Analyse de séries temporelles avec R

Alexis Gabadinho

2023-10-05

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

Section 1

Introduction

Support de cours et exercices

- Ce support est écrit en R markdown (document texte incluant du code R exécuté dynamiquement)
- L'environnement RStudio sera utilisé lors de la formation
- Le fichier source du support (`Rmd`) sera fourni aux participant-e-s, et leur permettra d'exécuter le code contenu dans les diapositives sur leur ordinateur
- Plusieurs jeux de données contenant des séries macroéconomiques sont utilisées pour les exemples. Ces données sont disponibles sous la forme de fichiers `csv` ou fournies par certaines des librairies R utilisées
- De nombreuses formules mathématiques sont présentes dans les diapositives. Il n'est pas nécessaire de les comprendre, elles seront expliquées en détail lors de la formation

Librairies R requises

- Les librairies suivantes doivent être installées:
  - `tidyverse` (il s'agit en fait d'une collection de librairies dont: `tibble`, `tidyr`, `ggplot2`)
  - `GGally` (ajout de fonctions à `ggplot2`)
  - `ggfortify` (ajout de fonctions à `ggplot2`)
  - `tsibble` (objets de type `tidy` pour le stockage de données temporelles)
  - `feasts` (description, décomposition, représentations graphiques de séries temporelles)
  - `fable`
  - `structchange` (tests de changement structurel)
  - `urca` (test de racine unitaire)
  - `vars` (modèles VAR et VEC)

Section 2

Environnement de travail (rappels)

Le tidyverse et ggplot

- Pour importer et manipuler les données, on va utiliser principalement le tidyverse une collection de librairies pour la science des données (data science)
- Les librairies partagent des structures de données, une philisophie

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

- Pour les graphiques nous utiliserons principalement la librairie ggplot2, avec le theme minimal

```
library(ggplot2)
theme_set(theme_minimal())
```

Données Nelson-Plosser - Introduction

- Nous allons utiliser les données 'Nelson-Plosser', provenant de l'article orignal: C. R. Nelson and C. I. Plosser (1982), Trends and Random Walks in Macroeconomic Time Series. Journal of Monetary Economics
- Le jeu de données contient 14 séries temporelles macroécononiques
- La longueur des séries est variable, mais elles se terminent toutes en 1988
- Le jeu de données est décrit ici

Données Nelson-Plosser - Séries

- Les 14 séries temporelles:
  - cpi = consumer price index
  - ip = industrial production
  - gnp.nom = nominal GNP (millions de dollars 1988)
  - vel = velocity
  - emp = employment
  - int.rate = interest rate
  - nom.wages = nominal wages
  - gnp.def = GNP deflator
  - money.stock = money stock
  - gnp.real = real GNP (millions de dollars 1958)
  - stock.prices = stock prices (S&P500)
  - gnp.capita = GNP per capita (dollars 1958)
  - real.wages = real wages
  - unemp = unemployment

Données Nelson-Plosser - Importation

- La fonction `read.csv2` permet d'importer les données à partir du fichier csv
- A noter: dans le fichier csv, le séparateur de champ est une virgule et le séparateur décimal est un point (format européen)

```
Nelson_Plosser <- read.csv2("../data/Nelson_Plosser.csv", header=TRUE, sep=",", dec = ".")
head(Nelson_Plosser)
```

```
## # A tibble: 6 x 15
##    year   cpi     ip gnp.nom   vel   emp int.rate nom.wages gnp.def money.stock
##   <int> <dbl>  <dbl>   <dbl> <dbl> <dbl>    <dbl>     <dbl>   <dbl>       <dbl>
## 1  1860  3.30 -0.105      NA    NA    NA       NA        NA      NA          NA
## 2  1861  3.30 -0.105      NA    NA    NA       NA        NA      NA          NA
## 3  1862  3.40 -0.105      NA    NA    NA       NA        NA      NA          NA
## 4  1863  3.61  0          NA    NA    NA       NA        NA      NA          NA
## 5  1864  3.87  0          NA    NA    NA       NA        NA      NA          NA
## 6  1865  3.85  0          NA    NA    NA       NA        NA      NA          NA
## # i 5 more variables: gnp.real <dbl>, stock.prices <dbl>, gnp.capita <dbl>,
## #   real.wages <dbl>, unemp <dbl>
```

Données Nelson-Plosser - Statistiques descriptives

- Pour les statistiques descriptives de chaque variable on peut utiliser la fonction `descr()` de la librairie `summarytools`

```
library(summarytools)
statdesc <- descr(Nelson_Plosser %>% select(1:6))
```

Données Nelson-Plosser - Statistiques descriptives

```
statdesc
```

Descriptive Statistics

Nelson_Plosser

N: 129

	cpi	emp	gnp.nom	ip	vel	year
Mean	3.99	10.85	12.59	2.73	0.78	1924.00
Std.Dev	0.71	0.45	1.47	1.56	0.38	37.38
Min	3.22	9.96	10.42	-0.11	0.15	1860.00
Q1	3.39	10.53	11.35	1.48	0.53	1892.00
Median	3.78	10.78	12.46	2.77	0.68	1924.00
Q3	4.40	11.19	13.71	4.07	0.92	1956.00
Max	5.87	11.67	15.40	5.23	1.72	1988.00
MAD	0.67	0.49	1.67	1.92	0.24	47.44
IQR	1.01	0.63	2.34	2.59	0.38	64.00
CV	0.18	0.04	0.12	0.57	0.48	0.02
Skewness	1.06	-0.09	0.34	-0.12	0.97	0.00
SE.Skewness	0.21	0.24	0.27	0.21	0.22	0.21
Kurtosis	0.24	-0.94	-1.13	-1.13	0.13	-1.23
N.Valid	129.00	99.00	80.00	129.00	120.00	129.00
Pct.Valid	100.00	76.74	62.02	100.00	93.02	100.00

Données Nelson-Plosser - Matrice de corrélation

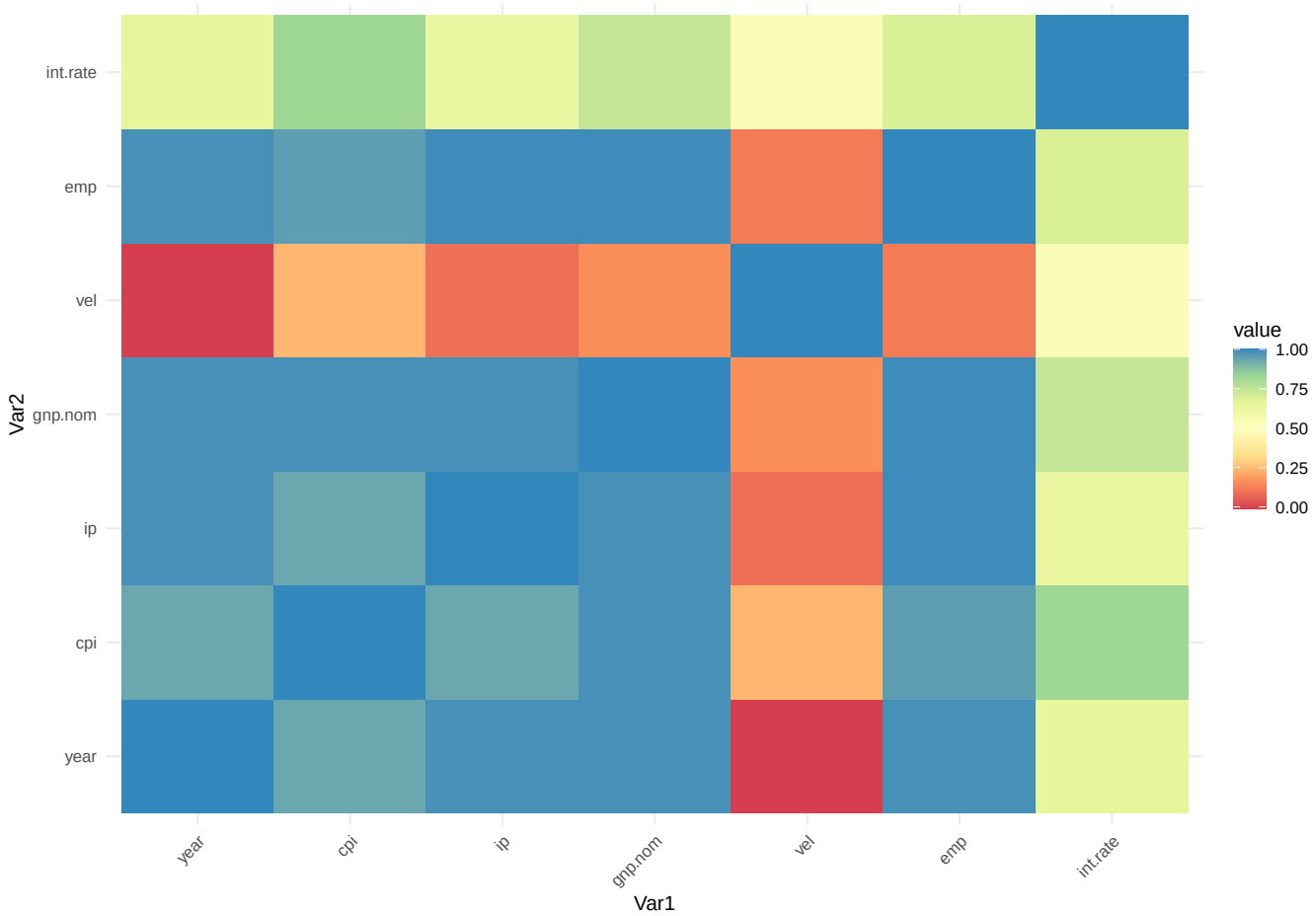
- La matrice de corrélation contient les coefficients de corrélation linéaire entre les variables prises 2 à 2

```
datanum <- Nelson_Plosser %>% filter(year>1909) %>% select(1:7)
cormat <- round(cor(datanum),2)
cormat
```

```
##             year  cpi   ip gnp.nom   vel  emp int.rate
## year        1.00 0.93 0.98    0.98 -0.01 0.98     0.65
## cpi         0.93 1.00 0.93    0.98  0.24 0.95     0.82
## ip          0.98 0.93 1.00    0.98  0.09 0.99     0.63
## gnp.nom     0.98 0.98 0.98    1.00  0.16 0.99     0.74
## vel        -0.01 0.24 0.09    0.16  1.00 0.12     0.53
## emp         0.98 0.95 0.99    0.99  0.12 1.00     0.69
## int.rate    0.65 0.82 0.63    0.74  0.53 0.69     1.00
```

