Université Abdelmalek Essaadi
Faculté Polydisciplinaire de
Tétouan

Manipulation des séries temporelles dans le logiciel R

Réalisé par :

ASSIDAH BOUCHRA

ABBADI SARA

LAAFIA NIDAE

Encadré par :

Mr. MOHAMMED El MEROUANI

Sommaire

Introduction

I-Partie Théorique

1-Principales fonctionnalités

2-Qualités du R

3-Défauts de R

4-Comment R travaille?

5-Editeur de code R

6-L'espace de travail R

7-Packages R

8-Quelques grandes fonctions de R

II-Partie Pratique

1-Analyse du processus AR

2-Analyse du processus MA

3-Analyse du processus ARMA

Conclusion

Introduction

R est un logiciel statistique, ou plus exactement un langage. C'est un logiciel libre, développé et maintenu par la communauté des participants au sein du projet R et proposé pour les trois systèmes d'exploitation : Unix/Linux, Windows et MacOS. C'est enfin un logiciel modulaire : de nombreux packages complémentaires offrent une grande variété de procédures mathématiques ou statistiques, incluant des méthodes de représentations graphiques complexes, le traitement des séries chronologiques, l'analyse des données, etc.

I-Partie théorique

1-Principales fonctionnalités

1. Gestionnaire de données

   - Lecture, manipulation, stockage.

2. Algèbre linéaire

   - Opérations classiques sur vecteurs, tableaux et matrices

3. Statistiques et analyse de données

   - Dispose d'un grand nombre de méthodes d'analyse de données (des plus anciennes et aux plus récentes).

4. Moteur de sorties graphiques

   - Sorties écran ou fichier

5. Système de modules

   - Alimenté par la communauté (+ de 2000 extensions !)

6. Interface « facile » avec C/C++, Fortran.

2-Qualités du R

- Libre et open source.
- Richesse des modules (en statistique).
- Rapidité d'exécution.
- Développement rapide (langage de scripts).
- Syntaxe intuitive et compact.
- Nombreuses possibilités graphiques.

3-Défauts de R

- Debugger un peu sec.

- Code parfois illisible (compacité).
- Personnalisation des graphiques un peu lourde.
- Aide intégrée succincte.

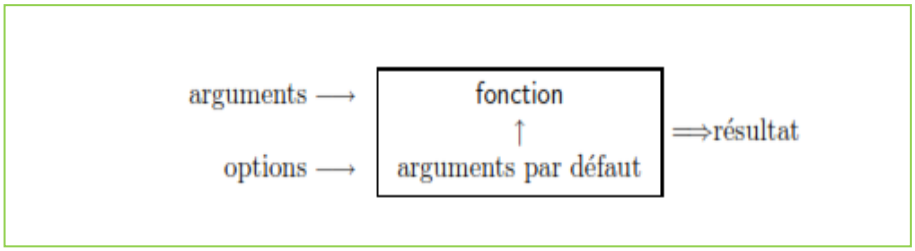
4-Comment R travaille?

Le fait que R soit un langage peut effrayer plus d'un utilisateur potentiel pensant « Je ne sais pas programmer ». Cela ne devrait pas être le cas pour deux raisons. D'abord, R est un langage interprété et non compilé, c'est-a-dire que les commandes tapées au clavier sont directement exécutées sans qu'il soit besoin de construire un programme complet comme cela est le cas pour la plupart des langages informatiques (C, Fortran, Pascal, . . .).

Ensuite, la syntaxe de R est très simple et intuitive. Par exemple, une régression linéaire pourra être faite avec la commande lm(y ~ x). Avec R, une fonction, pour être exécutée, s'écrit toujours avec des parenthèses, même si elles ne contiennent rien (par exemple ls()).

Si l'utilisateur tape le nom de la fonction sans parenthèses, R affichera le contenu des instructions de cette fonction. Dans la suite de ce document, les noms des fonctions sont généralement écrits avec des parenthèses pour les distinguer des autres objets sauf si le texte indique clairement qu'il s'agit d'une fonction. Quand R est utilisé, les variables, les données, les fonctions, les résultats, etc., sont stockée dans la mémoire de l'ordinateur sous forme d'objets qui ont chacun un nom. L'utilisateur va agir sur ces objets avec des operateurs (arithmétiques, logiques, de comparaison, . . .) et des fonctions (qui sont elles mêmes des objets).

