FACULTÉ POLYDISCIPLINAIRE
DE TÉTOUAN

جامعة عبد المالك السعدي
Université Abdelmalek Essaadi

الكلية المتعددة التخصصات

Rapport de thème :

Traitement des séries temporelles par le logiciel R

Réalisé par:

- CHERCHOUR Siham
- FARTAKH Naima

Encadré par:

Mr. ELMEROUANI MOHAMED

Master Spécialisé : Gestion Informatique de l'Entreprise
Année universitaire 2016-2017

Sommaire

Introduction

R est un langage de programmation pour l'analyse et la modélisation des données. R peut être utilise comme un langage oriente objet tout comme un environnement statistique dans lequel des listes d'instructions peuvent être exécutées en séquence sans l'intervention de l'utilisateur

le projet R consiste en une implémentation libre du langage S, développé depuis les années septante dans les laboratoires Bell par John Chambers et son équipe et distribué depuis 1993 sous licence commerciale exclusive par Insightful Corp. Initié dans les années nonante par Robert Gentleman et Ross Ihaka (Université d'Auckland, Nouvelle-Zélande), auxquels sont venus s'ajouter un noyau de chercheurs du monde entier en 1997, il constitue aujourd'hui un langage et un environnement de programmation intégré d'analyse statistique.

L'objectif de ce projet est de fournir un environnement interactif d'analyse de données, doté d'outils graphiques performants et permettant une adaptation aisée aux besoins des utilisateurs, depuis l'exécution de tâches routinières jusqu'au développement d'applications entières.

Le choix s'est donc porté sur un *langage fonctionnel orienté-objet*, structure alliant la facilité d'utilisation à la souplesse et la puissance de la programmation.
De plus, l'adoption d'une licence libre de type GNU/GPL (*General Public License*) a favorisé son développement et permis son port vers de nombreux systèmes informatiques (Unix, Linux, Macintosh, Windows, etc.). Projet dynamique, et en constante évolution et bénéficie de fréquentes mises à jour, disponibles gratuitement sur le site du CRAN (Comprehensive 5 Archive Network, http://cran.rproject.org/). Avant tout destiné aux scientifiques, il est aujourd'hui largement diffusé dans la communauté académique et sert de support à de nombreuses recherches et publications.

I) Qu'est ce que le logiciel R

1) Définition

R est un système d'analyse statistique et graphique développé par Ross Ihaka et Robert Gentleman. Ce logiciel constitue une alternative au logiciel S-PLUS, même si de nombreuses différences dans la conception existent. Cependant, de nombreux programmes écrits pour S-PLUS sont directement utilisables sous R.
Un point fort de R réside dans le fait que ce logiciel est distribué librement. Son installation peut être mise en œuvre à partir du site internet du Comprehensive R Archive Network (CRAN) qui d'une part met à disposition les exécutables et d'autres part donne des informations relatives à la procédure d'installation.

2) Utilisations de R

Malgré quelques critiques généralement portées, ce langage est très utilisé pour diverses raisons:

- Langage de programmation: découle du langage S et C
- Applications mathématiques (une grosse calculatrice): calcul matriciel, différentiel, intégral
- Applications statistiques et optimisation: éventail varié de possibilités
- Excellente capacité graphique: vaste éventail de graphiques
- Flexibilité: facile de créer des nouvelles fonctions, utilisation de boucles

3) Editeurs

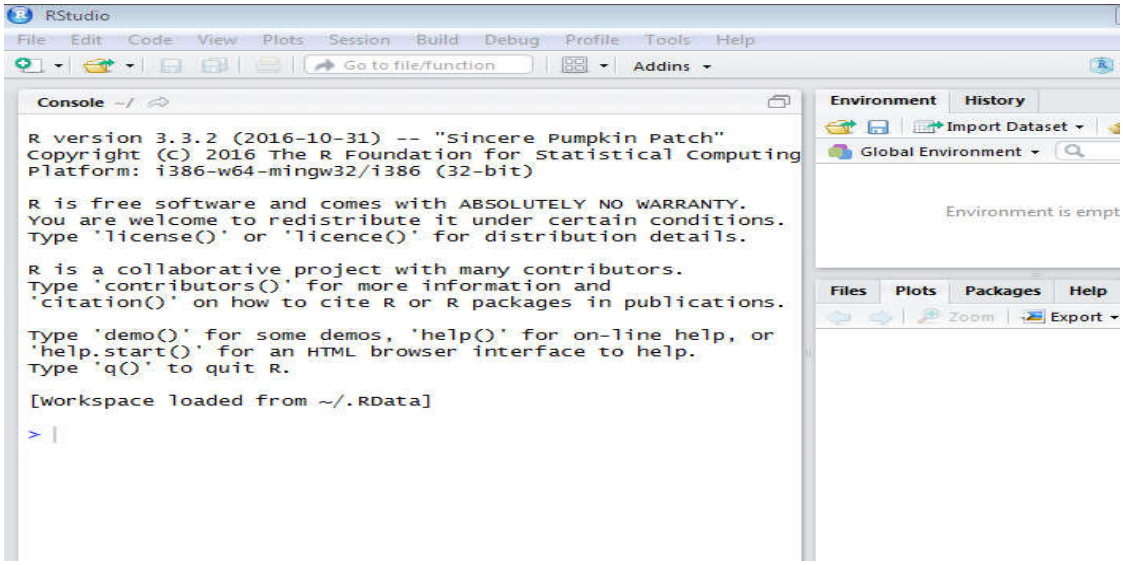
Plusieurs éditeurs intelligents disponibles pour R (Emacs, Tinn-R, Jedit)