Données Nelson-Plosser - Matrice de corrélation

```
library(reshape2)
cormat %>% melt() %>% ggplot(aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1)) +
  scale_fill_distiller(palette = "Spectral", direction = 1)
```



Données Nelson-Plosser - Pairplot

- On peut visualiser la distribution et la corrélation entre variables avec la fonction `ggpairs` de la librairie `GGally`

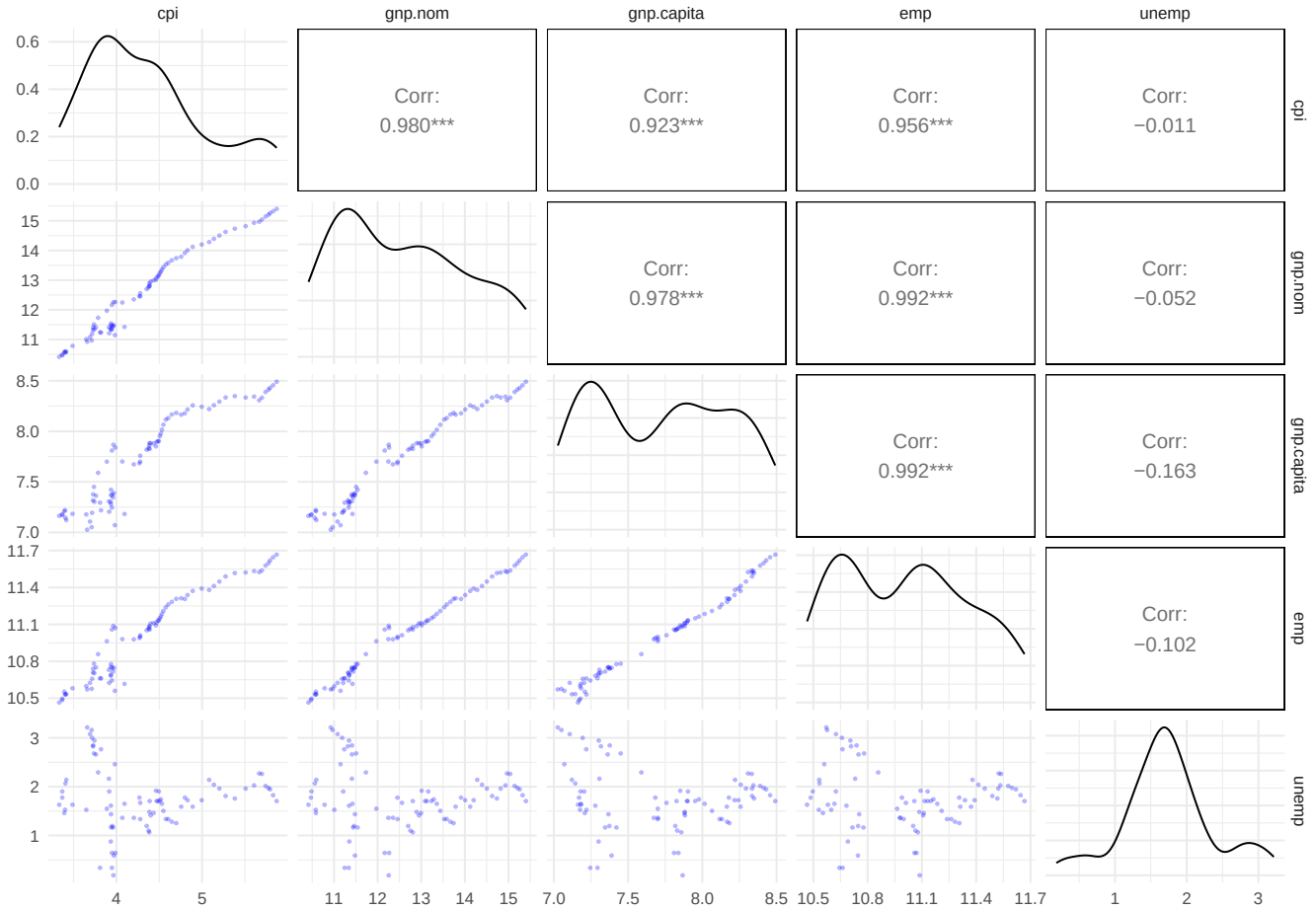
```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

- On sélectionne ici un sous-ensemble de variables pour limiter la taille du graphique

```
g <- Nelson_Plosser %>%
  filter(year>=1909) %>%
  select(cpi, gnp.nom, gnp.capita, emp, unemp) %>%
  ggpairs(upper = list(continuous = wrap("cor", size = 4)),
          lower = list(continuous = wrap("points", colour="blue", alpha=0.3, size=0.5)))
```

Données Nelson-Plosser - Pairplot

```
g
```



Données `retail` - Importation

- Série des ventes du commerce de détail et nourriture (Advance retail sales)
- Données mensuelles, en millions de dollars, non corrigées des variations saisonnières
- Source: FRED (Federal Reserve Bank Economic Data)
- On importe les données à partir du fichier csv
- A noter: dans le fichier csv, le séparateur décimal est un point

```
retail <- read.csv2("../data/RSXFSN.csv", header=TRUE, sep=",", dec = ".")
head(retail)
```

```
## # A tibble: 6 x 2
##   DATE       RSXFSN
##   <chr>       <dbl>
## 1 1992-01-01 130683
## 2 1992-02-01 131244
## 3 1992-03-01 142488
## 4 1992-04-01 147175
## 5 1992-05-01 152420
## 6 1992-06-01 151849
```

Données `retail` - Conversion de la date

- La date est au format `character`, il faut la convertir

```
summary(retail)
```

```
##      DATE                RSXFSN
##  Length:381         Min.   :130683
##  Class :character   1st Qu.:236830
##  Mode  :character   Median :315540
##                     Mean   :326550
##                     3rd Qu.:394764
##                     Max.   :654825
```

- On utilise la fonction `as.Date`

```
retail$DATE <- as.Date(retail$DATE,format="%Y-%m-%d")
summary(retail)
```

```
##       DATE                RSXFSN
##  Min.   :1992-01-01   Min.   :130683
##  1st Qu.:1999-12-01   1st Qu.:236830
##  Median :2007-11-01   Median :315540
##  Mean   :2007-10-31   Mean   :326550
##  3rd Qu.:2015-10-01   3rd Qu.:394764
##  Max.   :2023-09-01   Max.   :654825
```

Exercices

- Le fichier `pib_fr.csv` contient la série temporelle du PIB et de ses composants depuis 1949 (valeurs aux prix courants - Source: INSEE) :
- Importez les données à partir du fichier csv
- Explorez les données avec des statistiques descriptives et des graphiques

Section 3

Librairies spécialisées et structures de séries temporelles dans R

Liste des librairies

- En plus de la librairie standard `stats`, il existe plusieurs librairies R pour la manipulation et l'analyse des séries temporelles
  - tsibble
  - forecasts
  - feasts
  - fable
  - ggfortify
  - `tseries`
  - `zoo`
- Dans cette partie on se focalise sur les classes permettant de stocker les séries temporelles fournies par les librairies `stats` et `tsibble`

Les objets `ts` et `mts`

- La classe de base fournie par R pour représenter des séries temporelles s'appelle `ts` (ts = time series, série univariée) ou `mts` (mts = multiple time series, série multivariée)
- Cette classe est définie dans le package `stats`
- Elle concerne des séries temporelles qui sont échantillonnées à des périodes **équidistantes** dans le temps

Les objets `ts` et `mts` - Paramètres

- Les objets de classe `ts` ou `mts` possèdent trois paramètres caractéristiques :
  - `frequency`: 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
  - `start`: date de début de la série temporelle (nombre unique ou vecteur de deux entiers représentant respectivement une unité temporelle (e.g. une année) et une subdivision de cette unité (mois ou trimestre selon la valeur du paramètre frequency)
  - `end`: date de fin de la série temporelle, exprimée comme pour le paramètre start

Données `NelPlo` (Nelson-Plosser)

- Les données Nelson-Plosser sont également présentes dans la librairie `tseries`
- Les données annuelles (frequency=1) sont contenues dans un objet de type `mts`

```
library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
data(NelPlo)
class(NelPlo)
```

```
## [1] "mts" "ts"
```

Données NelPlo - Aperçu

- La fonction window permet d'extraire une portion d'une série temporelle
- L'ensemble des séries est complet à partir de 1909 (il y a des valeurs manquantes sur certaines séries avant)

