l'instruction n'est pas terminée. On ajoute alors la fin de l'instruction :

```
> 2*
+ 24
[1] 48
```

Retenons donc que, lorsque la commande est coupée, la ligne suivante apparaît avec un + au lieu de > et R attend la fin de l'instruction. On peut également assigner un nombre à une variable. Pour ce faire, on utilise la commande =. Ainsi, si on désire que a soit le nombre 2, il faut écrire :

```
> a = 2
```

Pour lire la variable a que nous avons créée, il suffit de l'écrire :

```
> a
[1] 2
```

a est à présent enregistré comme étant le nombre 2. On peut l'utiliser en le mélangeant avec les opérations fondamentales :

```
> (a*7)/9
[1] 1.555556
```

Pour modifier a, il suffit de lui assigner une autre valeur :

```
> a = 85
```

Celle-ci aura pour conséquence d'effacer l'ancienne valeur de a irrévocablement. Une variable qui, comme a, peut être assignée, transformée, utilisée dans une suite d'opérations est appelée *objet*. Les objets sont des notions fondamentales en R. Nous allons tout d'abord décrire les objets de base : les vecteurs, les matrices et les listes.

Les vecteurs

Pour créer un vecteur, il faut utiliser la commande c. De cette manière, si on veut attribuer à v le vecteur (1,2,3,...,7), on écrira :

```
> v = c(1,2,3,4,5,6,7)
```

Pour lire v, on écrit son nom dans la ligne de commande :

```
> v
[1] 1 2 3 4 5 6 7
```

Le crochet [1] signifie que "1" est la première composante du vecteur v. Pour lire cette composante, il faut écrire v[1] :

```
> v[1]
[1] 1
```

De même, pour lire la deuxième composante, il faut écrire v[2], etc. Une vecteur peut être sujet à des opérations arithmétiques :

```
> v-1
[1] 0 1 2 3 4 5
> v*2
[1] 2 4 6 8 10 12 14
```

ou des opérations "vectorielles", comme *, qui multiplie les vecteurs entre eux composante par composante :

```
> v*v
[1] 1 4 9 16 25 36 49
```

ou même c() pour créer de nouveaux vecteurs à partir d'un vecteur

```
> monvecteur = c(v,v+1,v+2,v+3,v+4,v+5)
> monvecteur
[1] 1 2 3 4 5 6 7 2 3 4 5 6 7 8 3 4 5 6 7
[20] 8 9 4 5 6 7 8 9 10 5 6 7 8 9 10 11 6 7 8
[39] 9 10 11 12
```

Un outil utile pour créer un vecteur est le symbole : (deux points). 1 :7 signifie que l'on considère le vecteur formé de tous les nombres entre 1 et 7 :

```
> 1 :7
[1] 1 2 3 4 5 6 7
```

Le tableau suivant résume toutes ses opérations et en signale d'autres : `sum`, `prod`, `min`, `max`, `length` et `rep`.

Fonction	Effet
c(v,18)	crée le vecteur (v,18) = (1,2,...,7,18)
1 :7	crée le vecteur (1,2,...,7)
*,/	multiplie ou divise les composantes entres elles
+,-	additionne ou soustrait les composantes entres elles
sum, prod	donne un nombre qui est la somme/le produit des composantes
min, max	donne un nombre qui est le minimum/maximum des composantes
length(v)	donne la longueur de v
rep(v,3)	crée le vecteur (v,v,v)

Pour conclure avec les vecteurs, ajoutons qu'un vecteur n'est pas forcément numérique. Ainsi,

pour produire le vecteur

$$"quant3812""quant3812""quant3812""2""3""4""5""6$$

on doit écrire :

```
> c(rep("quant3812",3),2 :6)
```

On pourra observer dans ce dernier exemple que les objets peuvent constituer l'argument d'autres objets, et être ainsi imbriqués les uns dans les autres. C'est là aussi une des particularités de la syntaxe utilisée par R : on dit qu'il utilise un langage orienté "objets".

Les matrices
Pour créer une matrice en R, on peut utiliser la fonction `matrix`, en spécifiant les nombre de lignes et/ou de colonnes. Par exemple :

```
> matrix(1 :16,nrow=2,ncol=8,byrow=T)
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 1 2 3 4 5 6 7 8
[2,] 9 10 11 12 13 14 15 16
```

L'argument `byrow=T` spécifie que, pour construire la matrice à partir du vecteur 1 :16, on remplit d'abord la première ligne, puis la deuxième.

Une autre façon de construire une matrice est d'utiliser les fonctions `rbind` et `cbind` qui ajoutent une ligne ou une colonne à un vecteur. Par exemple,

```
> mamatrice = rbind(v,4 :10)
> mamatrice
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
v       1    2    3    4    5    6    7
        4    5    6    7    8    9   10
```

Dans cet exemple, on constate que le nom de la première ligne de mamatrice est appelée v. Pour lire la deuxième ligne de cette matrice, on écrit :

```
> mamatrice[2,]
[1] 4 5 6 7 8 9 10
```

On peut associer `rbind` et `cbind` aux propriétés des vecteurs que nous avons mentionnées pour créer des matrices :

```
> rbind(1 :4,5 :8,9 :12)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> cbind(1 :4,5 :8,9 :12)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Le tableau suivant résume les opérations élémentaires sur les matrices :

Commande	Effet
mamatrice$[i, j]$	donne l'élément (i ; j) de mamatrice
mamatrice$[i,]$	donne la i-ème ligne de mamatrice
mamatrice$[, j]$	donne la j-ème colonne de mamatrice
mamatrice$[-i,]$	donne mamatrice sans la i-ème ligne
mamatrice$[, -j]$	donne mamatrice sans la j-ème colonne
matrice1*matrice2	donne une matrice dont chaque élément est la multiplication des éléments correspondants dans matrice1 et matrice2
matrice1 %*% matrice2	effectue le produit matriciel
t(matrice)	transposée de la matrice

Les listes
Une liste est un objet formé lui-même de différents d'objets. Ainsi, une liste peut être constituée par exemple d'un vecteur, d'une matrice, d'un nouveau vecteur, d'une chaîne de caractères, etc. Chaque objet composant la liste est identifié par un nom.

Créons par exemple la liste suivante :