Les caractéristiques d'un éditeur intelligent:

- reconnaît le langage R
- utilise des couleurs pour distinguer différents éléments apparie les parenthèses
- envoie les commandes directement à R
- peut aussi numéroter les lignes, compléter les commandes

**** Éditeur suggéré:** RStudio

- disponible au www.rstudio.org
- plusieurs options intéressantes
- fonctionne sur plusieurs plateformes (Windows, Mac, Linux)
- organise le tout en 4 panneaux (éditeur, console R, environnement de travail, graphiques/aide)



4) Les objets essentiels : vecteurs, matrices, listes et data.frame

Les vecteurs et les matrices sont les objets de base dans R.

4-1) vecteur

On peut créer manuellement un vecteur a l'aide de la commande c (elt1, elt2, ...).

Les composantes elt1, elt2, ... du vecteur peuvent être numériques (réelles ou complexes), logiques (TRUE, FALSE) ou alphanumériques (chaines de caractères).

```
>
> a= c (1,-2,1.35)
> a
[1]  1.00 -2.00  1.35
>
```

→ On crée un vecteur à qui a 3 éléments.

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

→ On extrait le second élément, puis les 1er et 3eme

```
b=a[c(1,3)]
```

→ Ce nouveau vecteur est appelé b

```
>
> log(a)
[1] 0.0000000       NaN 0.3001046
Warning message:
In log(a) : NaNs produced
>
```

→ L'application de la fonction log au vecteur a (c-`a-d `a toutes ses composantes) renvoie un message d'avertissement ! !

```
>
> cos(a)
[1]  0.5403023 -0.4161468  0.2190067
> exp(a)
[1] 2.7182818 0.1353353 3.8574255
> a-2*a
[1] -1.00  2.00 -1.35
> 1/a^2
[1] 1.0000000 0.2500000 0.5486968
>
```

→ Les fonctions mathématiques classiques (abs, sqrt, cos, sin, tan, exp, log, log10, asin, acos, ^ , ...) sont présentes.

Quelques fonctions utiles sur les vecteurs

Les vecteurs crées a,A,B, ...sont stockes dans la mémoire vive de l'ordinateur. Le logiciel peut parfois nécessiter une mémoire vive importante (2 Gigas voire plus) lorsqu'on manipule de très gros objets.

```
>
> sort(a)
[1] -2.00  1.00  1.35
> order(a)
[1] 2 1 3
> a[order(a)]
[1] -2.00  1.00  1.35
>
```

→ La fonction sort trie les éléments de a dans l'ordre croissant, la fonction order indique l'ordre des éléments.

```
>
> A = seq(-1,1,length=3)
> A
[1] -1  0  1
> A = seq(0,1,length=4)
> A
[1] 0.0000000 0.3333333 0.6666667 1.0000000
>
```

→ La fonction seq crée une suite de 3 points equidistants entre -1 et 1, puis de 4 points entre 0 et 1.

```
>
> a[c(1,3,4)]
[1] -1.0000000  0.3333333  1.0000000
> b= c(1:3,7)
> b
[1] 1 2 3 7
> a*b
[1] -1.0000000 -0.6666667  1.0000000  7.0000000
> 3*a[2]
[1] -1
>
```

→ On peut afficher certains éléments (1,3,4) du vecteur a

→ On crée ensuite automatiquement un vecteur B et on effectue une multiplication

4-2) matrice

Les matrices sont également les objets de base pour R. On peut effectuer sur les matrices de nombreuses manipulations de manière très simple

```
>
> A0 = matrix(c(1:6),ncol=2)
> A0
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
>
```

→ La fonction matrix crée une matrice a 2 colonnes en "empilant" les éléments (1, 2,...,6) en "colonnes".

```
>
> A = matrix(c(1:6),ncol=2,byrow=TRUE)
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
>
```

→ Pour la matrice A, l'option byrow=TRUE empile les éléments dans le sens des lignes.

```
>
> A[1,2]
[1] 2
> A[1,]
[1] 1 2
> A[,1]
[1] 1 3 5
>
>
```

```
>
> dim(A)
[1] 3 2
> t(A)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> C = diag(c(1,2)) ### matrice diagonale
> C
     [,1] [,2]
[1,]    1    0
[2,]    0    2
>
```

→ La fonction dim renvoie le vecteur qui contient le nombre de lignes et de colonnes de la matrice.
La transposée est obtenue avec t(A). Il est également possible de créer directement des matrices diagonales.