Une fonction de R peut être schématisée comme suit :



Les arguments peuvent être des objets (« données », formules, expressions,) dont certains peuvent être définis par défaut dans la fonction ; ces valeurs par d´défaut peuvent être modifiées par l'utilisateur avec les options.

Une fonction de R peut ne nécessiter aucun argument de la part de l'utilisateur : soit tous les arguments sont définis par défaut (et peuvent être changées avec les options), ou soit aucun argument n'est défini.

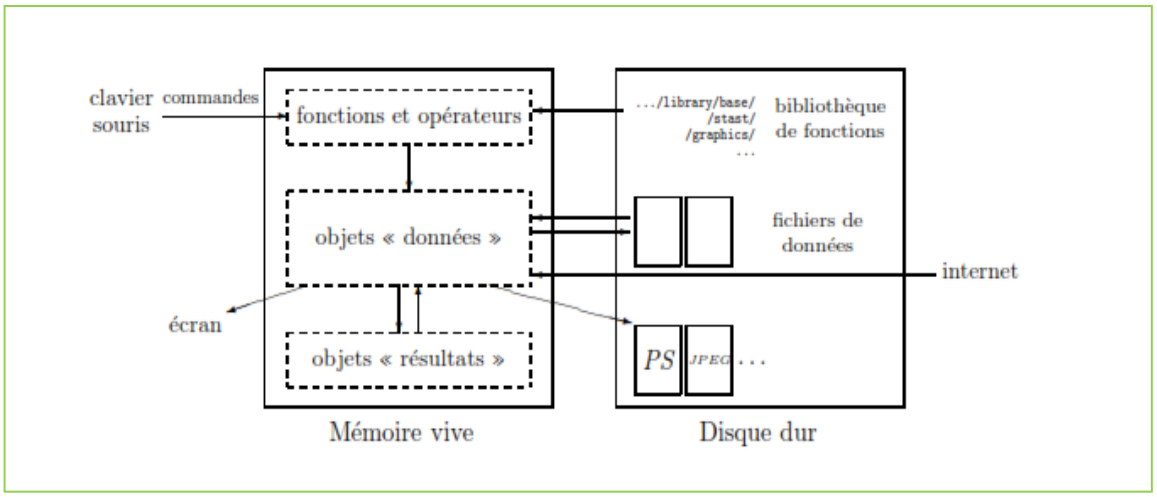
Toutes les actions de R sont effectuées sur les objets présents dans la mémoire vive de l'ordinateur : aucun fichier temporaire n'est utilisé.

Les lectures et écritures de fichiers sont utilisées pour la lecture et l'enregistrement des données et des résultats (graphiques, . . .). L'utilisateur exécute des fonctions par l'intermédiaire de commandes. Les résultats sont affiches directement a l´écran, ou stockes dans un objet, ou encore écrits sur le disque (en particulier pour les graphiques). Les résultats étant eux-mêmes des objets, ils peuvent être considérées comme des données et être analysé à leur tour. Les fichiers de données peuvent être lus sur le disque de l'ordinateur local ou sur un serveur distant via internet.

Les fonctions disponibles sont stockées dans une bibliothèque localisées sur le disque dans le répertoire R HOME/Library (R HOME désignant le répertoire ou R est installé. Ce répertoire contient des packages de fonctions, eux-mêmes présents sur le disque sous forme de répertoires.

Le package nommé base est en quelque sorte le cœur de R et contient les fonctions de base.

Du langage, en particulier pour la lecture et la manipulation des données. Chaque package a un répertoire nomme R avec un fichier qui a pour nom celui du package (par exemple, pour base, ce sera le fichier R HOME/Library/base/R/base). Ce fichier contient les fonctions du package.



Une vue schématique du fonctionnement de R.

5-Editeur de code R

RStudio est un environnement de développement multiplateforme pour R, un langage de programmation utilisé pour le traitement de données et l'analyse statistique.

Il est disponible sous la licence libre AGPLv3, ou bien sous une licence commerciale, soumise à un abonnement annuel.

RStudio est livré en deux versions : RStudio Desktop, pour une exécution locale du logiciel comme tout autre application, et RStudio Server qui, lancé sur un serveur Linux, permet d'accéder à RStudio par un navigateur web.

Des distributions de RStudio Desktop sont disponibles pour Microsoft Windows, OS X et GNU/Linux.

RStudio a été écrit en langage C++, et son interface graphique utilise l'interface de programmation Qt.