```
> maliste = list("Premier"=1,"Deuxieme"=4 :7,
+ "Troisieme"=matrix(1 :12, nrow=3, ncol=4),
+ "Quatrieme"="phrase")
```

Cette liste est constituée de quatre objets (un nombre, un vecteur, une matrice et une chaîne de caractères) codées par les noms "Premier", "Deuxieme", "Troisieme" et "Quatrieme". Pour afficher cette liste à l'écran, il faut écrire le nom de la liste :

```
> maliste
$Premier :
[1] 1
$Deuxieme :
[1] 4 5 6 7
$Troisieme :
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
$Quatrieme :
[1] "phrase"
```

Une fois qu'une liste a été définie, on peut accéder à certaines de ses composantes sans devoir en afficher tous les objets. Par exemple, pour afficher l'objet appelé "Deuxieme" dans `maliste`, il suffit d'écrire `maliste$Deuxieme`. Cet objet peut également être affiché en écrivant `maliste`[[2]], qui a l'avantage de ne pas devoir connaître exactement le nom du deuxième objet de la liste. Dans le même ordre d'idée, pour afficher les trois premiers objets de la liste, il faut écrire `maliste`[1 : 3] (avec un seul crochet). Il est néanmoins possible de retrouver le nom des composantes d'une liste, sans en afficher les objets, en utilisant names() :

> names(maliste)
[1] "Premier" "Deuxieme" "Troisieme" "Quatrieme"

3 L'objet `ts`

R comporte encore bien d'autres objets. Par exemple, une série chronologique est un objet en R que nous allons à présent étudier en détail. Pour définir une série chronologique (univariée) en R, quatre objets sont nécessaires :

1. L'objet `data` contenant les données proprement dites. Ces données peuvent être par exemple mises sous la forme d'un vecteur. Cet objet est obligatoire pour définir un série chronologique.
2. L'objet `start`, qui indique la date à laquelle commence la série. Si on a une série mensuelle, `start` peut par exemple être égal à "Février 1990".
3. L'objet `frequency` qui spécifie le nombre de réalisations observées sur une unité de temps. Par exemple, dans le cas de données mensuelles prélevées sur plusieurs années, l'unité de temps est l'année et la fréquence des observations est de 12 par an.
4. L'objet `end` qui détermine la date de la dernière réalisation du processus.

Pour illustrer ces objets, supposons que l'on désire créer un objet "série chronologique" qui contiennent les données du vecteur `monvecteur` que nous avons créé ci-dessus. `monvecteur` est un vecteur de 42 éléments qui, par construction, comporte cinq périodes de longueur sept. La série chronologique que nous allons constituer est donc telle que

```
frequency = 7
```

Imaginons que cette série représente l'évolution d'une variable jour après jour, à commencer, disons, le mercredi de la cinquième semaine. On écrira alors :

```
start = 5 + (2/7)
```

pour spécifier la date de la première réalisation de la série (avec 0/7 = lundi, 1/7 = mardi, etc.). Nous n'avons donc plus le choix pour spécifier le dernier objet, qui doit être :

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

Pour encoder tous ces objets dans un objet de type "série chronologique" appelé `maserie`, on utilise la notation `ts` :

> maserie = ts(monvecteur , start = 5+(2/7), frequency = 7, end = 11+(1/7))
La série se présente alors comme suit :

```
> maserie
Series :
Start = c(5, 3)
End = c(11, 2)
Frequency = 7
 [1]  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
[26]  8  9 10  5  6  7  8  9 10 11  6  7  8  9 10 11 12
```

R arrange donc l'objet de type `ts` sous la forme d'un tableau dont les lignes représentent l'unité de temps (ici, le numéro de la semaine) et dont le nombre de colonnes est déterminé par la fréquence de la série. Il faut remarquer que, bien que R présente les données sous forme d'un

tableau, l'objet `ts` n'est pas un objet matrice ou tableau proprement dit : la série est univariée et la lecture de ses données doit se faire de gauche à droite, puis de haut en bas. Ainsi, si on veut connaître la 20-ème réalisation de `maserie`, il faut écrire `maserie`[20]. L'objet `maserie` se manipule donc comme un vecteur. Pour définir une série chronologique, seul l'objet `data` est obligatoire. On pourrait en effet définir une série chronologique `maserie2` par

> maserie2 = ts(monvecteur)

Si rien n'est spécifié, les autres objets de la série prennent une valeur par défaut qui est donnée dans le tableau suivant :

Objet	Valeur par défaut
start	1
frequency	1
end	longueur de la série

On a déjà remarqué que les objets `data` et `start` fixent la valeur de `end`. Pour éviter les erreurs, il ne faudra donc fixer qu'un des deux objets `start` ou `end` de la série. Ainsi, on aurait pu définir `maserie` par

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

avec le même résultat. Une simplification intéressante consiste à spécifier, pour `start`, un vecteur à deux éléments contenant l'unité de temps (ici, le numéro de la première semaine considérée) et la position de la première observation dans la période. `maserie` peut donc être défini d'une nouvelle manière :

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

Pour obtenir la valeur de `data`, `start` et `end` d'une série donnée, il faut faire appel à la fonction `tsp`. Ainsi, pour `maserie`, on obtient :