4-3) les listes

Les listes sont des collections d'objets (vecteurs, matrices, ...) qui ne sont pas nécessairement du même type. Elles sont utilisées par de nombreuses fonctions pour retourner les résultats et permettent en particulier de stocker et manipuler simplement des objets de longueurs différentes dans une même structure.

```
>
> Malist = list(nom=c("Truc","Machin"),y=c(1,2,5),M=c(TRUE
,FALSE))
>
> Malist$y
[1] 1 2 5
> Malist[[1]] ## = Malist$nom
[1] "Truc"   "Machin"
> Malist$M ## =Malist[[3]]
[1]  TRUE FALSE
```

→ On crée une liste avec la fonction list
Chaque élément de la liste peut être appelé soit en utilisant
• son nom précédé de $.
•Son indice entre double crochet [[.]]

4-4) data. Frame

Les data. Frame se présentent sous la forme d'une matrice dont les colonnes peuvent être associées a des objets de différents modes (numeric, character, ...). Ils constituent une classe particulière de listes ou chaque élément de la liste a la même longueur et est associe a une colonne.

Ce format est bien adapte au stockage de données statistiques :

- individus en lignes
- variables (quantitatives et qualitatives) en colonnes. Chaque colonne a un nom (liste).

On peut saisir un data frame à l'aide de la fonction **data.frame ().** On peut ajouter des colonnes avec **cbind(),** des lignes avec **rbind()** ou par d'autres moyens. Tout ceci est illustré par l'exemple suivant :

```
p1.palmitic <- c(1075, 1088,911,966,1051) # création colonne

> oleic <- c(7823, 7709, NA, 7952, 7771) # création colonne

> jeu <- data.frame(p1.palmitic,oleic) #création data frame

> jeu

> l1.linoleic <- c(672,781,63,619,671) # création colonne

> jeu <- cbind(jeu,l1.linoleic) # ajout colonne meth 1

> jeu

> jeu$arachidic <- c(60,61,63,78,NA) # ajout colonne meth 2

> jeu

> jeu <- rbind(jeu,c(911,NA,678,70) # ajout ligne

> jeu

> jeu[nrow(jeu)+1,] <- c(922,7990,618,56) # ajout ligne

> jeu

> jeu <- cbind(jeu, eicosenoic = c(29,NA,29,35,46,44,29)) # ajout colonne meth 3
```

II) les séries temporelles avec R

1) Création de séries temporelles

La classe de base fournie par R pour représenter des séries temporelles s'appelle Ts (abréviation de l'anglais time series). Cette classe est définie dans le package stats. Elle concerne des séries temporelles qui sont échantillonnées à des périodes équidistantes dans le temps. Un objet de classe ts possède trois paramètres caractéristiques :

– le paramètre **frequency** désigne le nombre d'observations par unité de temps. Si l'unité de temps de la série est l'année, la valeur 4 correspond à des trimestres et la valeur 12 à des mois ;

– le paramètre **start** désigne la date de début de la série temporelle. Elle est exprimée comme un nombre unique ou comme un vecteur de deux entiers qui représentent respectivement une unité temporelle (comme une année) et une subdivision de cette unité (comme un mois ou un trimestre selon la valeur du paramètre frequency) ;

– le paramètre **end** désigne la date de fin de la série temporelle. Sa valeur est exprimée comme pour le paramètre start

Par exemple, si le paramètre start vaut c (2013 ,2), il s'agit du mois de février 2013 si la fréquence est égale à 12, du deuxième trimestre de 2013 si la fréquence est égale à 4 ou du deuxième semestre si la fréquence vaut 2.

Les objets de classe ts peuvent être créés au moyen de la fonction de même nom ts . Celle-ci prend en argument un vecteur numérique et possède des arguments frequency, start et end. Si la date de début et la fréquence sont spécifiées, il n'est pas nécessaire d'indiquer la date de fin : elle est calculée automatiquement en fonction de la longueur du vecteur.

Voici un exemple de série avec une fréquence trimestrielle :

```
>
> stemp <- ts(1:10, frequency = 4, start = c(2013, 2))
> stemp
     Qtr1 Qtr2 Qtr3 Qtr4
2013         1    2    3
2014    4    5    6    7
2015    8    9   10
```

La fonction générique **print** a une méthode particulière pour les objets de classe Ts comme on peut le voir sur la manière dont les observations sont affichées dans l'exemple précédent. C'est en réalité la méthode **print.ts** qui est appelée dans ce cas. Elle possède un argument **calendar** qui prend une valeur logique (TRUE ou FALSE) : s'il est égal à TRUE, les valeurs sont arrangées sous forme d'un tableau dont les lignes représentent les unités de temps et les colonnes les subdivisions correspondant à la fréquence. Autrement, les valeurs sont représentées séquentiellement comme

un vecteur. Au lieu de la fréquence, on peut utiliser l'argument optionnel **deltat** qui exprime la fraction de l'unité de temps entre deux observations, comme par exemple 1/7 pour des observations quotidiennes lorsque l'unité de temps est la semaine.