6-L'espace de travail R

L'espace de travail est votre environnement R de travail actuel et comprend tous les objets définis par l'utilisateur. À la fin d'une session R, l'utilisateur peut

enregistrer une image de l'espace de travail courant qui est automatiquement rechargé la prochaine fois que R est démarré.

Répertoire courant :

```
getwd()  # Affiche le répertoire de travail actuel
ls()  # liste les objets définis dans l'espace de travail actuel
setwd('monRepertoire') #change le répertoire courant
dir() #Lis le contenu du répertoire courant
```

Historique :

```
history() # Affiche les 25 dernières commandes utilisées
history(max.show=Inf) # Affiche toutes les commandes utilisées
savehistory(file="myfile.Rhistory") # Sauvegarde l'historique
```

7-Packages R

Les packages sont des collections de fonctions R. Le répertoire où les packages sont stockés est appelée Library. R est livré avec un ensemble standard de packages. D'autres sont disponibles pour le téléchargement et l'installation.

Une fois installés, ils doivent être chargés dans la session pour être utilisés.

Le tableau suivant liste les packages standards distribuées avec une installation de base de R.

Package	Description
base	fonctions de base de R
datasets	jeux de données de base
grDevices	périphériques graphiques pour modes base et grid
graphics	graphiques base
grid	graphiques grid
methods	définition des méthodes et classes pour les objets R ainsi que des utilitaires pour la programmation
splines	régression et classes utilisant les représentations polynomiales
stats	fonctions statistiques
stats4	fonctions statistiques utilisant les classes S4
tcltk	fonctions pour utiliser les éléments de l'interface graphique Tcl/Tk
tools	utilitaires pour le développement de package et l'administration
utils	fonctions utilitaires de R

Les packages standards

Les packages recommandés sont souvent distribues avec une installation de base de R. Ils sont brièvement décrits dans le tableau ci-dessous.

Package	Description
boot	méthodes de ré-échantillonnage et de bootstrap
class	méthodes de classification
cluster	méthodes d'aggrégation
foreign	fonctions pour importer des données enregistrés sous divers formats (S3, Stata, SAS, Minitab, SPSS, Epi Info)
KernSmooth	méthodes pour le calcul de fonctions de densité (y compris bivariées)
lattice	graphiques Lattice (Trellis)
MASS	contient de nombreuses fonctions, utilitaires et jeux de données accompagnant le livre « Modern Applied Statistics with S » par Venables & Ripley
mgcv	modèles additifs généralisés
nlme	modèles linéaires ou non-linéaires à effets mixtes
nnet	réseaux neuronaux et modèles log-linéaires multinomiaux
rpart	méthodes de partitionnement récursif
spatial	analyses spatiales (« kriging », covariance spatiale, ...)
survival	analyses de survie

Les packages recommandés

8-Quelques grandes fonctions de R

Voici un aperçu des fonctions graphiques principales de R.

`plot(x)`	graphe des valeurs de `x` (sur l'axe des y) ordonnées sur l'axe des x
`plot(x, y)`	graphe bivarié de `x` (sur l'axe des x) et `y` (sur l'axe des y)
`sunflowerplot(x, y)`	idem que `plot()` mais les points superposés sont dessinés en forme de fleurs dont le nombre de pétales représente le nombre de points
`pie(x)`	graphe en camembert
`boxplot(x)`	graphe boîtes et moustaches
`stripchart(x)`	graphe des valeurs de `x` sur une ligne (une alternative à `boxplot()` pour des petits échantillons)
`coplot(x~y | z)`	graphe bivarié de `x` et `y` pour chaque valeur (ou intervalle de valeurs) de `z`
`interaction.plot (f1, f2, y)`	si `f1` et `f2` sont des facteurs, graphe des moyennes de `y` (sur l'axe des y) en fonction des valeurs de `f1` (sur l'axe des x) et de `f2` (différentes courbes) ; l'option `fun` permet de choisir la statistique résumée de `y` (par défaut `fun=mean`)
`matplot(x,y)`	graphe bivarié de la $1^{\text{ère}}$ colonne de `x` contre la $1^{\text{ère}}$ de `y`, la $2^{\text{ème}}$ de `x` contre la $2^{\text{ème}}$ de `y`, etc.
`dotchart(x)`	si `x` est un tableau de données, dessine un graphe de Cleveland (graphes superposés ligne par ligne et colonne par colonne)
`fourfoldplot(x)`	visualise, avec des quarts de cercles, l'association entre deux variables dichotomiques pour différentes populations (`x` doit être un tableau avec `dim=c(2, 2, k)` ou une matrice avec `dim=c(2, 2)` si $k = 1$)
`assocplot(x)`	graphe de Cohen–Friendly indiquant les déviations de l'hypothèse d'indépendance des lignes et des colonnes dans un tableau de contingence à deux dimensions
`mosaicplot(x)`	graphe en 'mosaïque' des résidus d'une régression log-linéaire sur une table de contingence
`pairs(x)`	si `x` est une matrice ou un tableau de données, dessine tous les graphes bivariés entre les colonnes de `x`
`plot.ts(x)`	si `x` est un objet de classe `"ts"`, graphe de `x` en fonction du temps, `x` peut être multivarié mais les séries doivent avoir les mêmes fréquence et dates