```
> tsp(maserie)
[1] 5.285714 11.142857 7.000000
```

La dernière valeur de ce vecteur est le nombre de réalisations observées durant une unité de temps de la série. Le premier élément est `start` et correspond au mercredi de la cinquième semaine (la fréquence étant de 7 observations sur une année, le temps écoulé entre deux observations est donc 1/7=0.1428 sem). Enfin, la composante centrale de `tsp(maserie)` indique que la dernière observation a eu lieu le mardi de la onzième semaine.

4 Simuler des processus

Dans cette section, nous expliquons comment simuler en R des processus simples tels par exemple qu'un bruit blanc. Cependant, avant de simuler ces processus, on se demandera comment un logiciel peut créer un échantillon aléatoire. La réponse à cette question passe par la notion de générateur de nombres aléatoires par le logiciel. Nous allons voir que ce générateur permet de simuler un échantillon aléatoire provenant d'une variable uniforme sur $[0, 1]$. A partir de la simulation d'une variable uniforme, il sera alors possible de générer des échantillons pour d'autres variables aléatoires continues (exponentielle, normale,...).

Simuler un échantillon aléatoire d'une variables continue
Pour obtenir une suite de nombres aléatoires, R, comme d'ailleurs un grand nombre de logiciels, possède une fonction prédéfinie. Cette fonction se présente en réalité sous la forme d'une grande suite de nombres qui, pratiquement, est semblable à un échantillon issu d'une distribution uniforme sur $[0, 1]$. Cependant, bien qu'elle soit très longue, la suite obtenue est forcément finie, et la fonction génératrice apparaît comme une fonction cyclique qui "boucle" cette série. La fonction génératrice de nombres aléatoires est donc une fonction déterministe et cyclique et les nombres générés sont dits *pseudo-aléatoires*.

L'objet `.Random.seed` est un vecteur qui indique la valeur initiale prise par la fonction génératrice. Cet objet est lui-même randomisé dans le but d'entrer dans le cycle générateur par des valeurs initiales différentes (en pratique, cet objet peut dépendre par exemple de la date et de l'heure de l'horloge de l'ordinateur, ou de l'heure d'installation d'un logiciel, etc.).

Le générateur aléatoire de nombres simule donc une variable aléatoire uniforme sur $[0, 1]$ à partir de laquelle il est possible de simuler des variables aléatoires continues. Nous allons à présent décrire certaines de ces méthodes.

La *méthode de la transformation inverse* est basée sur le théorème de la transformation inverse. Ce théorème peut être énoncé comme suit :

Théorème 4.1 *Si U est une variable uniforme sur l'intervalle $[0, 1]$ et F est une fonction de répartition continue quelconque, alors la variable aléatoire X définie par $X = F^{-1}(U)$ a pour fonction de répartition F.*

Grâce à ce résultat, nous pouvons simuler une variable aléatoire X de fonction de répartition continue F à partir d'une variable uniforme U en posant $X = F^{-1}(U)$. Par exemple, pour simuler une variable aléatoire exponentielle de moyenne 1, il suffit de prendre $F(x) = 1 - \exp\{-x\}$ et un simple calcul montre que la variable

$$F^{-1}(U) = -\ln(1 - U)$$

est de distribution exponentielle d'espérance 1. Mais, si U est uniforme sur $[0, 1]$, il en est de même pour $1 - U$ et on a donc que

$$X = -\ln U$$

est une variable aléatoire exponentielle d'espérance 1.

Pour simuler un échantillon aléatoire provenant d'une variable exponentielle avec R, il faut utiliser la fonction `rexp()` avec deux paramètres : `n`, qui est la taille de l'échantillon requis, et `rate`, qui est le taux de hasard de l'exponentielle. Dans R, simulons un tel échantillon de taille 100 et de paramètre 2 :

```
> rexp(100,2)
```

R affiche alors à l'écran les 100 valeurs de la simulation. Lors de cette simulation, R a donc créé un vecteur `.Random.seed` pour spécifier une valeur initale à introduire dans le générateur de nombres aléatoires. Le nombre obtenu provenant, pratiquement, d'une uniforme, il est transformé pour obtenir un nombre provenant d'une exponentielle aux paramètres spécifiés. R considère

alors la valeur suivante dans le générateur cyclique, et le transforme également pour obtenir une deuxième valeur provenant de la variable désirée, et ainsi de suite pour les cent valeurs. La création du vecteur `.Random.seed` dans le répertoire *_Data* peut être constatée en appelant la fonction `ls()`. L'objet `.Random.seed` apparaît et on peut lire son contenu en écrivant

```
> .Random.seed
```

Une autre possibilité pour simuler des variables aléatoires continues est d'utiliser la *méthode de rejet*. Celle-ci permet de simuler une variable aléatoire continue ayant une fonction de densité f à partir d'une autre variable simulée de densité g selon la procédure suivante :

- Étape 1 : Simuler une réalisation y provenant de la variable aléatoire Y dont la fonction de densité est g. Simuler indépendamment de Y une réalisation u provenant de la variable aléatoire U uniforme sur $[0, 1]$.
- Étape 2 : Si $u \leq f(y)/cg(y)$ pour une certaine constante c, alors on garde la réalisation y et on pose $x = y$. Sinon, on ne génère aucune valeur.

Cette procédure est répétée autant de fois que nécessaire pour arriver à la taille requise pour l'échantillon, et on montre que la variable aléatoire X générée par cette méthode a la fonction de densité f (voir Ross (1994)).

Pour simuler une variable aléatoire normale standardisée Z, on commence par observer que la fonction de densité de la valeur absolue de Z est

$$f(x) = 2(2\pi)^{-1/2} \exp\{-x^2/2\}$$

pour $0 < x < \infty$. Pour simuler $|Z|$ par la méthode de rejet, on utilise comme fonction de densité g la densité d'une variable exponentielle de moyenne 1 :

$$g(x) = \exp\{-x\}$$

pour $0 < x < \infty$, puis on montre par un simple calcul que

$$\frac{f(x)}{g(x)} = \exp\{-(x-1)^2/2\}\sqrt{2e/\pi} \leq \sqrt{2e/\pi}$$

En prenant $c = (2e/\pi)^{1/2}$ dans la méthode de rejet, on génère une variable aléatoire X de densité f en suivant l'algorithme :

- Étape 1 : Simuler deux variables aléatoires indépendantes $Y \sim \mathcal{E}(1)$ et $U \sim U[0, 1]$.
- Étape 2 : Si $U \leq \exp(-(Y-1)^2/2)$, poser $X = Y$. Sinon, on ne génère aucune valeur.

On peut à présent simuler la variable Z, où $Z = \pm X$ de manière équiprobable.

Notons que R génère directement une variable aléatoire normale (selon une autre méthode appelée *méthode des rapports*). La fonction permettant de simuler un tel échantillon en R est `rnorm`. Toutes les autres variables aléatoires continues sont générées par des techniques similaires. Le tableau suivant précise comment simuler des échantillons pour les principales d'entre elles (`n` spécifie la taille de l'échantillon) :

Variable continue à simuler	Fonction à appeler en R
Uniforme sur $[a, b]$	`runif(n,a,b)`
Exponentielle λ	`rexp(n,`λ`)`
Gamma (s,λ)	`rgamma(n,s,`λ`)`
Normale (μ,σ^2)	`rnorm(n,`μ`,`σ`)`

Remarquons que le dernier argument pour spécifier un échantillon normal est la racine carrée

de la variance et non σ^2.

Simuler un bruit blanc et le représenter graphiquement
Nous allons à présent simuler un bruit blanc gaussien de variance 2,5 à l'aide de la fonction `rnorm` et le représenter graphiquement. Pour ce faire, il faut générer, disons, 100 réalisations d'un tel processus en utilisant la fonction `rnorm(100,0,`$\sqrt{}$`(2.5))`. Cette opération produit un objet de type vecteur que nous désirons utiliser pour définir une série chronologique. Nous devons alors utiliser la fonction `ts()` pour faire de ce vecteur une série chronologique. La syntaxe de R nous permet de faire ces deux opérations en une seule ligne. Si on appelle `sim` le bruit blanc que nous désirons simuler, on écrira alors :

> sim = ts(rnorm(100,0,sqrt(2.5)))

Comme nous le verrons plus loin, R est un logiciel qui est également apprécié pour la qualité et la facilité d'utilisation de ses graphiques. La fonction générique pour réaliser un graphique est appelée `plot`. Dans le cadre de notre simulation, R met à disposition de ses utilisateurs une fonction graphique spécifique pour les séries chronologiques : `ts.plot`. Ainsi, pour représenter graphiquement le processus `sim` que nous avons créé, il suffit d'écrire :

> ts.plot(sim)

La représentation graphique de notre série apparaît à l'écran dans une nouvelle fenêtre appelée par défaut GSD2. Il est alors possible d'enregistrer ce graphique en cliquant, dans le menu *File*, sur *Save As*. Le graphique est enregistré en fichier de format `.sgr` (R Graph Sheet) lisible par R. Nous reviendrons ultérieurement sur les différents paramètres de `ts.plot` qui nous permettrons de varier le style et les options des réalisations graphiques ainsi que d'exporter les fichiers `.sgr` dans des formats standards.

5 Manipuler des données

Importation de données
Nous verrons à la Section 6 qu'il est possible de partager, d'exporter et d'importer des répertoires `_Data` par les fonctions `data.dump` et `data.restore`. Dans cette section, nous allons voir comment importer des données reçues sous la forme d'un fichier texte ou d'un tableau Excel afin de le mettre dans un répertoire de travail `_Data`. R peut importer de nombreux types de fichiers. La commande nécessaire pour importer les données est `scan`. Ainsi, pour importer un fichier dont l'adresse complète est file, il faut écrire :

> scan("file")

où, comme dans le cas de `data.restore` (cf. plus loin), l'adresse doit être placée entre guillemets et écrite avec des double barres. Cependant, cette utilisation de la fonction `scan` suppose que le fichier à importer ne comporte que des valeurs numériques séparées par un espace blanc. Bien entendu, les données ne se présentent pas toujours sous cette forme. Parfois même, des titres sont ajoutés. Pour pallier ces éventualités, `scan` possède de nombreux paramètres facultatifs dont nous allons décrire le plus important : le paramètre `what`. C'est ce paramètre qui permettra de spécifier si les valeurs à lire dans le fichier sont numériques ou si ce sont des caractères alphabétiques. Ainsi, si `what = character()` ou `what = ""`, la fonction `scan` lira les données comme des caractères :