Séries multiples

Le package stats définit aussi une notion de série temporelle multiple. Ce sont des objets de classe mts (abréviation de multiple time series) qui représentent simultanément plusieurs séries temporelles dont les observations correspondent au même découpage du temps : elles ont les mêmes paramètres **start**, end et **frequency**.

Le jeu de données appelé EuStockMarkets est un exemple de série temporelle multiple. Il comporte 4 séries temporelles représentant l'évolution simultanée de 4 indices boursiers. Les séries multiples sont aussi créées avec la fonction ts vue précédemment : il suffit de passer en premier argument une matrice au lieu d'un vecteur. Chaque colonne de la matrice représente une série temporelle. L'argument optionnel Names permet de donner un nom à chaque série : sa valeur est un vecteur de chaînes de caractères. Par exemple:

```
> mult <- ts(cbind(1:10,10:1), frequency=7, start=c(5, 1),names=c("
S1","S2"))
>
> print(mult,calendar=TRUE)
     S1 S2
5 1   1 10
5 2   2  9
5 3   3  8
5 4   4  7
5 5   5  6
5 6   6  5
5 7   7  4
6 1   8  3
6 2   9  2
6 3  10  1
```

2) Manipulation des séries temporelles

On dispose, dans le package stats, d'un certain nombre de fonctions utilitaires pour manipuler les séries temporelles.

Les fonctions **start** , **end** ,**frequency** et **deltat** renvoient respectivement la date de la première et de la dernière observation, la fréquence et l'intervalle de temps entre deux observations successives. La fonction **tsp** permet d'obtenir ou de modifier l'attribut tsp d'une série temporelle.
La fonction **cycle** renvoie un vecteur qui indique pour chaque observation quelle est sa position dans le cycle défini par la fréquence. Par exemple, en reprenant la série « **Stemp** » définie dans la section précédente :

```
>
> cycle(stemp)
     Qtr1 Qtr2 Qtr3 Qtr4
2013         2    3    4
2014    1    2    3    4
2015    1    2    3
```

La valeur de retour de la fonction cycle est un objet de classe ts (même dans le cas d'une série multiple). Elle est souvent utilisée sous forme de facteur pour calculer des statistiques pour chaque période des cycles. Par exemple, l'instruction suivante calcule les sommes des valeurs observées trimestre par trimestre :

```
>
> tapply(stemp, cycle(stemp), sum)
 1  2  3  4
12 15 18 10
```

La fonction time crée un vecteur de valeurs numériques correspondant aux temps de chaque observation de la série. La valeur de retour prend aussi la forme d'un objet de classe ts. Par exemple :

```
> time(stemp)
        Qtr1    Qtr2    Qtr3    Qtr4
2013         2013.25 2013.50 2013.75
2014 2014.00 2014.25 2014.50 2014.75
2015 2015.00 2015.25 2015.50
```

La fonction **lag** permet de décaler les dates de début et de fin d'une série. Si on spécifie un argument positif pour le décalage, celui se produit vers le passé. Par exemple, l'instruction suivante décale la série stemp en remontant de deux périodes :

```
> print(lag(stemp, 2),calendar=FALSE)
Time Series:
Start = c(2012, 4)
End = c(2015, 1)
Frequency = 4
 [1]  1  2  3  4  5  6  7  8  9 10
```

La date de départ est maintenant c (2012, 4) au lieu c(2013, 2).

La fonction **window** permet d'extraire une portion d'une série temporelle. Elle possède des arguments **start** et **end** pour indiquer les dates de début et de fin de la série extraite. On peut aussi utiliser l'argument optionnel **frequency** pour réétalonner la nouvelle série selon une fréquence différente. L'argument optionnel **extend** prend une valeur logique (TRUE ou FALSE) : il autorise l'extension d'une série temporelle à des dates qui ne figurent pas dans la série initiale.

Voici un exemple d'application avec le jeu de données Nile qui indique le débit annuel du Nil à Ashwan de 1871 à 1970. On extrait ici les valeurs de la période 1960 à 1970 et on la prolonge jusqu'à 1975 (les cinq années ajoutées reçoivent la valeur NA) :

```
> subNil <- window(Nile,start=1960, end=1975, extend=TRUE)
> subNil
Time Series:
Start = 1960
End = 1975
Frequency = 1
 [1]  815 1020  906  901 1170  912  746  919  718  714  740   NA
[13]   NA   NA   NA   NA
```

Auto-corrélation

La fonction **acf** calcule les auto-covariances, les coefficients d'auto-corrélation ou les coefficients d'auto-corrélation partielle, pour différents décalages. L'argument type permet de spécifier lesquelles de ces trois quantités on souhaite calculer : les valeurs possibles sont respectivement "covariance","correlation"ou"partial". Du point de vue théorique, cela suppose que la série observée peut être considérée comme stationnaire.