Fonctions graphiques principales de R

`points(x, y)`	ajoute des points (l'option `type=` peut être utilisée)
`lines(x, y)`	idem mais avec des lignes
`text(x, y, labels, ...)`	ajoute le texte spécifié par `labels` au coordonnées `(x,y)` ; un usage typique sera : `plot(x, y, type="n") ; text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	ajoute le texte spécifié par `text` dans la marge spécifiée par `side` (*cf.* `axis()` plus bas) ; `line` spécifie la ligne à partir du cadre de traçage
`segments(x0, y0, x1, y1)`	trace des lignes des points `(x0,y0)` aux points `(x1,y1)`
`arrows(x0, y0, x1, y1, angle=30, code=2)`	idem avec des flèches aux points `(x0,y0)` si `code=2`, aux points `(x1,y1)` si `code=1`, ou aux deux si `code=3` ; `angle` contrôle l'angle de la pointe par rapport à l'axe
`abline(a,b)`	trace une ligne de pente `b` et ordonnée à l'origine `a`
`abline(h=y)`	trace une ligne horizontale sur l'ordonnée `y`
`abline(v=x)`	trace une ligne verticale sur l'abcisse `x`
`abline(lm.obj)`	trace la droite de régression donnée par `lm.obj` (*cf.* section 5)
`rect(x1, y1, x2, y2)`	trace un rectangle délimité à gauche par `x1`, à droite par `x2`, en bas par `y1` et en haut par `y2`
`polygon(x, y)`	trace un polygone reliant les points dont les coordonnées sont données par `x` et `y`
`legend(x, y, legend)`	ajoute la légende au point de coordonnées `(x,y)` avec les symboles donnés par `legend`
`title()`	ajoute un titre et optionnellement un sous-titre
`axis(side, vect)`	ajoute un axe en bas (`side=1`), à gauche (2), en haut (3) ou à droite (4) ; `vect` (optionnel) indique les abcisses (ou ordonnées) où les graduations seront tracées
`box()`	ajoute un cadre autour du graphe
`rug(x)`	dessine les données `x` sur l'axe des x sous forme de petits traits verticaux
`locator(n, type="n", ...)`	retourne les coordonnées (x, y) après que l'utilisateur ait cliqué `n` fois sur le graphe avec la souris ; également trace des symboles (`type="p"`) ou des lignes (`type="l"`) en fonction de paramètres graphiques optionnels (...) ; par défaut ne trace rien (`type="n"`)

Fonctions graphiques secondaires

II-Partie Pratique

1-Analyse du processus AR

Ces processus à tout instant t peuvent être extrapolés linéairement à partir de p valeurs précédents Xt − 1, ..., Xt − p à un bruit blanc près.

X est dit autorégressif d'ordre p, si : $X_t = \epsilon_t + \sum_{j=1}^{p} a_j X_{t-j},$

Pour simuler un processus AR(1) :