```
> scan("file",what="")
```

Il arrive souvent que les données à importer contiennent à la fois des caractères alphabétiques qui spécifient une colonne ou une ligne d'un tableau et des valeurs numériques. Supposons par exemple que l'on veuille importer le fichier `C :\monfichier.txt` se présentant comme suit :

```
Mars   154
Avril  174
Mai    182
Juin   180
```

Lors de l'importation, on aimerait évidemment que les mois soient interprétés comme des caractères et que les chiffres soient numériques. Pour ce faire, voici comment la fonction `scan` doit être utilisée :

```
> monfichier = scan("C :\\monfichier.txt",what=list("",0))
```

Le paramètre `what` est ici une liste composée de deux types d'objets qui permettrons à `scan` d'interpréter chaque colonne du fichier (0 représentant un objet numérique). La lecture du nouvel objet `monfichier` que nous avons créé affiche le résultat suivant :

```
> monfichier
[[1]] :
[1] "Mars" "Avril" "Mai" "Juin"

[[2]] :
[1] 154 174 182 180
```

On peut à présent travailler avec les données numériques `monfichier`[[2]].

Supposons à présent que ce fichier ait été enregistré sous la forme d'un tableau Excel nommé `C :\monfichier.xls`. Pour introduire ce fichier, une procédure simple consiste à utiliser la barre d'outils. A l'aide de la souris, dans le menu *File*, aller sur *Import Data* puis sur *From File*. Sélectionner alors le fichier souhaité (éventuellement en modifiant le paramètre *Files of type*, pour préciser que le type de fichier est importé au format `.xls`). Une nouvelle fenêtre apparaît sous la forme d'un tableau. En revenant sur la ligne de commande, on peut voir par la commande `ls()` que le fichier souhaité a été introduit dans le répertoire. Pour le lire, il suffit de taper son nom :