Le nombre de valeurs à calculer (autrement dit le nombre de décalages) est indiqué au moyen de l'argument lag.max. Si on ne le précise pas, il est choisi par défaut égal à$10 \times \log_{10}(N)$ où N est la longueur de la série et est limité de toute façon à N 1.

Si l'argument plot est fixé à la valeur TRUE (ce qui est le cas par défaut), la commande produit aussi une représentation graphique (appelée habituellement corrélogramme) des valeurs calculées en fonction du décalage.

Par exemple, si on calcule les coefficients d'auto-corrélation ρ_k, le graphe est un diagramme en bâtons avec les valeurs de k en abscisses et les valeurs de ρ_k en ordonnées. Dans ce cas-là, les valeurs sont renvoyées de manière invisible et il faut placer la commande entre parenthèses pour les afficher.

À titre d'exemple, reprenons le jeu de données ''USAccDeaths'' utilisé précédemment. On calcule les coefficients d'auto-corrélation comme ceci (le résultat est reproduit ici partiellement) :

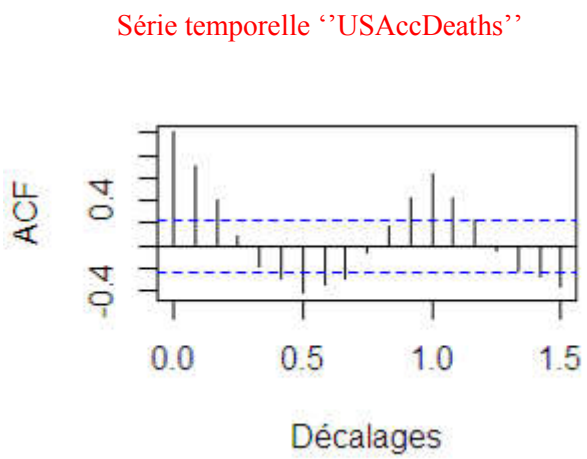
```
>  coeff <- acf(USAccDeaths, type="correlation", plot=FALSE)
> coeff

Autocorrelations of series 'USAccDeaths', by lag

0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667
 1.000  0.707  0.409  0.084 -0.182 -0.294 -0.423 -0.346 -0.285
0.7500 0.8333 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167
-0.065  0.162  0.414  0.629  0.429  0.221 -0.038 -0.216 -0.278
1.5000
-0.362
```

Il faut lire ce résultat en regardant les valeurs 2 par 2 verticalement : la valeur supérieure représente le décalage et la valeur inférieure le coefficient. Ici, le coefficient pour le décalage 0.0833 (c'est-à-dire 1/12) vaut 0.707. L'objet renvoyé dans la variable coeff est de classe acf. C'est une liste qui contient en particulier des composantes lag,acf et type. Par exemple :

```
> as.vector(coeff$acf)
 [1]  1.00000000  0.70747095  0.40859237  0.08350945
 [5] -0.18166719 -0.29443556 -0.42310357 -0.34606117
 [9] -0.28464031 -0.06472498  0.16242916  0.41398669
[13]  0.62858918  0.42851500  0.22121445 -0.03823558
[17] -0.21635478 -0.27752726 -0.36249779
```

La fonction générique plot peut s'appliquer aux objets de classe acf :

```
>
> plot(coeff, main="Série temporelle 'USAccDeaths'", xlab="Décalages")
```



3) Lissage des séries temporelles par moyenne mobile

Le lissage des séries temporelles est une décomposition de la série en une combinaison de trois termes représentant la tendance, les variations saisonnières et un terme d'erreurs ou de perturbations. Il existe de nombreux modèles et algorithmes différents pour effectuer ce genre de décomposition.

Nous allons focaliser sur la méthode la plus utilisée, le lissage par moyenne mobile :

La fonction **decompose** définie dans le package stats implémente le lissage par moyenne mobile qui cherche à représenter la série Yt sous la forme : Yt=Tt+St+et
Où Tt représente la tendance, St la saisonnalité et et les résidus. Cette fonction ne s'applique pas aux séries dont la fréquence est égale à 1 : il faut nécessairement que l'unité de temps soit subdivisée si on veut pouvoir dégager des effets saisonniers.

La tendance est calculée en appliquant un filtre mobile qui est une combinaison linéaire de coefficients successifs de la série.

Voici un exemple utilisant le jeu de données de R appelé AirPassengers qui comporte des valeurs mensuelles sur plusieurs années successives de 1949 à 1960.

```
> dec <- decompose(AirPassengers)
> class(dec)
[1] "decomposed.ts"
```

Elle renvoie un objet de classe **decomposed.ts** qui comporte plusieurs éléments d'intérêt : en particulier, les éléments trend, seasonal et randomont une structure de série temporelle et correspondent aux termes Tt,St et et du modèle respectivement. Le terme seasonal est cyclique et se reproduit identiquement pour chaque unité de temps (les années dans le cas de notre exemple).
L'élément figure de la valeur de retour contient les valeurs saisonnières sur une seule période (au lieu de les répéter en boucle sur toute la longueur de la série).
On a ici 12 valeurs saisonnières puisque la fréquence de la série est égale à 12

```
> dec$figure
 [1] -24.748737 -36.188131  -2.241162  -8.036616  -4.506313
 [6]  35.402778  63.830808  62.823232  16.520202 -20.642677
[11] -53.593434 -28.619949
```

Un argument optionnel **type** permet de spécifier le type de modèle souhaité : les valeurs possibles sont "additive" ou "multiplicative". Le modèle multiplicatif correspond à une décomposition de la forme Yt=Tt ×St+ et.