```
window(NelPlo, start=1905, end=1910)
```

```
## Time Series:
## Start = 1905
## End = 1910
## Frequency = 1
##           cpi       ip  gnp.nom       vel      emp int.rate nom.wages  gnp.def
## 1905 3.295837 2.219203       NA 0.7561220 10.37274     3.50  6.329721 3.277145
## 1906 3.332205 2.282382       NA 0.8241754 10.42671     3.55  6.357842 3.303217
## 1907 3.367296 2.302585       NA 0.8241754 10.44497     3.80  6.393591 3.342862
## 1908 3.332205 2.140066       NA 0.7227060 10.41169     3.95  6.306275 3.335770
## 1909 3.332205 2.302585 10.41631 0.7839015 10.46516     3.77  6.395262 3.370738
## 1910 3.367296 2.360854 10.47164 0.7747272 10.48464     3.80  6.478510 3.397858
##      money.stock gnp.real stock.prices gnp.capita real.wages     unemp
## 1905    2.326302       NA     2.196113         NA   3.033991 1.4586150
## 1906    2.405142       NA     2.265921         NA   3.025776 0.5306283
## 1907    2.451005       NA     2.059239         NA   3.026261 1.0296194
## 1908    2.437116       NA     2.051556         NA   2.973998 2.0794415
## 1909    2.540026 4.760463     2.273156   7.163172   3.062924 1.6292405
## 1910    2.590767 4.788325     2.235376   7.170120   3.111291 1.7749524
```

Données growthofmoney (Growth of Money Supply)

- Deux séries temporelles concernant le contrôle de la monnaie par la réserve fédérale aux USA
- Article original: R.L. Hetzel (1981), The Federal Reserve System and Control of the Money Supply in the 1970's. Journal of Money, Credit and Banking 13, 31–43
- Série temporelle multivariée, données trimestrielles

```
library(lmtest)
data(growthofmoney)
window(growthofmoney, end=c(1971,4))
```

```
##         TG1.TG0 AG0.TG0
## 1970 Q2     0.0    -0.4
## 1970 Q3     1.0    -1.0
## 1970 Q4     1.0     1.1
## 1971 Q1     2.5     5.8
## 1971 Q2    -6.0    -4.4
## 1971 Q3     4.5    -1.6
## 1971 Q4    -0.5     1.6
```

Données `USIncExp` (US Income and Expenditure)

- Deux séries macro-économiques (USA) de janvier 1959 à février 2001, corrigées des variations saisonnières:
  - Revenus personnels mensuels aggrégés (millions de dollars)
  - Dépences de consommation aggrégées (millions de dollars)

```
library(strucchange)
data("USIncExp")
window(USIncExp, start=c(1959,6), end=c(1960,6))
```

```
##          income expenditure
## Jun 1959  396.3       319.2
## Jul 1959  396.5       318.8
## Aug 1959  395.0       321.2
## Sep 1959  396.2       325.2
## Oct 1959  397.8       323.8
## Nov 1959  401.2       323.9
## Dec 1959  405.7       323.9
## Jan 1960  407.0       324.6
## Feb 1960  407.7       326.4
## Mar 1960  408.6       331.2
## Apr 1960  411.3       337.6
## May 1960  412.8       331.1
## Jun 1960  413.1       331.2
```

Conversion des données `retail` en objet `ts`

- Pour les données retail, frequency=12 (mois)

```
retail_ts <- ts(retail['RSXFSN'], frequency=12, start=c(1992,1))
window(retail_ts, start=2020)
```

```
##          Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 2020 417123 413923 426258 376867 461785 479784 492689 487325 474038 493601
## 2021 461308 437031 560218 554269 565177 557446 552179 550169 529133 553588
## 2022 516923 507901 598541 596690 616626 609743 599929 613508 577966 597170
## 2023 547156 529374 604084 588220 631496 612243 605403 628892 592660
##          Nov    Dec
## 2020 490941 557696
## 2021 574978 627113
## 2022 605205 654825
## 2023
```

Objets ts - Extraction de l'index

- La fonction tsp permet d'extraire les propriétés d'un objet ts ou mts

```
tsp(retail_ts)
```

```
## [1] 1992.000 2023.667   12.000
```

- Extraction de l'index

```
retail2022 <- window(retail_ts, start=c(2022,1), end=c(2022,12))
time(retail2022)
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 2022 2022.000 2022.083 2022.167 2022.250 2022.333 2022.417 2022.500 2022.583
##           Sep      Oct      Nov      Dec
## 2022 2022.667 2022.750 2022.833 2022.917
```

- Conversion de l'index au format numérique

```
library(lubridate)
as.numeric(time(retail2022))
```

```
##  [1] 2022.000 2022.083 2022.167 2022.250 2022.333 2022.417 2022.500 2022.583
##  [9] 2022.667 2022.750 2022.833 2022.917
```

Les objets `tsibble`

- La librairie `tsibble` fournit une structure de type `tidy` pour les données temporelles ainsi que des outils pour le traitement de ces données
- Les objets de la classe `tsibble` possèdent deux attributs principaux:
  - `index` est la variable qui représente le temps (qui permet d'ordonner du passé au présent)
  - `key` identifie (éventuellement) la série (unité d'observation)
- Chaque observation est identifiée de manière unique par la combinaison `index` et `key`

Conversion des données growthofmoney

- On utilise la fonction as_tsibble()

```
library(tsibble)
gmoney_tsbl <- growthofmoney %>% as_tsibble()
gmoney_tsbl
```

```
## # A tibble: 38 x 3
##       index key     value
##       <qtr> <chr>   <dbl>
##  1 1970 Q2 TG1.TG0   0
##  2 1970 Q3 TG1.TG0   1
##  3 1970 Q4 TG1.TG0   1
##  4 1971 Q1 TG1.TG0   2.5
##  5 1971 Q2 TG1.TG0  -6
##  6 1971 Q3 TG1.TG0   4.5
##  7 1971 Q4 TG1.TG0  -0.5
##  8 1972 Q1 TG1.TG0  -1
##  9 1972 Q2 TG1.TG0   0.5
## 10 1972 Q3 TG1.TG0  -1.5
## # i 28 more rows
```

Conversion des données `NelPlo`

- La transformation d'un objet de type `mts` en objet `tsibble` produit un jeu de données au format long

```
nelplo_tsbl <- NelPlo %>% as_tsibble()
nelplo_tsbl %>% filter(index>1980 & key=="gnp.capita")
```

```
## # A tibble: 8 x 3
##   index key         value
##   <dbl> <chr>       <dbl>
## 1  1981 gnp.capita  8.34
## 2  1982 gnp.capita  8.31
## 3  1983 gnp.capita  8.33
## 4  1984 gnp.capita  8.39
## 5  1985 gnp.capita  8.41
## 6  1986 gnp.capita  8.43
## 7  1987 gnp.capita  8.46
## 8  1988 gnp.capita  8.49
```

Conversion des données `NelPlo`

- Les 14 séries sont identifiées par la variable `key`

```
nelplo_tsbl %>% distinct(key)
```

```
## # A tibble: 14 x 1
##    key
##    <chr>
##  1 cpi
##  2 ip
##  3 gnp.nom
##  4 vel
##  5 emp
##  6 int.rate
##  7 nom.wages
##  8 gnp.def
##  9 money.stock
## 10 gnp.real
## 11 stock.prices
## 12 gnp.capita
## 13 real.wages
## 14 unemp
```

Conversion des données `retail`

```
retail_tsbl <- retail_ts %>% as_tsibble()
head(retail_tsbl)
```

```
## # A tibble: 6 x 2
##        index  value
##        <mth>  <dbl>
## 1 1992 janv. 130683
## 2 1992 févr. 131244
## 3  1992 mars 142488
## 4 1992 avril 147175
## 5   1992 mai 152420
## 6  1992 juin 151849
```

Exercice

- Convertissez les données PIB en objet `mts`
- Convertissez les données PIB en objet `tsibble`

Section 4

Définitions

Série temporelle et processus stochastique

- Série temporelle univariée à temps discret = ensemble d'observations dans $\mathbb{R}$ enregistrées à un temps spécifique $t \in \mathbb{Z}$
- En statistique l'observation x est considérée comme la réalisation d'une **variable aléatoire** X
- Une série temporelle $(x_t)_{t\in\mathbb{Z}}$ sera considérée comme la réalisation d'un **processus stochastique** $(X_t)_{t\in\mathbb{Z}}$
- Processus stochastique => pour tout $t \in \mathbb{Z}$ fixé, X_t est une variable aléatoire réelle
- L'objectif est d'étudier les caractéristiques principales de ce processus (tendance, variation saisonnière), de le modéliser et de faire des prévisions

Stationarité

- Dans de très nombreux cas, on ne peut pas renouveler la suite de mesures dans des conditions identiques
- Pour que le modèle déduit à partir d'une suite d'observations ait un sens, il faut que toute portion de la trajectoire observée fournisse des informations sur la loi du processus et que des portions différentes, mais de même longueur, fournissent les mêmes indications. D'où la notion de stationnarité.
- Un processus $(X_t)_{t\in\mathbb{Z}}$ est (faiblement) stationnaire si son espérance et ses autocovariances sont invariantes par translation dans le temps :
  - $\forall t \in \mathbb{Z} : \mathbb{E}(X_t) = \mu$
  - $\forall t \in \mathbb{Z}, \forall h \in \mathbb{Z} : Cov(X_t, X_{t-h})$ dépend de l'intervalle h, mais pas de t

Fonction d'autocovariance et d'autocorrélation

- La fonction d'autocorrelation (ACF) est la corrélation entre x_t et x_{t-1}, x_{t-2}, x_{t-3}, etc ... :

$$\rho_j = \frac{Cov(y_t, y_{t-j})}{\sqrt{Var(y_t) \cdot Var(y_{t-j})}}$$

- Elle permet d'identifier une structure dans la série temporelle

Fonction d'autocorrélation avec `ACF()`

- Pour les objets `tsibble` on peut utiliser la fonction `ACF()` de la librairie `feasts`