```
> monfichier
                V1   V2
1             Mars  154
2            Avril  174
3              Mai  182
4             Juin  180
```

`monfichier` se présente donc sous la forme d'un tableau de trois colonnes dont la première indique le numéro de la ligne dans le tableau. L'enregistrement des mois et des valeurs numériques peut alors se faire en écrivant :

> monfichier$V1
[1] Mars Avril Mai Juin
> monfichier$V2
[1] 154 174 182 180

Bien entendu, il est également possible de modifier le format des données avant de les importer dans R. Excel offre par exemple la possibilité de convertir les fichiers `.xls` en des fichiers ASCII en plaçant des tabulations à la place des changements de colonne. Pour une approche détaillée des paramètres de `scan`, on consultera l'aide incluse dans R pour cette fonction.

Représentations graphiques
Un attrait particulièrement intéressant de R est la qualité et la flexibilité de ses représentations graphiques. R donne la possibilité de personnaliser ces réalisations en permettant de modifier de nombreux paramètres dont nous allons présenter les principaux dans cette section.

La fonction générique pour produire un graphique en R est `plot()`, qui fournit des résultats variables suivant les arguments placés entre parenthèses. On dit de cette fonction qu'elle est "générique", c'est-à-dire que son résultat dépend principalement de son argument.

La liste des paramètres graphiques peut être obtenue en appelant la rubrique d'aide sur le mot `par`. Si on utilise `plot()`, la syntaxe se présente comme suit :

> plot(x, paramètres)

où x est la série à représenter et `paramètres` doit être remplacé par un ou plusieurs arguments du tableau suivant :

`type = " "`	Détermine si la série est tracée point par point avec des symboles 1 (`type= "p"`), en ligne continue (`"l"`, c'est le paramètre par défaut pour la fonction `plot()`), en points reliés par des lignes qui ne touchent par les points (`"b"`), en points reliés par des lignes qui touchent les points (`"o"`), en lignes verticales (`"h"`) ou sans aucune représentation(`"n"`). Les types `"s"` et `"S"` produisent les fonctions en escalier inférieures ou supérieures à la série.
`pch = " "` ou `n`	Spécifie le type de points. Par défaut pour `plot()`, la valeur de ce paramètre est `"1"`. On peut également obtenir des points spéciaux en écrivant un nombre sans guillements, par exemple `pch = 3` indique des signes +
`xlab=" "` et `ylab=" "`	Permettent de donner une légende aux axes.
`title(" ")` et `title(sub=" ")`	Inscrivent un titre au-dessus ou en-dessous du graphe.

Il est également possible de placer plusieurs graphiques sur une feuille. Il faut pour cela définir un tableau qui accueillera un par un les graphiques réalisés. Ainsi, pour représenter sur une même feuille six graphiques selon trois lignes et deux colonnes, on a écrit la ligne suivante

> par(mfrow=c(3,2))

avant d'introduire successivement les six graphiques. Pour retourner à la configuration de un graphique par page, il faut écrire `par(mfrow=c(1,1))`.

Il existe bien d'autres paramètres graphiques. Pour les connaître, on consultera la rubrique

d'aide ou la plupart des ouvrages d'introduction sur le logiciel. En particulier, on trouvera d'autres paramètres à utiliser dans l'ouvrage de Baumgartner (1994) disponible "en ligne".

Pour représenter une série chronologique, nous utiliserons la fonction `ts.plot()` qui se manipule comme la fonction `plot()`.

Chaque fois que l'on réalise un graphique avec R, il remplace le précédent et on perd ainsi systématiquement les anciens graphiques. Il est cependant possible de conserver les graphiques obtenus de deux manières : en enregistrant le graphique ou bien en ouvrant plusieurs fenêtres graphiques.

Pour enregistrer un graphique, il faut, dans le menu *File*, cliquer sur *Save As* et ensuite placer le graphique enregistré dans un répertoire sous le nom souhaité. Le format de ce graphique est `.sgr`, lisible par R seulement. Pour enregistrer un graphique sous un autre format, il faut, dans le menu *File*, cliquer sur *Export Graph* puis enregistrer le fichier au format souhaité grâce à l'option *File name*.

Après avoir réalisé un graphique que l'on désire conserver, on peut aussi ouvrir une autre fenêtre graphique en écrivant la commande

> win.graph()

qui a pour effet de laisser l'ancien graphique et de changer de fenêtre courante.

6 Le répertoire `_Data`

Depuis que nous avons ouvert notre session R, plusieurs objets ont été créés. Ces objets ont été systématiquement enregistrés dans un répertoire appelé `_Data`, que R a créé lors de son premier démarrage. L'adresse de ce répertoire dans l'ordinateur est indiquée à la première ligne de la fenêtre de commandes. On peut vérifier, grâce à l'explorateur de Windows, que, non seulement ce dossier a été créé à l'adresse indiquée, mais qu'un autre dossier appelé `_Prefs` a également été créé. Ce dernier contient les préférences de l'utilisateur, c'est-à-dire les paramètres de configuration de travail que l'utilisateur a choisi lors de sa dernière session. En ouvrant le répertoire `_Data`, on peut voir les différents objets définis lors de la session de travail. Chacun de ses objets constitue un fichier répertorié automatiquement par R. Il est absolument déconseillé de modifier ces objets dans l'explorateur de Windows car ils sont directement classés par R et ont une structure qu'on ne peut pas toujours lire directement en utilisant un éditeur de texte. Il est néanmoins possible de lire, de modifier ou de supprimer certains objets de ce répertoire à partir de la ligne de commandes de R. Pour lire le nom des objets contenus dans le répertoire `_Data`, il faut écrire, dans la ligne de commandes (sans rien ajouter entre les parenthèses) :

> ls()

Tous les objets créés depuis le début de la session apparaissent. On peut y trouver par exemple l'objet `maliste` que nous avons créé dans la section précédente. Si on désire lire l'objet `maliste`, il suffit d'écrire son nom dans la fenêtre de commandes. Pour le modifier, on utilise les techniques que nous avons développées ci-dessus. Si nous voulons par exemple modifier le quatrième objet de `maliste` en un vecteur (0,1,2,...,10), il suffit donc d'écrire :

> maliste[[4]] = c(0 :10)