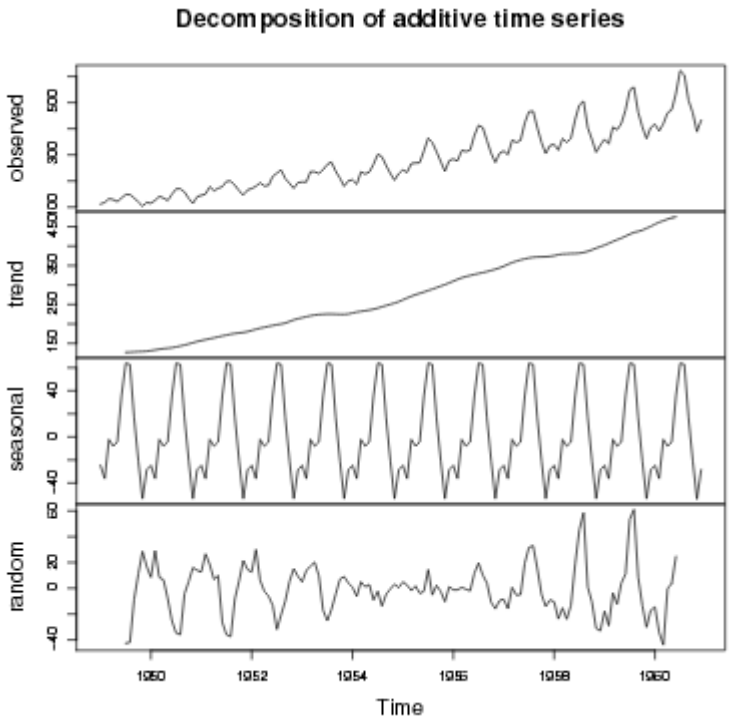
L'argument **filter** permet d'indiquer la combinaison linéaire à utiliser pour calculer la tendance. Sa valeur est un vecteur représentant les coefficients de la combinaison linéaire en sens inverse de l'ordre chronologique. Par exemple, si l'argument filter vaut c (1, 2,3), les combinaisons opérées seront de la forme 3Yt + 2Yt+1+Yt+2. Par défaut, lorsque cet argument n'est pas spécifié, le filtre consiste à faire la moyenne des valeurs sur une fenêtre symétrique centrée en chaque terme de la série initiale et couvrant une unité de temps complète. Il existe une fonction appelée **filter** qui effectue justement le filtrage par combinaison linéaire qui vient d'être expliqué. Elle renvoie la série temporelle filtrée. En voici un exemple simple :

```
> filter(1:10, c(1/2,1,1/2))
Time Series:
Start = 1
End = 10
Frequency = 1
 [1] NA  4  6  8 10 12 14 16 18 NA
```

Ici chaque terme de la séquence initiale (les nombres de 1 à 10) est remplacé par sa somme avec la moitié du terme qui le précède et la moitié du terme qui le suit. Cette formule ne peut évidemment pas s'appliquer au premier et au dernier terme puisqu'ils n'ont pas de prédécesseur ou de successeur respectivement, ce que R indique par les deux valeurs
NA obtenues.

La fonction générique **plot** s'applique aux objets de classe decomposed.ts et produit un graphe qui représente séparément la série elle-même, la tendance, les variations saisonnières et les résidus. Le graphe de la figure est obtenu avec l'instruction suivante :

```
plot (dec)
```



4) Packages relatifs aux séries temporelles

La page suivante, disponible sur les sites CRAN (Comprehensive R Archive Network), maintient une liste très complète des packages externes implémentant des méthodes relatives à l'analyse des séries temporelles : http://cran.r project.org/web/views/TimeSeries.html
Les sections qui suivent donnent une description succincte de quelques-uns d'entre eux afin de servir de référence.

- **Le package tseries**

Ce package permet de manipuler des séries temporelles dont les indices sont irrégulièrement espacés dans le temps. La fonction **irts** est utilisée pour créer ce type d'objets et il y a de nombreuses fonctions et méthodes qui leur sont associées. Le package définit aussi des fonctions arma et garch pour ajuster des séries sur des modèles de type ARMA ou GARCH et manipuler les résultats.

- **Le package forecast**

Ce package implémente de nombreuses fonctions qui calculent des modèles classiques de processus stochastiques tels que AR, ARIMA, ARFIMA, BATS, TBATS. Il supporte aussi les méthodes de lissage exponentiel et de Holt-Winters.