```
library(feasts)
```

```
## Le chargement a nécessité le package : fabletools
```

```
nelplo_tsbl %>%
  filter(key=="cpi") %>%
  ACF(value)
```

```
## # A tibble: 21 x 3
##    key        lag   acf
##    <chr> <cf_lag> <dbl>
##  1 cpi         1Y 0.966
##  2 cpi         2Y 0.928
##  3 cpi         3Y 0.889
##  4 cpi         4Y 0.853
##  5 cpi         5Y 0.818
##  6 cpi         6Y 0.784
##  7 cpi         7Y 0.748
##  8 cpi         8Y 0.712
##  9 cpi         9Y 0.677
## 10 cpi        10Y 0.645
## # i 11 more rows
```

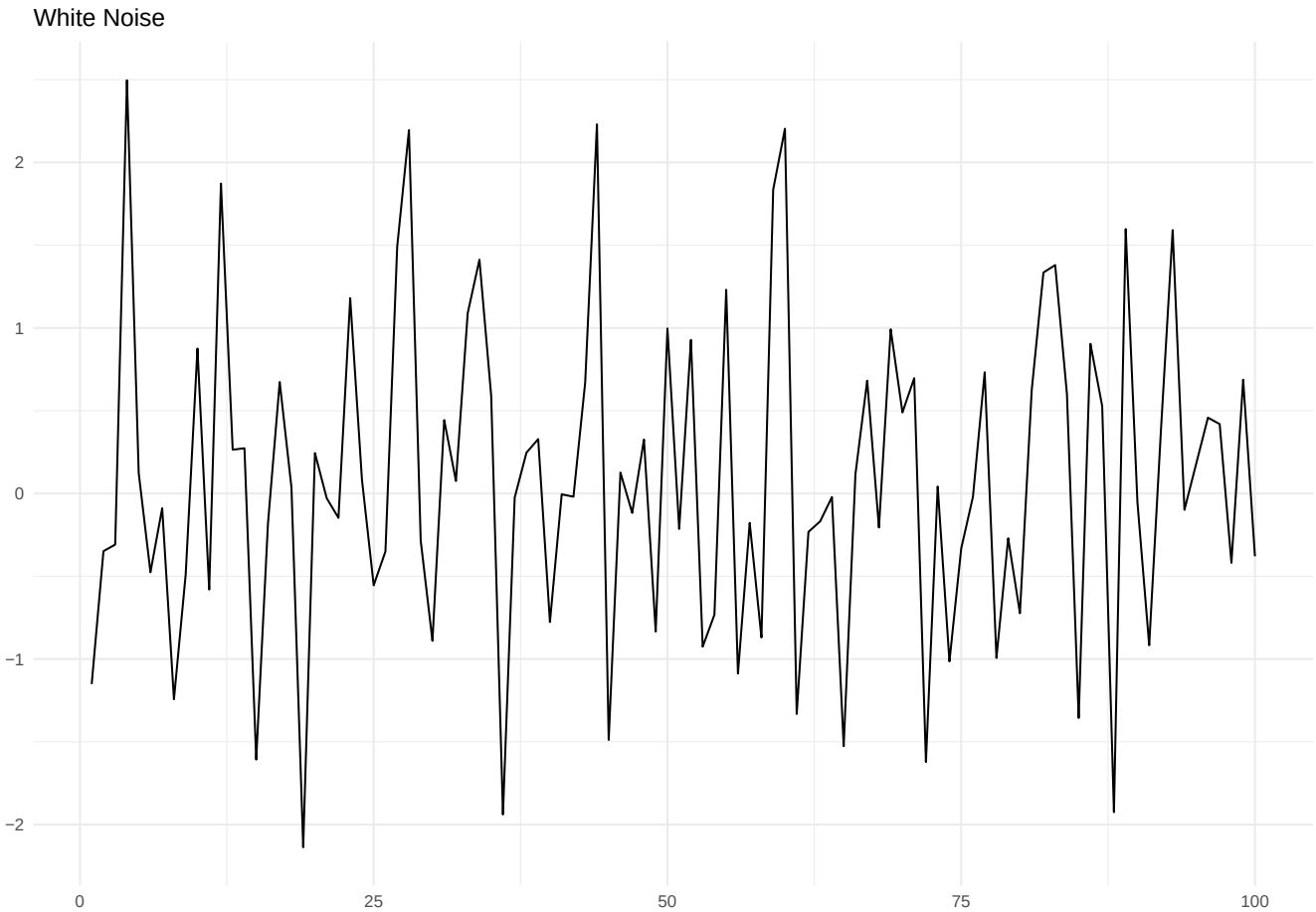
- La corrélation entre le cpi à l'instant t et le cpi à l'instant $t-1$ (une année avant) est de 0.966

Bruit blanc (white noise)

- Un bruit blanc ϵ_t est une série de variables aléatoires **non corrélées** de moyenne nulle et de variance constante
- $(\epsilon_t)_{t\in\mathbb{Z}}$ est un bruit blanc faible si :
  - Son espérance est égale à 0: $\forall t \in \mathbb{Z} : \mathbb{E}(\epsilon_t) = 0$
  - Sa variance est constante: $\mathbb{E}(\sigma_t^2) = \sigma^2$
  - La covariance entre (ϵ_t) et (ϵ_{t-h}) est nulle : $\forall t \in \mathbb{Z}, \forall h \in \mathbb{Z} : Cov(\epsilon_t, \epsilon_{t-h}) = 0$
- Un bruit blanc gaussien ϵ_t est une série de variables aléatoires indépendantes et identiquement distribuées (i.i.d), suivant une loi normale de moyenne nulle et de variance σ_z^2 $N(0, \sigma_z^2)$
- Un bruit blanc est une série strictement stationnaire

Bruit blanc (white noise) - Graphique

```
library(ggfortify)
autoplot(ts(rnorm(100)))+
  ggtitle("White Noise")
```



Test de Portmanteau

- A l'issue d'une modélisation, il nous faudrait idéalement obtenir un signal résiduel qui ne contient plus d'information temporelle
- Dans le cadre des modèles ARMA, on souhaite que le résidu soit un bruit blanc (faible), c'est-à-dire sans dépendance temporelle linéaire
- On peut tester la **blancheur** d'une série $y_t, t = 1, \ldots, T$ en utilisant le test de Portmanteau
- La statistique de Portmanteau est calculée à partir les k premiers coefficients d'autocorrélation $Q_k = T \sum_{h=1}^{k} \hat{\rho}_j^2$ où h est un décalage choisi par l'utilisateur et ρ_j l'estimateur du coefficient d'autocorrélation d'ordre j de la série y_t

Test de Portmanteau - Exemple

- On peut utiliser les fonctions `box_pierce()` et `ljung_box()` (pour les petits échantillons) de la librairie `feasts`
- Hypothèse H0: pas d'autocorrélation