et le fichier correspondant à cet objet est automatiquement modifié dans le répertoire `_Data`. Pour supprimer un objet de ce répertoire, il faut employer la commande `rm()`. Ainsi, pour supprimer maliste, on écrira :

> rm(maliste)

et on peut vérifier par la commande `ls()` que cet objet a bien été supprimé du répertoire `_Data`. Insistons encore sur le fait que, pour supprimer un objet de ce répertoire, il est obligatoire de passer par la ligne de commande R et qu'il ne faut pas le supprimer manuellement par l'explorateur de Windows. A force d'utilisations, le répertoire `_Data` contiendra très rapidement beaucoup d'objets qu'il deviendra difficile de gérer. Pour remédier à ce problème, nous avons signalé la fonction `rm()` qui permet d'effacer des objets de ce répertoire. Il est également possible de multiplier les répertoires `_Data` et `_Prefs`. De cette façon, selon que l'on travaille sur une matière donnée, on choisira d'ouvrir R avec le répertoire `_Data` associé à cette matière. Il est également possible d'exporter un répertoire `_Data` pour le partager avec un autre utilisateur R. Supposons que l'on souhaite enregistrer le répertoire `_Data` en un fichier dont l'adresse est `C :\\R\\mondata`. La fonction à utiliser pour exporter ce répertoire est `data.dump` et la syntaxe est donnée par :

> data.dump(ls(), "C :\\R\\mondata")

où `ls()` précise qu'on exporte le répertoire `_Data` courant et en écrivant le fichier de destination entre guillemets et des double barres \\. Le fichier créé, `mondata`, est un fichier ASCII dans lequel les objets du répertoire `_Data` sont intercalés par des instructions lisibles par n'importe quelle version de R. Il peut ensuite être partagé vers d'autres utilisateurs qui, s'ils veulent l'introduire dans leur propre répertoire `_Data`, utiliserons la fonction inverse à `data.dump` :

> data.restore("C :\\R\\mondata")

7 Terminer une session

Pour terminer une session, il suffit de fermer la fenêtre principale du logiciel. Apparaît alors une nouvelle fenêtre qui reprend les objets créés ou modifiés au cours de la session. L'utilisateur peut alors choisir de les conserver tous, de n'en conserver aucun ou de ne conserver qu'une partie des objets ou des modifications d'objets en les sélectionnant. L'opération correspondante est automatiquement traduite dans le répertoire `_Data` utilisé lors de la session.

M1 ISMAG
MISB40Y - Séries chronologiques

TP 1 - Manipulation d'une série chronologique

Fonctions de base sur la série chronologique

Les données concernent le trafic voyageur de la SNCF en 2ème classe. Elles sont exprimées en millions de voyageurs kilomètres. Les observations mensuelles portent sur la période 1963-1980 (source : Gouriéroux et Monfort, 1983).

Ouvrir R en cliquant sur l'icône et copier le fichier des données 'trafic.dat' du répertoire commun Ismag. dans le répertoire courant de R (pour le connaître cliquer sur Ouvrir un fichier).

`help.start()`	Ouvrir la fenêtre d'aide
`X= scan('trafic.dat')`	Chargement des données
`is.ts(X)`	Vérifier si X est un objet R de type *time series*
`X=as.ts(X)`	Transformer X en un objet du type *time series*
`is.ts(X)`	Vérifier
`X`	Affichage de `X`
`X=ts(X,start=1963,frequency=12)`	Renouveler la série des temps `i.e. assigner un début et une fréquence`
`X`	Vérifier
`tsp(X)`	Obtenir le début, la fin et la fréquence de la série
`frequency(X)`	Fréquence de la série des temps
`plot.ts(X)`	Représenter la série
`matrix(X,12,10)`	Représentations annuelles de la série
`matrix(X,10,12)`	Comparez
`ts.plot(matrix(X,12,10))`	Graphes
`ts.plot(matrix(X,10,12))`	Comparez
`XX=window(X,c(1970,4),1980)`	Extraire une sous-série

Simulation d'un bruit blanc gaussien

`eps=rnorm(100)`	Simuler un bruit blanc gaussien de variance 1 et de longueur 100
`y=sort(eps)`	Ordonner le vecteur eps
`hist(eps,freq=FALSE)`	Tracer l'histogramme
`lines(y,dnorm(y))`	Comparer avec la densité d'une $\mathcal{N}(0,1)$
`qqnorm(eps)`	Tracer eps en fonction des quantiles d'une loi $\mathcal{N}(0,1)$
`qqline(eps)`	Tracer la droite de régression sur les données
`abline(0,1)`	Tracer la première bissectrice et commenter.
`help(abline)`	Pour avoir un descriptif de la fonction `abline`
`? abline`	Même chose

Exercice :

1. Comment générer un vecteur gaussien de moyenne m et de variance σ^2

- à partir de `eps` ?
- avec la commande `rnorm` ? Faîtes `help(norm)` pour vous aider.

2. Recommencer cette simulation mais avec un bruit blanc de longueur 1000 puis 10000. Vérifier que l'histogramme devient de plus en plus proche de la densité de la gaussienne (courbe en cloche).
   De même, vérifier que la droite de régession sur les données est de plus en plus proche de la première bissectrice.
   Pourquoi ?

Simulation d'une série chronologique

Simuler une série chronologique Y mensuelle commençant au mois de mars 1980 et finissant au mois de décembre 2000 et telle que :

$$Y_t = Z_t + S_t + \epsilon_t$$

où :
- la tendance Z_t est linéaire : $Z_t = 0,2t + 2$;
- la composante saisonnière S_t est une fonction périodique de période 12 : $S_t = \cos(2\pi(t - 1980))$;
- l'erreur ϵ_t est un bruit blanc gaussien de variance 1.

Utiliser les fonctions ci-dessus pour transformer Y en objet `time series`, renouveler la série des temps, représenter la série, superposer les séries annuelles des années 1981 à 1998 et extraire une sous série commençant en janvier 1985 et finissant en juillet 1991.

Exercice : A rendre la semaine prochaine
• Simuler une série chronologique trimestrielle débutant au deuxième trimestre 1986 et finissant au premier trimestre 2001 satisfaisant un modèle additif et ayant une tendance quadratique (c'est-à-dire du type $at^2 + bt + c$), une composante saisonnière fonction périodique de période 4 et une erreur de type bruit blanc gaussien (on choisira les différents paramètres du modèle : coeffcients de la tendance, expression de la saisonnalité, variance du bruit blanc).
• Afficher les différentes composantes.
• Tracer la tendance, la saisonnalité, le bruit ainsi que la série chronologique obtenue.
• Superposer les séries annuelles des années 1987 à 2000.
• Extraire une sous série commençant au deuxième trimestre 1989 et finissant au troisième trimestre 1991.

M1 ISMAG
MISB40Y - Séries chronologiques
TP2 - Décomposition d'une série temporelle

Elimination du saisonnier