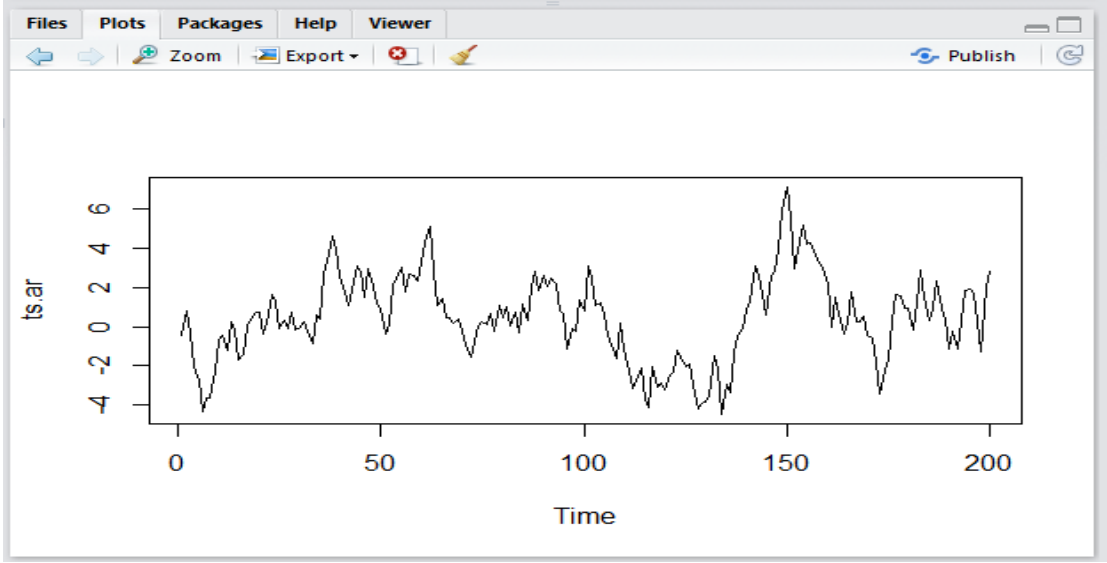
yt = 0.9 y t − 1 + ε t

déclaration du processus:

```
>ts.ar<- arima.sim(list(order = c(1,0,0), ar =0.9), n = 200)
```

Présentation graphique

```
>ts.plot(ts.ar,type="l")
```



Test de la stationnarité

adf.test(ts.ar, alternative=c("stationary"), k=0)

```
> adf.test(ts.ar, alternative=c("stationary"), k=0)

        Augmented Dickey-Fuller Test

data:  ts.ar
Dickey-Fuller = -2.7534, Lag order = 0, p-value = 0.2606
alternative hypothesis: stationary

>
```

Test de l'inversibilité

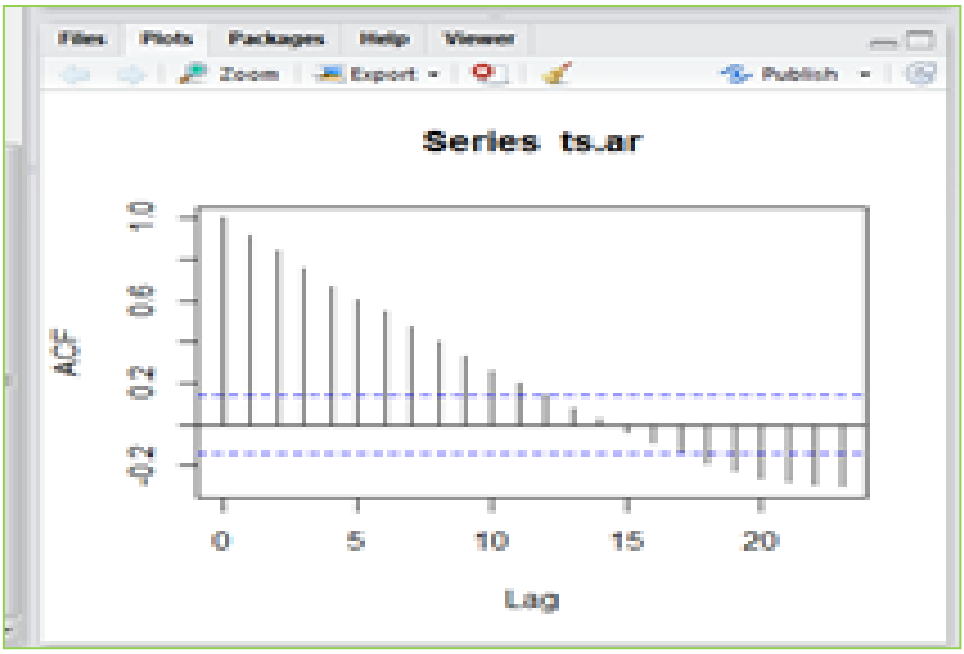
adf.test(ts.ar, alternative=c("explosive"), k=0)

```
> adf.test(ts.ar, alternative=c("explosive"), k=0)

        Augmented Dickey-Fuller Test

data:  ts.ar
Dickey-Fuller = -2.7534, Lag order = 0, p-value = 0.7394
alternative hypothesis: explosive

>
```