- **La fonction générique forecast**

Peut être appliquée à la plupart des objets créés pour effectuer des prédictions. Quelques fonctions permettent d'autre part de simuler des séries obéissant à ces modèles : **simulate.ets**, **simulate.ar**, **simulate.Arima**, **simulate.fracdiff**.

- **Le package TSA**

Ce package est lié à un ouvrage classique d'analyse des séries temporelles appelé Time Series Analysis). Il fournit des implémentations pour les modèles ARIMA, ARIMAX, QAR (Quadratic Auto-Regressive) et TAR (Threshold Auto-Regressive). Il possède des fonctions pour simuler des échantillons selon les modèles GARCH, QAR et TAR : garch.sim, qar.sim et tar.sim

- **Le package tsDyn**

Ce package implémente des modèles auto-régressifs non-linéaires désignés par les acronymes suivants :

TAR Threshold Auto-Regressive

STAR Smooth Transition Auto-Regressive

LSTAR Logistic Smooth Transition Auto-Regressive

AAR Additive Auto-Regressive

NNET Neural Network Auto-Regressive

SETAR Self Exciting Threshold Auto-Regressive

Une vignette accompagne le package et donne des exemples d'utilisation de ces différents modèles :
> vignette("tsDyn")

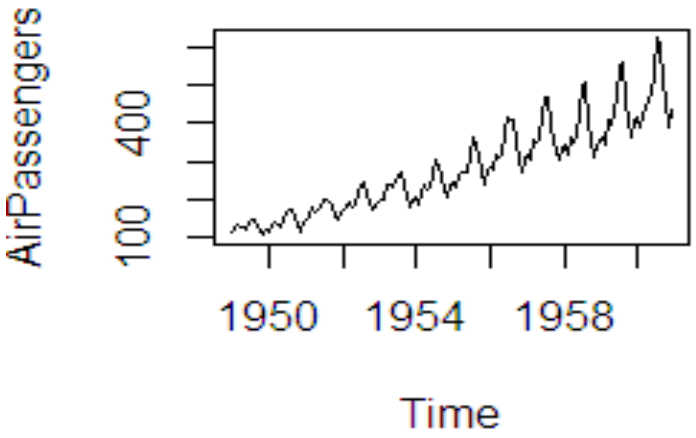
II) Cas pratiques

Nous étudions la série chronologique du nombre de passagers par mois (en milliers) dans les transports aériens, de 1949 à 1960.cette série est disponible sur R (**AirPassengers**).

1) La lecture des données

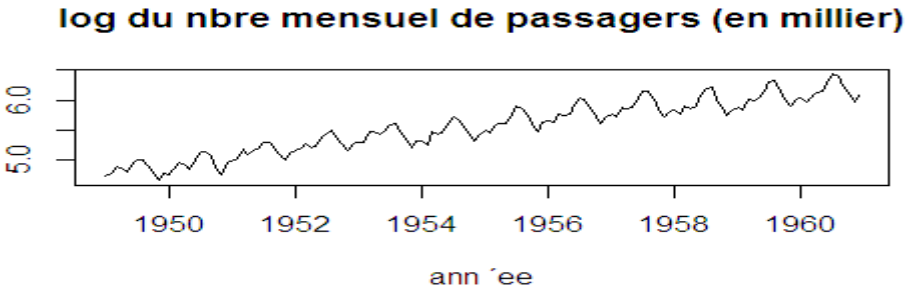
```
> help(AirPassengers)
> data=log(AirPassengers)
> data=ts(data,frequency=12,start=c(1949,1))
>
```

```
> plot(AirPassengers)
```

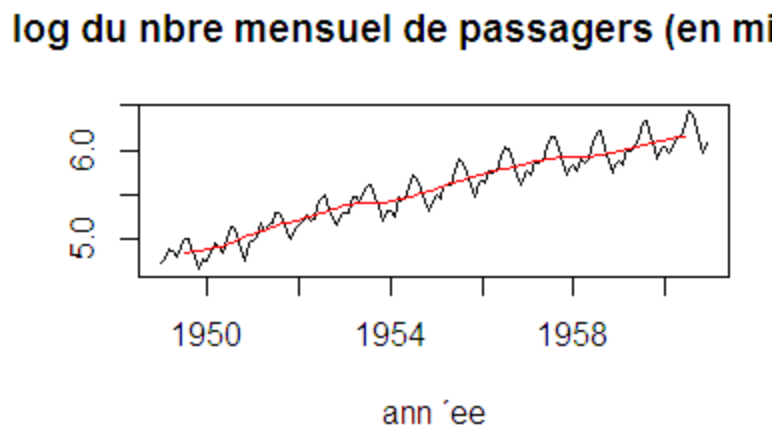


2) Représentation de la tendance

```
>
> dataMOVA=filter(data,filter=rep(1/12,12), sides=2)
>
> dataMOVA=filter(data,filter=rep(1/12, 12))
> dataMOVA=filter(data,filter=c(1/24,rep(1/12,11),1/24))
>
```