```
box_pierce(rnorm(100))
```

```
##     bp_stat   bp_pvalue
## 0.004199147 0.948332587
```

- La p-valeur est supérieure à 0.05, on accepte H0
- A noter: quand le test est appliqué non sur des v.a. indépendantes, mais sur les résidus d'un ajustement estimant m paramètres, on utilise le paramètre `dof` (degrees of freedom, degrés de liberté)

Marche aléatoire (random walk)

- La série $y_t = y_{t-1} + \epsilon_t$ est une **marche aléatoire**
- Une série suivant une marche aléatoire prend, à deux dates consécutives, des valeurs proches et la variation par rapport à la date précédente est **indépendante du passé** (c'est un bruit blanc)
- En exprimant y_{t-1} en fonction de $y_{t-2}, \ldots$ et d'une valeur initiale y_0 on obtient : $y_t = y_0 + \epsilon_1 + \epsilon_2 + \ldots + \epsilon_t$
- Espérance: $E(y_t) = y_0$
- Variance: $var(y_t) = t \cdot \sigma_\epsilon^2$
- Covariance: $cov(y_t, y_{t+k}) = t \cdot \sigma_z^2, (k > 0)$
- Il s'agit d'une série non-stationnaire car ni la variance ni l'autoccorélation ne sont constantes

Marche aléatoire (random walk) - Simulation

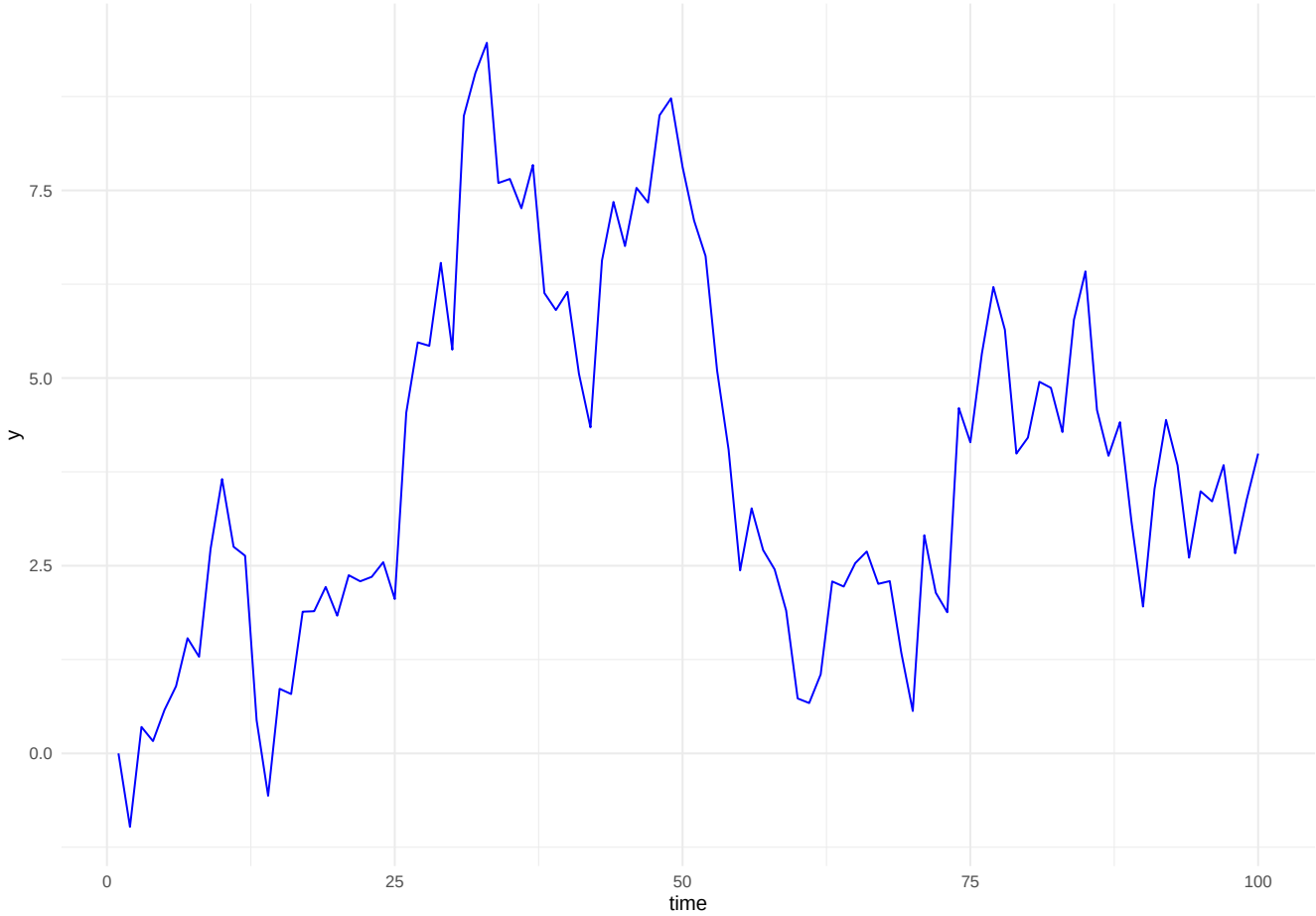
- On simule une marche aléatoire avec un bruit blanc gaussien de moyenne 0 et d'écart type 1

```
tmax <- 100
# ourDrift <- 0.005
wnoise <- rnorm(99, mean=0, sd=1)
y <- rep(0,100)

for (t in 2:tmax) {
  y[t] = y[t-1] + wnoise[t-1]
}
rw <- tibble(time=1:tmax, y=y)
rw
```

```
## # A tibble: 100 x 2
##     time      y
##    <int>  <dbl>
##  1     1  0
##  2     2 -0.979
##  3     3  0.352
##  4     4  0.163
##  5     5  0.579
##  6     6  0.895
##  7     7  1.53
##  8     8  1.29
##  9     9  2.72
## 10    10  3.65
## # i 90 more rows
```

Marche aléatoire (random walk) - Graphique

```
ggplot(rw) + geom_line(aes(x=time, y=y), colour="blue")
```



Marche aléatoire avec dérive (random walk with drift)

- La série $y_t = \alpha + y_{t-1} + \epsilon_t$ est une **marche aléatoire** avec dérive
- En exprimant y_{t-1} en fonction de $y_{t-2}, \ldots$ et d'une valeur initiale y_0 on obtient : $y_t = \alpha \cdot t + y_0 + \epsilon_1 + \epsilon_2 + \ldots + \epsilon_t$
- Espérance: $E(y_t) = \alpha \cdot t + y_0$
- Variance: $var(y_t) = t \cdot \sigma_z^2$
- Covariance: $cov(y_t, y_{t+k}) = t \cdot \sigma_z^2, (k > 0)$

Marche aléatoire avec dérive - Simulation

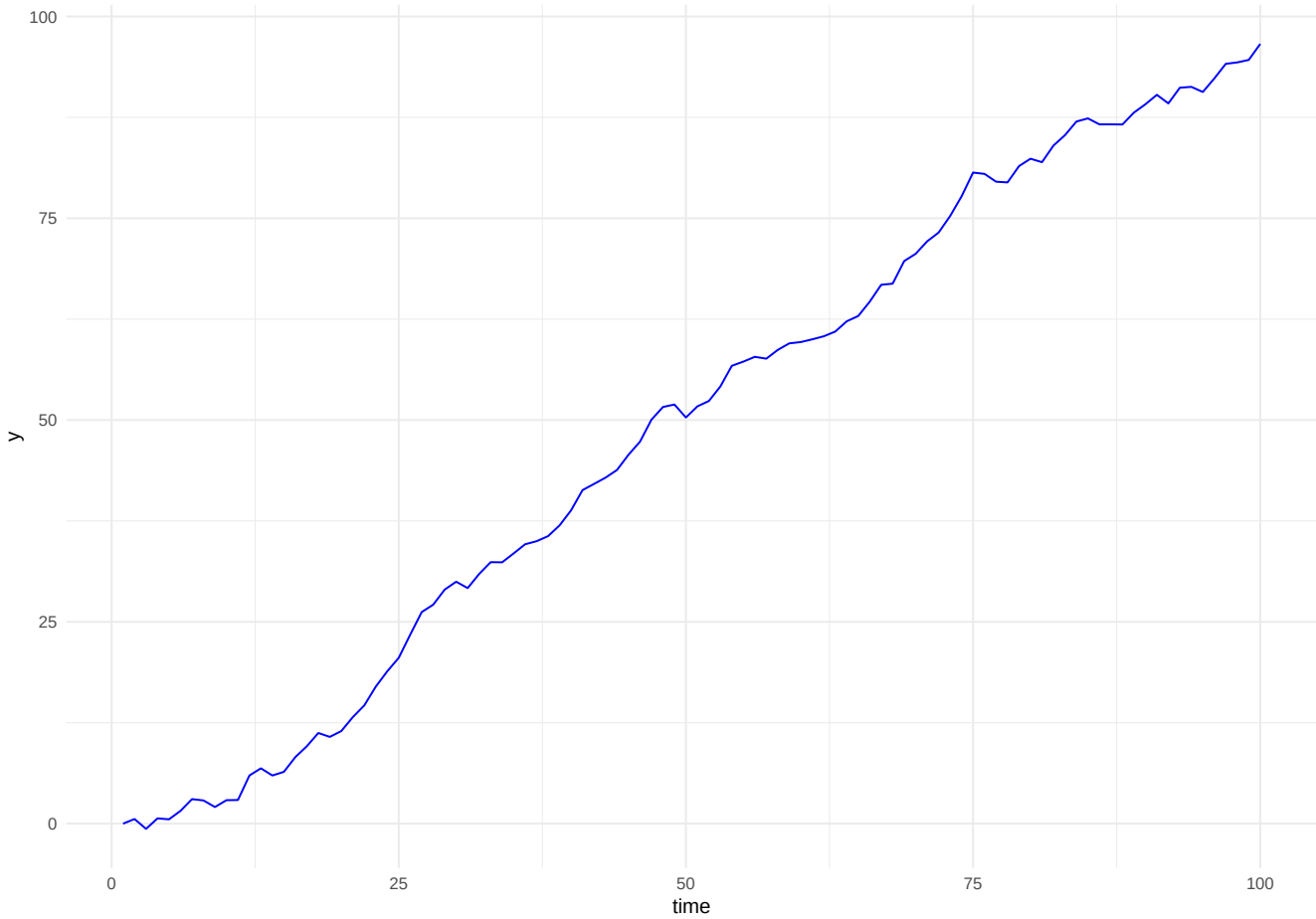
```
tmax <- 100

wnoise <- rnorm(99, mean=0, sd=1)
y <- rep(0,100)
alpha <- 0.8

for (t in 2:tmax) {
  y[t] = alpha + y[t-1] + wnoise[t-1]
}
rwd <- tibble(time=1:tmax, y=y)
rwd
```

```
## # A tibble: 100 x 2
##     time      y
##    <int>  <dbl>
##  1     1  0
##  2     2  0.571
##  3     3 -0.648
##  4     4  0.649
##  5     5  0.535
##  6     6  1.58
##  7     7  3.04
##  8     8  2.87
##  9     9  2.06
## 10    10  2.91
## # i 90 more rows
```

Marche aléatoire avec dérive: Graphique

- Le graphique de y_t en fonction du temps est donc celui d'une droite à laquelle est superposée une marche aléatoire