Les auto-covariances :

(acf(ts.ar))

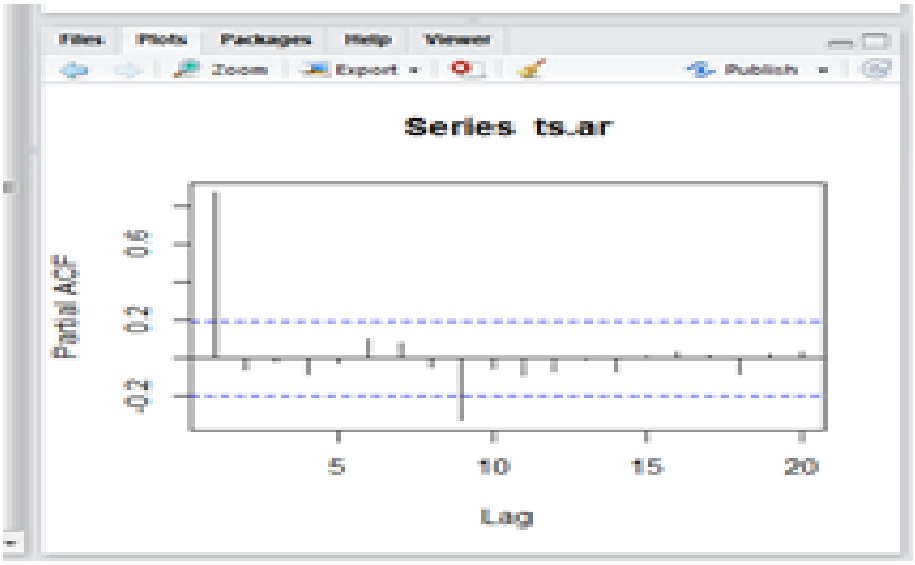
```
> (acf(ts.ar))

Autocorrelations of series 'ts.ar', by lag

    0     1     2     3     4     5     6     7     8     9    10    11    12    13
1.000 0.916 0.842 0.757 0.673 0.608 0.545 0.478 0.405 0.332 0.256 0.200 0.143 0.079
   14    15    16    17    18    19    20    21    22    23
0.025 -0.037 -0.086 -0.140 -0.188 -0.224 -0.265 -0.279 -0.293 -0.302
>
```



Les auto-corrélations :

(pacf(ts.ar))

```
> (pacf(ts.ar))

Partial autocorrelations of series 'ts.ar', by lag

     1      2      3      4      5      6      7      8      9     10     11     12     13     14     15
 0.872 -0.056 -0.017 -0.086 -0.023  0.103  0.081 -0.046 -0.326 -0.049 -0.088 -0.068 -0.005 -0.071  0.009  0
.034  0.018
    16     17     18     19     20
-0.084  0.021  0.033
>
```



2-Analyse du processus MA

Ces processus à tout instant t peuvent être extrapolés linéairement à partir de q résidus précédents ε t − 1, ..., ε t − p à un bruit blanc près.

X est dit moyenne mobile d'ordre q, si :

$$X_t = \epsilon_t + b_1\epsilon_{t-1} + \ldots + b_q\epsilon_{t-q},$$

Pour simuler un processus MA(1)