```
X=scan('trafic.dat')
filt=rep(1/12,11)                    Définition des coefficients
filt=c(1/24,filt,1/24)               Définition de la moyenne mobile
Z=filter(X,filter=filt,sides=2)      Calcul de la série filtrée par moyenne arithmétique
                                     d'ordre 13 modifiée aux extrémités
Z=ts(Z,start=1963,frequency=12)      Renouveler la série des temps
ts.plot(X,Z)                         Superposition de la série brute et de la série filtrée
```

Estimation du saisonnier

```
S=X-Z
s=tapply(S,cycle(S),mean,na.rm=T)    Estimation des coefficients du saisonnier
s=s-mean(s)
```

Série corrigée des variations saisonnières

```
CVS=matrix(1,18,12)
for (i in 1 :18) {for (j in 1 :12) {CVS[i,j]=t(matrix(X,12,18))[i,j]-s[j]}}
                                                    Calcul de X_CVS = X − c̃
CVS=as.vector(t(CVS))
CVS=as.ts(CVS)
CVS=ts(CVS,start=1963,frequency=12)
ts.plot(X)
ts.plot(CVS)
```

Estimation de la tendance

```
y=time(CVS)
z=time(CVS)∧2
CVS.lm=lm(CVS∼ y+z)                  Estimation de la tendance : f(t) = a + bt + ct².
CVS.lm$coefficients                  Valeurs des coefficients a, b et c
ts.plot(CVS)
ts.plot(time(CVS),CVS.lm$fitted.values)
```

Prévision

```
X1=rep(1,12)
for (i in 1 :12)
{X1[i]=a+b*(1981+(i-1)/12)+c*(1981+(i-1)/12)^2+s[i]}   Prévoir le trafic voyageur pour 1981
X2=c(as.vector(X),X1)                Juxtaposition de la série
                                     chronologique (jusqu'en 1980)
                                     et de la prévision pour 1981
X2=as.ts(X2)                         Transformation en SC
X2=ts(X2,start=1963,frequency=12)    Renouvellement des temps
```

Analyse des résidus

`res=CVS-CVS.lm$fitted.values`	Définition des résidus
`res=res/sqrt(var(res))`	Définition des résidus réduits
`acf(res)`	Corrélogramme des résidus
`hist(res)`	Histogramme des résidus
`qqnorm(res)`	Comparaison des quantiles des résidus
`abline(0,1)`	et des quantiles d'une loi normale

Utilisation de la fonction `decompose` de R

La fonction `decompose` de R permet directement de décomposer, comme son nom l'indique, une série temporelle selon le modèle additif (par défaut) ou le modèle multiplicatif :

```
X.dcp= decompose(X,type="add")
plot(X.dcp)
```

L'option `type="add"` ou `"mult"` permet de spécifier si on souhaite utiliser un modèle additif ou multiplicatif. Sur la représentation graphique, quatre courbes sont représentées : de haut en bas, figurent la série initiale, la tendance, la composante saisonnière et la partie résiduelle. Ces quatres parties correspondent aux différentes composantes de l'objet ainsi créé. La décomposition repose sur l'application de moyennes mobiles dont on peut préciser le filtre éventuellement. Par défaut, une moyenne mobile symétrique est employée.

Exercice : A rendre la semaine prochaine

Simuler une série chronologique $(Y_t)_{t=1,\dots,100}$ suivant le modèle

$$Y_t = 0,01t + 1 + 2\sin(2\pi t/5) + \epsilon_t,$$

où $(\epsilon_t)_t$ est un bruit blanc gaussien de variance $1/100$.

1. Déterminer la tendance, la saisonnalité (période) de cette série chronologique. Les tracer.
2. Utiliser la méthode des moyennes mobiles ci-dessus pour éliminer la saisonnalité puis estimer les coefficients du saisonnier.
3. Utiliser une régression linéaire par moindres carrés pour estimer les coefficients de la tendance.
4. Comparer les estimateurs avec les vrais coefficients.
5. Proposer une prévision à l'horizon 3.
6. Analyser les résidus. Les représenter.
7. Appliquer la fonction `decompose` et comparer avec les vraies valeurs.

M1 ISMAG
MISB40Y - Séries chronologiques
TP3 - Lissage exponentiel - Méthode de Holt-Winters

On rappelle que la prédiction à l'horizon h par la méthode de Holt-Winters est la suivante pour le modèle additif :

$$\hat{X}_T(h) = \begin{cases} \hat{a}_T h + \hat{b}_T + \hat{S}_{T+h-P}, & \text{si} \quad 1 \leq h \leq P \\ \hat{a}_T h + \hat{b}_T + \hat{S}_{T+h-2P}, & \text{si} \quad P+1 \leq h \leq 2P \\ \dots \end{cases}$$

avec

$$\begin{cases} \hat{a}_T &= (1-\beta)\hat{a}_{T-1} + \beta\left(\hat{b}_T - \hat{b}_{T-1}\right), \\ \hat{b}_T &= \alpha\left(X_T - \hat{S}_{T-P}\right) + (1-\alpha)\left(\hat{b}_{T-1} + \hat{a}_{T-1}\right) \\ \hat{S}_T &= \gamma\left(X_T - \hat{b}_T\right) + (1-\gamma)\hat{S}_{T-P} \end{cases}$$

On doit calculer les valeurs initiales pour utiliser les formules de mise à jour ci-dessus. Si la période est de 4, on prend par exemple

$$\hat{b}_3 = M_4(3), \quad \hat{b}_4 = M_4(4) \quad \text{et} \quad \hat{a}_4 = \hat{b}_4 - \hat{b}_3$$

et

$$\begin{cases} \hat{S}_4 &= X_4 - \hat{b}_4 \\ \hat{S}_3 &= X_3 - \hat{b}_3 \\ \hat{S}_2 &= X_2 - \hat{b}_3 + \hat{a}_4 \\ \hat{S}_1 &= X_1 - \hat{b}_3 + 2\hat{a}_4 \end{cases}$$

Construction d'une fonction de Holt-Winters additive