```
ggplot(rwd) + geom_line(aes(x=time, y=y), colour="blue")
```



Exercice

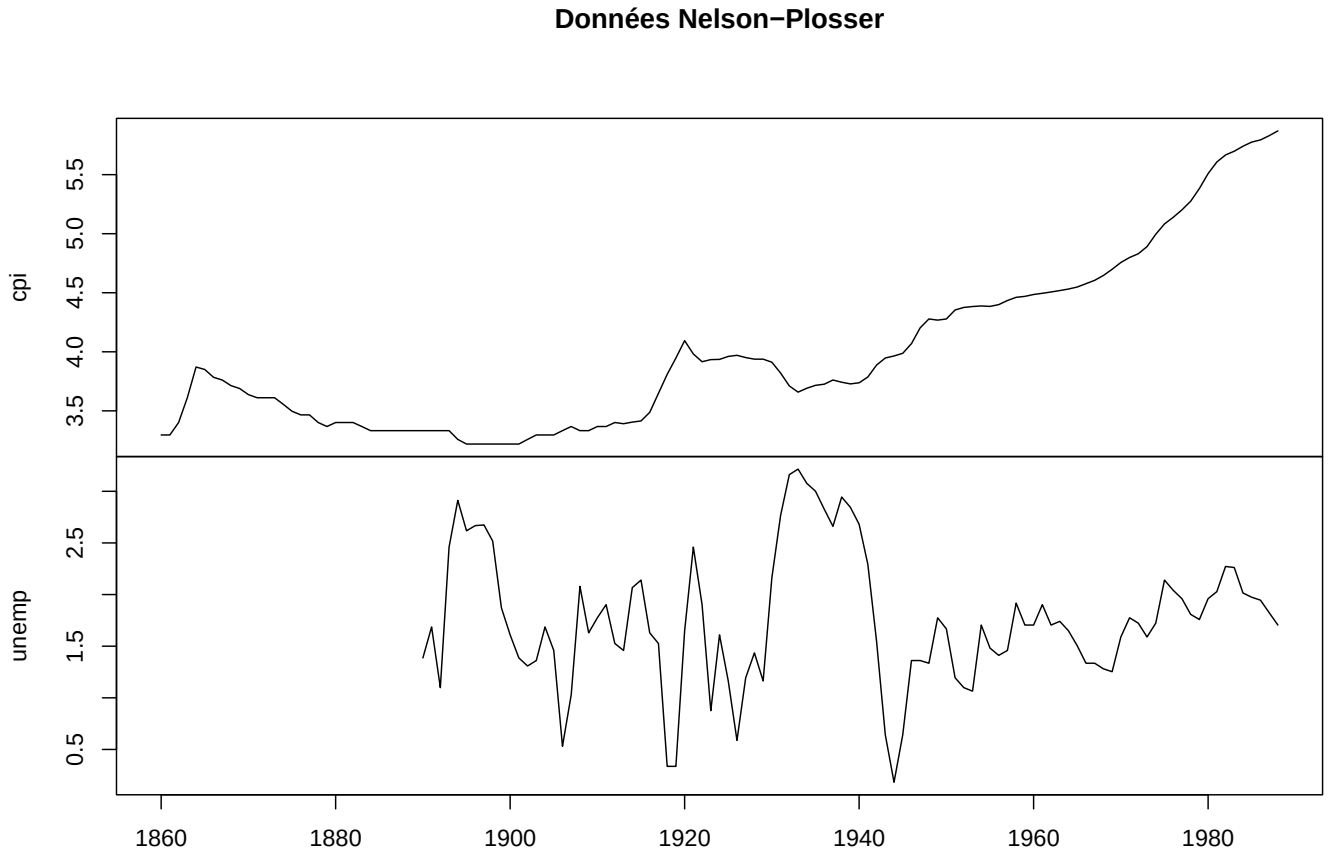
- Construisez une série y_t représentant une marche aléatoire avec $y_0 = 1.39$ (données Nelson-Plosser) et $\sigma_\epsilon^2 = 0.41$
- Calculez les coefficients d'autocorrélation de la série
- Réalisez un test de Portmanteau sur la série, que concluez vous ?

Section 5

Analyse descriptive et représentations graphiques

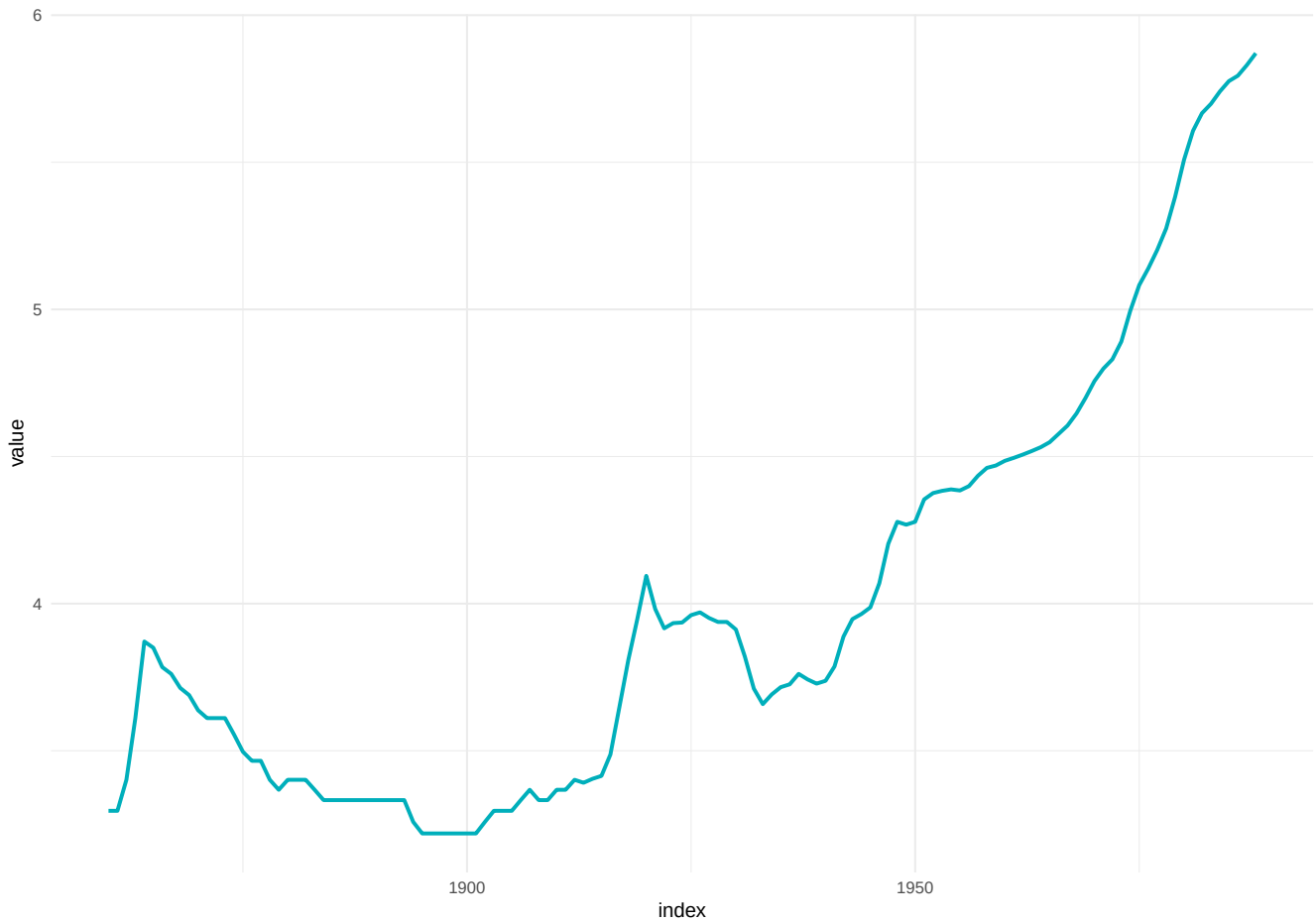
Représenter des séries temporelles: chronogramme

- Chronogramme: diagramme des points (x=date, y=valeur de l'observation)
- Pour un objet de la classe `ts` on peut utiliser la méthode générique `plot()`



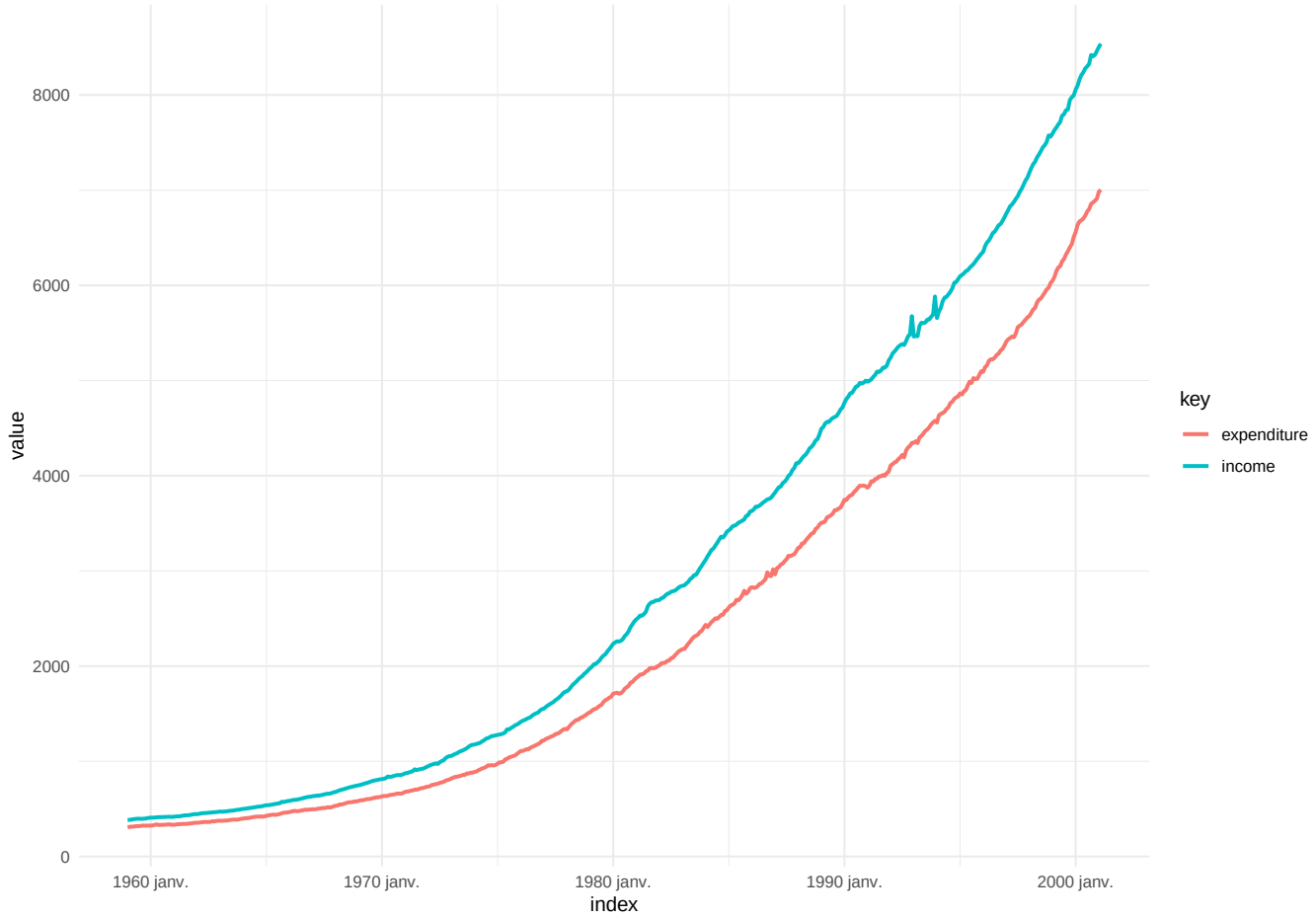
Chronogramme - Données Nelson Plosser - Price index