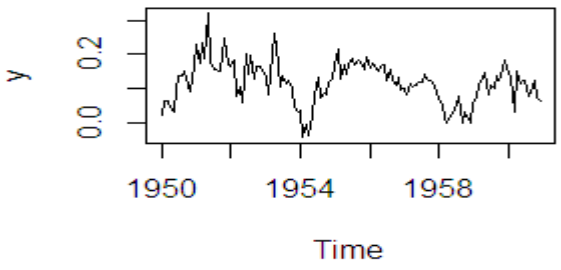
```
>
> lines(dataMOVA,col="red")
>
```



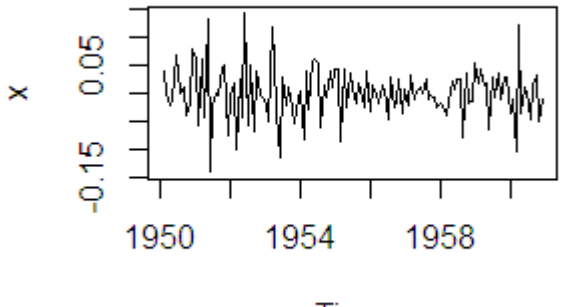

```
>
> y=as.vector(data)
>
>

>
> x=as.vector(time(data))
>
<


>
> reg=lm(y~poly(x,3))
>
>
>
> lines(predict(reg)~x)
```

3) Stationnarité

```
> y=diff(data,lag=12,differences=1)
> x=diff(y,lag=1,differences=1)
> plot(y)
```



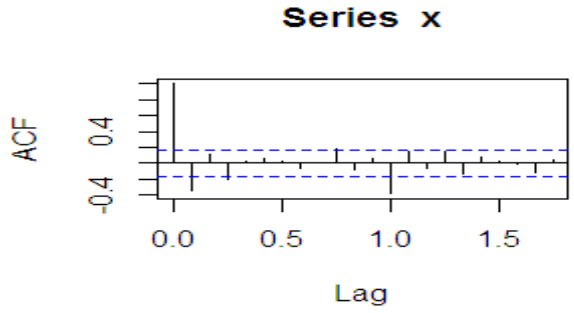
Plot(x)




```
>
> mean(x)
[1] 0.0002908799

> library (tseries)

     'tseries' version: 0.10-37

     'tseries' is a package for time series
     analysis and computational finance.

     See 'library(help="tseries")' for
     details.

> adf.test(data, alternative=c("stationary"),12)

         Augmented Dickey-Fuller Test

data:  data
Dickey-Fuller = -1.5325, Lag order = 12,
p-value = 0.7711
alternative hypothesis: stationary

  .


> adf.test(x, alternative=c("stationary"),12)

        Augmented Dickey-Fuller Test

data:  x
Dickey-Fuller = -4.413, Lag order = 12,
p-value = 0.01
alternative hypothesis: stationary

>
> adf.test(y, alternative=c("stationary"),12)

         Augmented Dickey-Fuller Test

data:  y
Dickey-Fuller = -3.137, Lag order = 12,
p-value = 0.1037
alternative hypothesis: stationary
```

4) Auto-corrélations

acf(x)



5) modèle ARIMA

```
>
> arima(data, order=c(3,1,1), seasonal=list(order=c(1,1,1), period=12))

Call:
arima(x = data, order = c(3, 1, 1), seasonal = list(order = c(1, 1, 1),
 period = 12))

Coefficients:
         ar1     ar2      ar3      ma1     sar1
      0.2366  0.1065  -0.0887  -0.6326  -0.0613
s.e.  0.3202  0.1523   0.1070   0.3115   0.1572
         sma1
      -0.5256
s.e.   0.1366

sigma^2 estimated as 0.001317:  log likelihood = 246.09,  aic = -478.18

>
> arima(data, order=c(0,1,1), seasonal=list(order=c(0,1,1), period=12))

Call:
arima(x = data, order = c(0, 1, 1), seasonal = list(order = c(0, 1, 1),
 period = 12))

Coefficients:
          ma1     sma1
      -0.4018  -0.5569
s.e.   0.0896   0.0731

sigma^2 estimated as 0.001348:  log likelihood = 244.7,  aic = -483.4
>

>
> arima(data, order=c(1,1,1), seasonal=list(order=c(1,1,1), period=12))

Call:
arima(x = data, order = c(1, 1, 1), seasonal = list(order = c(1, 1, 1),
 period = 12))

Coefficients:
         ar1      ma1    sar1     sma1
      0.1667  -0.5615  -0.099  -0.4973
s.e.  0.2459   0.2116   0.154   0.1360

sigma^2 estimated as 0.001336:  log likelihood = 245.16,  aic = -480.31
>
```

Bibliographie

- http://www.cef-cfr.ca

- http://math.unice.fr/

- INTRODUCTION À L'ENVIRONNEMENT DE PROGRAMMATION STATISTIQUE R Y. BROSTAUX

- Cours de Statistiques et Économétrie ‘’ UNIVERSITÉ PARIS OUEST NANTERRE LA DÉFENSE’’

- http://www.math-evry.cnrs.fr