```
hwa=function(Z,alpha,beta,gamma)
{T1=length(Z)
S=rep(0,T1)
b=rep(0,T1)
a=rep(0,T1)
b[3]=(Z[1]/2+Z[2]+Z[3]+Z[4]+Z[5]/2)/4
b[4]=(Z[2]/2+Z[3]+Z[4]+Z[5]+Z[6]/2)/4
a[4]=b[4]-b[3]
S[1]=Z[1]-b[3]+2*a[4]
S[2]=Z[2]-b[3]+a[4]
S[3]=Z[3]-b[3]
S[4]=Z[4]-b[4]
for (i in 5 :T1) {b[i]=alpha*(Z[i]-S[i-4])+(1-alpha)*(b[i-1]+a[i-1])
a[i]= beta*(b[i]-b[i-1])+(1-beta)*a[i-1]
S[i]=gamma*(Z[i]-b[i])+(1-gamma)*S[i-4]}
list(b=b,a=a,S=S)}
```

Application à la série `trafic`

On reprend les données concernant le trafic voyageur de la SNCF dans le fichier `trafic.dat` et on transforme la série mensuelle en une série trimestrielle.

`X=as.ts(X)`	
`X=ts(X,start=1963,frequency=12)`	
`Y=aggregate(X,4,mean)`	Regroupement des données en trimestres : on calcule pour chaque trimestre la moyenne (mean) pour les trois mois
`plot.ts(Y)`	Représentation de la série trimestrielle
`Y1=as.vector(Y)`	Transformation en un vecteur
`L=length(Y1)`	Longueur de Y1
`T=tsp(X)[2]`	Temps de la dernière valeur observée

- Que peut-on dire de la saisonnalité de la série Y ?

On va utiliser la fonction `hwa` pour prédire le trafic voyageur pour l'année 1981 (on déterminera tout d'abord des valeurs "judicieuses" selon le type de lissage que l'on veut faire pour les constantes de lissage α, β et γ).

`Y1.hwa=hwa(Y1,alpha,beta,gamma)`	Utilisation de la fonction `hwa`
`Y1.hwa`	Affichage du résultat
`Y1.hwa$a`	Affichage uniquement de a
`aT=Y1.hwa$a[L]`	Affectation à aT de la dernière estimation de a
`bT=Y1.hwa$a[L]`	Affectation à bT de la dernière estimation de b
`ST=Y1.hwa$S[(L-3) :L]`	Affectation à ST des quatre dernières estimations de S Pour avoir une estimation de S sur toute une période
`T.pred=seq(1981,1981+3/4,by=1/4)`	Année à prédire
`Y1.pred=aT*(T.pred-T)+bT+ST`	Prédiction pour l'année 1981

Utilisation de la fonction `HoltWinters` de R

La fonction `HoltWinters` permet, comme son nom l'indique, d'appliquer directement un lissage exponentiel de type Holt-Winters. Par défaut, c'est le modèle additif qui est utilisé. Pour choisir le modèle multiplicatif, il suffit de rajouter l'option `seasonal = "mult"`. Plus précisément, pour une série temporelle X, cette fonction permet donc de réaliser :

- un lissage exponentiel simple
  `X.hw=HoltWinters(X,alpha=`α`, beta=0, gamma=0)`
- un lissage Holt-Winters sans composante saisonnière
  `X.hw=HoltWinters(X,alpha=`α`, beta=`β`, gamma=0)`
- un lissage Holt-Winters additif
  `X.hw=HoltWinters(X,alpha=`α`, beta=`β`, gamma=`γ`, seasonal="add")`
- un lissage Holt-Winters multiplicatif
  `X.hw=HoltWinters(X,alpha=`α`, beta=`β`, gamma=`γ`, seasonal="mult")`

Reprenons l'exemple précédent :

```
Y.hw=HoltWinters(Y,seasonal="add")
par(mfrow=c(1,1))
plot(Y.hw,xlab="Années",ylab="Nombre de passagers",
+ main="Série initiale et série lissée")
```

La fonction renvoie un objet contenant plusieurs composantes dont les suivants :

`alpha`	paramètre optimal de lissage
`beta`	paramètre optimal de lissage
`gamma`	paramètre optimal de lissage
`coefficients`	coefficients (dont ceux saisonniers)
`SSE`	somme des carrés résiduels
`fitted`	valeurs ajustées par lissage de la série et sa décomposition

Lorsqu'aucune valeur n'est précisée pour les constantes de lissage, ces valeurs sont choisies automatiquement de manière optimale en minimisant l'erreur quadratique de prédiction. Il est possible de forcer la valeur des paramètres α, β et γ en entrée de la fonction. Cela permet d'effectuer d'autres types de lissage ; en particulier, on peut faire un lissage exponentiel simple si $\beta = 0$ ou bien appliquer la méthode de Holt-Winters sans composante saisonnière est appliquée si $\gamma = 0$.

Pour comparer différentes méthodes de lissages sur un jeu de données, on peut utiliser la somme des carrés résiduels (SSE). L'affichage et la visualisation des résultats peuvent être réalisés à l'aide des commandes :

`summary(X.hw)`	description de l'objet `X.hw` obtenu précédemment par la procédure `HoltWinters`
`plot(X.hw)`	représentation des valeurs observées et des valeurs lissées
`plot(X.hw$fitted[,1])`	représentation de l'ajustement de la série remis à jour à chaque observation

Il est possible d'effectuer une autre représentation graphique contenant l'ajustement par le lissage ainsi que la décompostion de la série en trois éléments (level ?, tendance et composantes saisonnières) :

```
plot(X.hw$fitted)
```

Enfin, les prévisions sont réalisées à l'aide de la fonction générique `predict`. Un intervalle de confiance (dont le fondement théorique n'a pas été étudié dans ce cours) peut être obtenu en validant à `TRUE` l'option `prediction.interval`.

```
Y.pred= predict(Y.hw,50,prediction.interval=TRUE,level=0.95)
plot(Y.hw,Y.pred)
```

Remarque : On rappelle que, pour $\gamma = 0$, la méthode de Holt-Winters est un lissage exponentiel double si $\alpha = \beta$. Malheureusement, il ne semble pas possible de forcer, dans la fonction `HoltWinters`, à ce que les valeurs de α et de β soient identiques, sans en donner les valeurs. On ne peut pas donc effectuer un lissage exponentiel double avec un paramètre de lissage optimal (uniquement avec un choix a priori du paramètre de lissage). Utiliser

la fonction `HoltWinters` dans une boucle pour déterminer le paramètre optimal pour un lissage exponentiel double.

- Récupérer les valeurs optimales des paramètres α, β et γ obtenues avec la fonction `HoltWinters` et les injecter dans la fonction `hwa`. Attention : les paramètres α, β et γ de `HoltWinters` correspondent à $\alpha = 1 - \alpha$, $\gamma = 1 - \beta$ et $\delta = 1 - \gamma$ de `hwa`.
- Comparer les résultats.

Exercice : A rendre la semaine prochaine
En vous inspirant de la fonction `hwa` ci-dessus, créer une fonction `hwm` pour la méthode multiplicative saisonnière de Holt-Winters. Simuler une série chronologique $(X2_t)_{t=1,...,200}$ suivant le modèle :

$$X2_t = (0,01(t-200)+1)2\sin(2\pi t/5) + \epsilon_t,$$

où $(\epsilon_t)_t$ est un bruit blanc gaussien de variance 1.

1. Déterminer la tendance, la saisonnalité (période) de cette série chronologique.
2. Utiliser la méthode de Holt-Winters pour prédire les dix dernières valeurs de la série $(X2_{191}, ..., X2_{200})$.
3. Comparer ces prévisions avec les vraies valeurs.