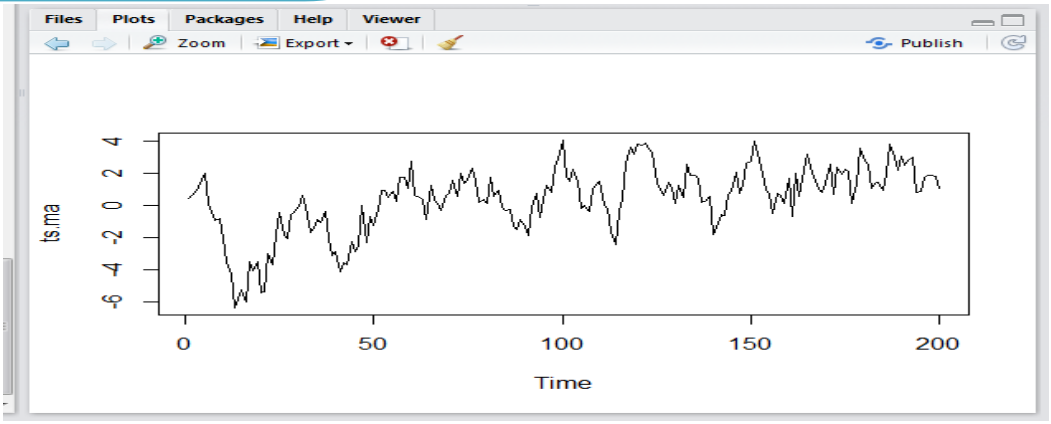
y t = 0.9 ε t − 1 + ε t

Déclaration du processus

```
ts.ma<- arima.sim(list(order = c(0,0,1), ar =0.9), n = 200)
```

Présentation graphique

```
>ts.plot(ts.ar,type="l")
```



Test de la stationnarité

```
adf.test(ts.ma, alternative=c("stationary"), k=0)
```

```
> adf.test(ts.ma, alternative=c("stationary"), k=0)

        Augmented Dickey-Fuller Test

data:  ts.ma
Dickey-Fuller = -7.4326, Lag order = 0, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(ts.ma, alternative = c("stationary"), k = 0) :
  p-value smaller than printed p-value
>
```

Test de l'inversibilité

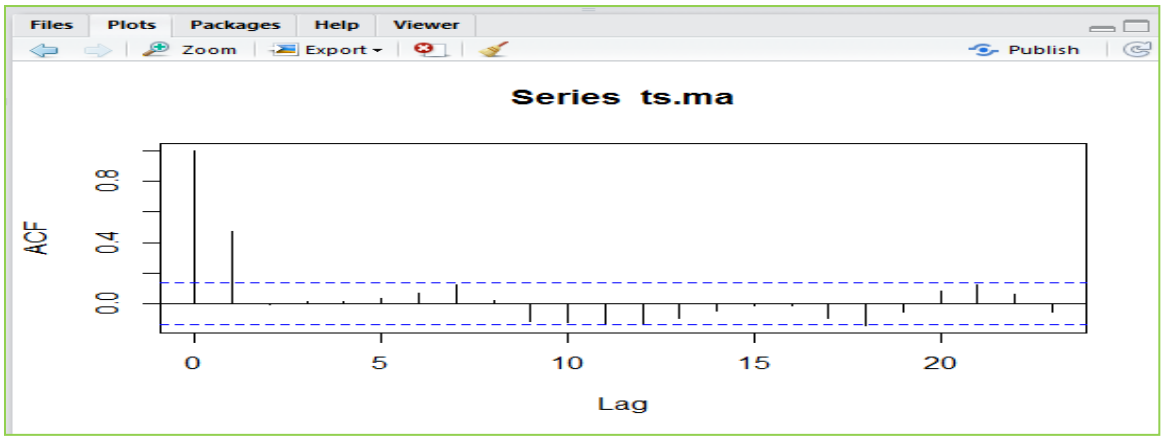
adf.test(ts.ma, alternative=c("explosive "), k=0)

```
> adf.test(ts.ma, alternative=c("explosive"), k=0)

        Augmented Dickey-Fuller Test

data:  ts.ma
Dickey-Fuller = -7.4326, Lag order = 0, p-value = 0.99
alternative hypothesis: explosive

Warning message:
In adf.test(ts.ma, alternative = c("explosive"), k = 0) :
  p-value smaller than printed p-value
>
```