```
nelplo_tsbl %>% filter(key=="cpi") %>%
  ggplot(aes(x = index, y = value))+
    geom_line(color = "#00AFBB", size = 1)
```



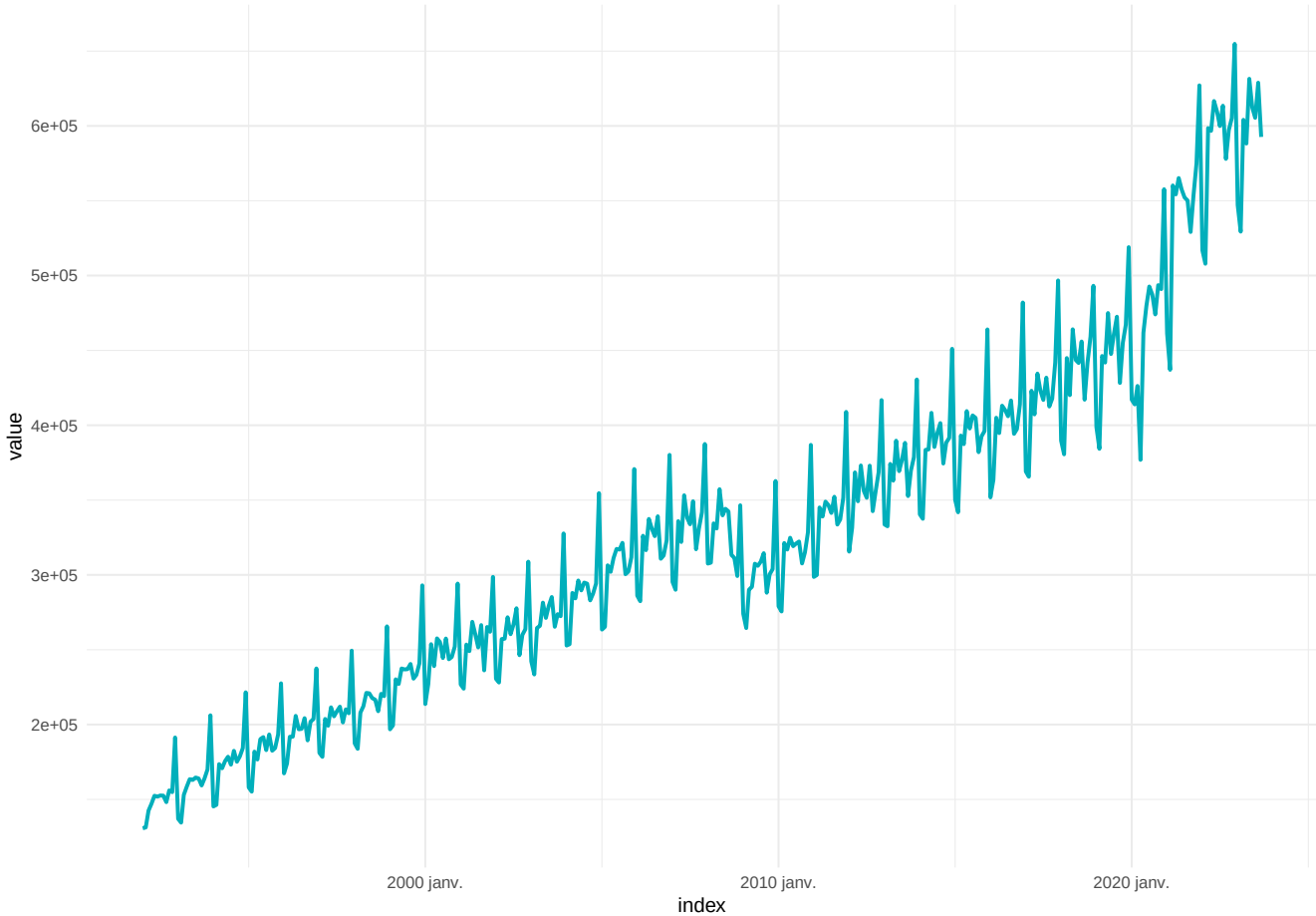
Chronogramme - Données USIncExp

```
USIncExp %>% as_tsibble() %>%
  ggplot(aes(x = index, y = value))+
    geom_line(aes(color = key), size = 1)
```



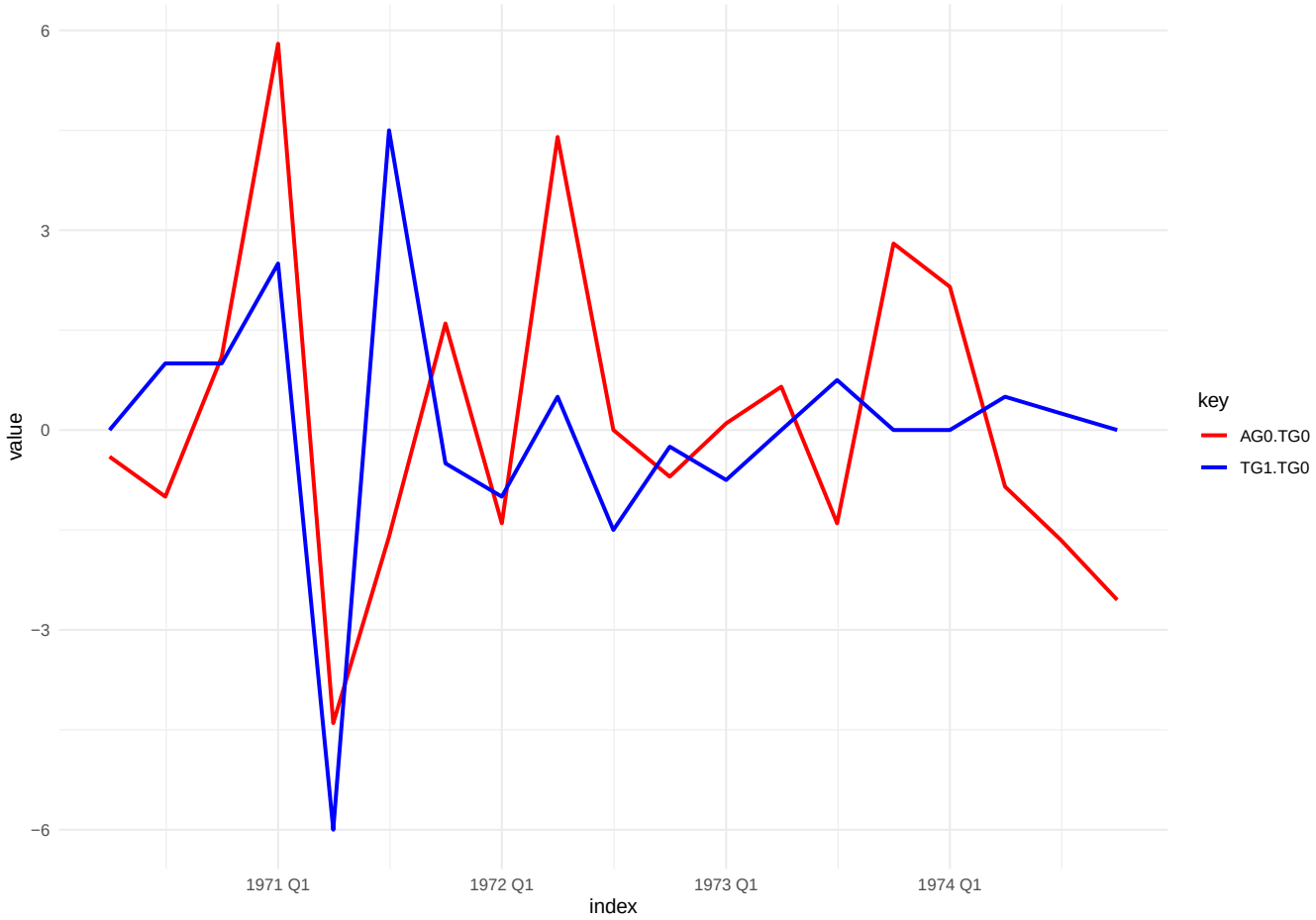
Chronogramme - Données retail