Les auto-covariances

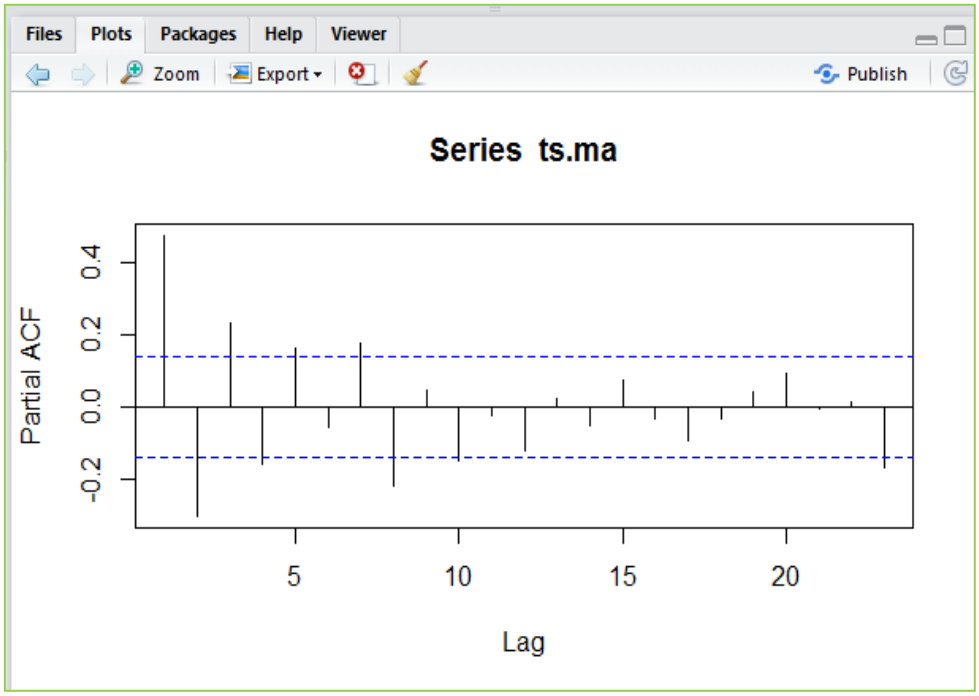
>acf(ts.ma)

```
Autocorrelations of series 'ts.ma', by lag

    0      1      2      3      4      5      6      7      8      9     10     11
12
1.000  0.475 -0.008  0.014  0.017  0.033  0.067  0.120  0.019 -0.118 -0.121 -0.129 -0.1
39
   13     14     15     16     17     18     19     20     21     22     23
```



Les auto-corrélations

```
>(pacf(ts.ma))
```




```
> (pacf(ts.ma))

Partial autocorrelations of series 'ts.ma', by lag

    1      2      3      4      5      6      7      8      9     10     11     12
13
 0.475 -0.302  0.231 -0.159  0.164 -0.053  0.178 -0.216  0.044 -0.147 -0.023 -0.118  0.0
22
    14     15     16     17     18     19     20     21     22     23
-0.053  0.074 -0.033 -0.091 -0.030  0.042  0.092 -0.007  0.014 -0.169
> |
```

3-Analyse du processus ARMA

- **ARMAacf** pour le calcul des covariances théorique d'un modèle ARMA
- **ARMAtoMA** pour le développement en MA infinie d'un modèle ARMA .
- **arima.sim** pour simuler des trajectoires d'un modèle ARMA ou ARIMA.
- La fonction diff différentie la série.
- Les fonctions **acf** et la **pacf** calculent la corrélation et la corrélation partiellle.
- La fonction arma () estime un ARMA :

fit <- arma(y, order = c(p,q)).

La table d'analyse de variance s'obtient alors avec **summary(fit)**, les graphiques de validation avec **plot(fit),** et les résidus avec : **r <- residuals(fit).**

- Pour le contrôle graphique de la normalité des résidus : **qqnorm** et **qqline**.
- Pour un test de normalité :**ks.test (test de Kolmogorov-Smirnov)**.
- La fonction predict() calcule les prévisions :

fitarima <- arima(x, order=c(p,d,q))

tsdiag(fitarima)

predict(fitarima, n.ahead = h)

tsdiag fournit des graphiques de validation complémentaire.

- La fonction **ar** permet d'estimer les paramètres d'un processus AR.

Pour les modèles ARMA d'ordre (p,q).

- **X[t] = a[1]X[t " 1] + ... + a[p]X[t " p] + e[t] + b[1]e[t " 1] + ... + b[q]e[t " q]**
- on utilise la fonction arima, la syntaxe est **out=arima(x,order=c(p,0,q))**
- la sortie out est une liste contenant :

out$coef : estimation des coefficients.

out$resid : estimation des résidus e[t].

Conclusion

R propose aux utilisateurs un langage et un environnement logiciel open source pour les calculs statistiques et graphiques.

R fourni une grande variété de statistiques (modélisation linéaire et non linéaire, tests de statistiques classiques, classification, clustering…) et de techniques graphiques, auxquels peut s'ajouter des éléments complémentaires.

Enfin, Un des points forts de R réside dans sa capacité à faciliter la production des graphiques en incluant des symboles mathématiques et des formules lorsque c'est nécessaire.