```
retail_tsbl %>%
  ggplot(aes(x = index, y = value))+
    geom_line(color = "#00AFBB", size = 1)
```



Chronogramme - Séries growthofmoney

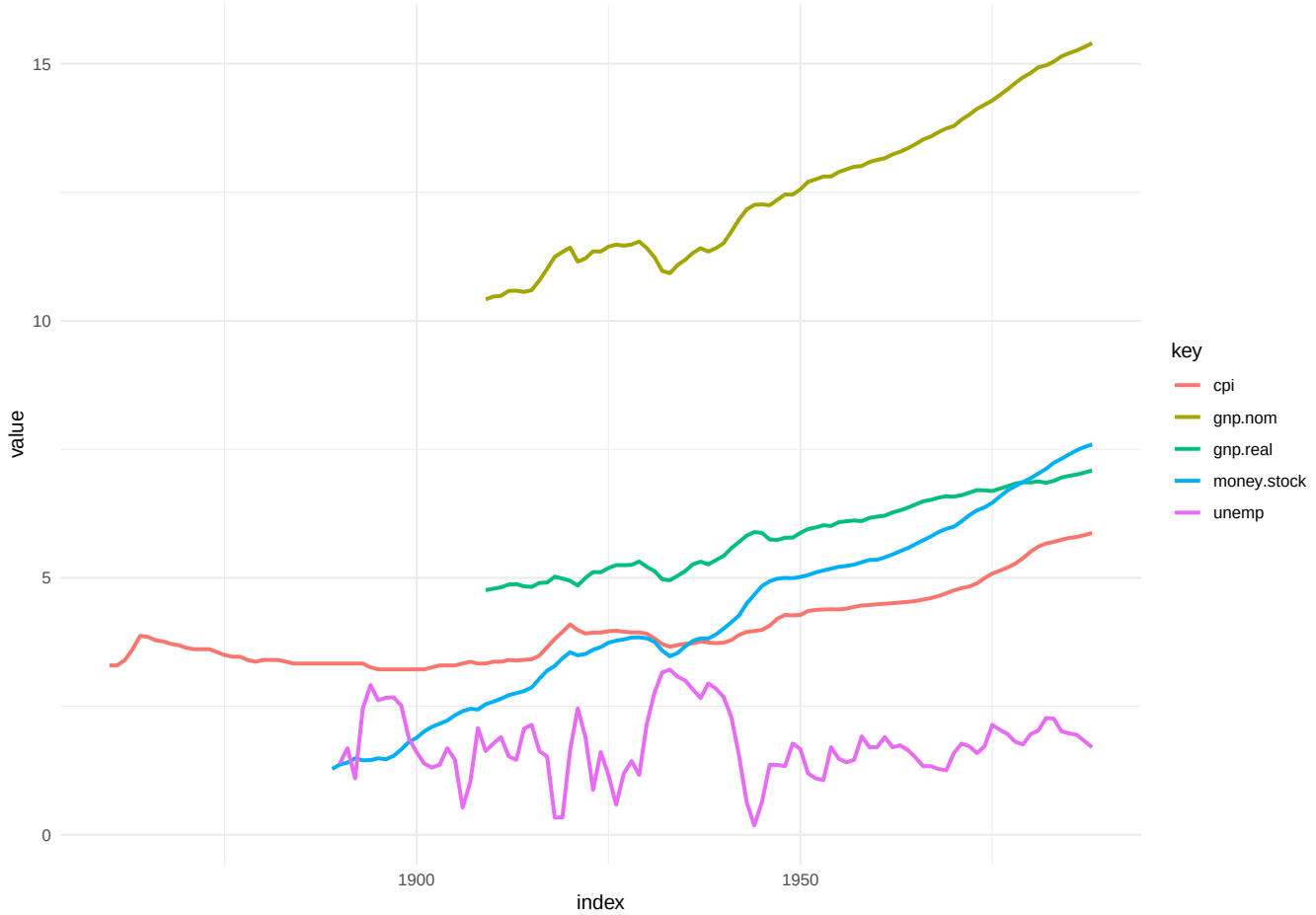
```
ggplot(gmoney_tsbl, aes(x = index, y = value)) +
  geom_line(aes(color = key), size = 1) +
  scale_color_manual(values = c("red", "blue"))
```



Séries du jeu de données nelplo (Nelson-Plosser)

```
nelplo_tsbl %>% filter(key %in% c("gnp.nom", "gnp.real", "unemp", "cpi", "money.stock")) %>%
  ggplot(aes(x = index, y = value)) +
    geom_line(aes(color = key), size = 1)
```

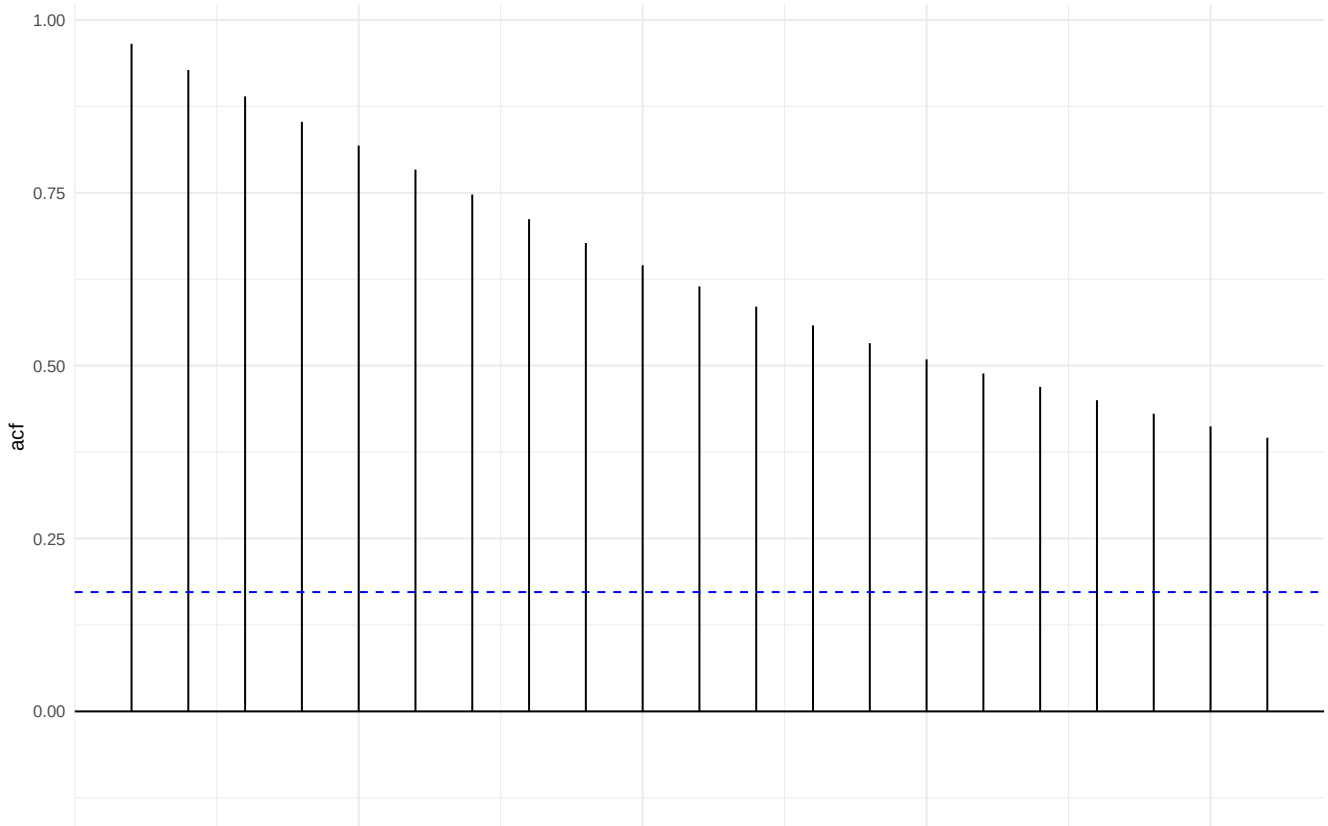
```
## Warning: Removed 157 rows containing missing values (`geom_line()`).
```



Fonction d'autocorrélation - Données Nelson-Plosser - Price index

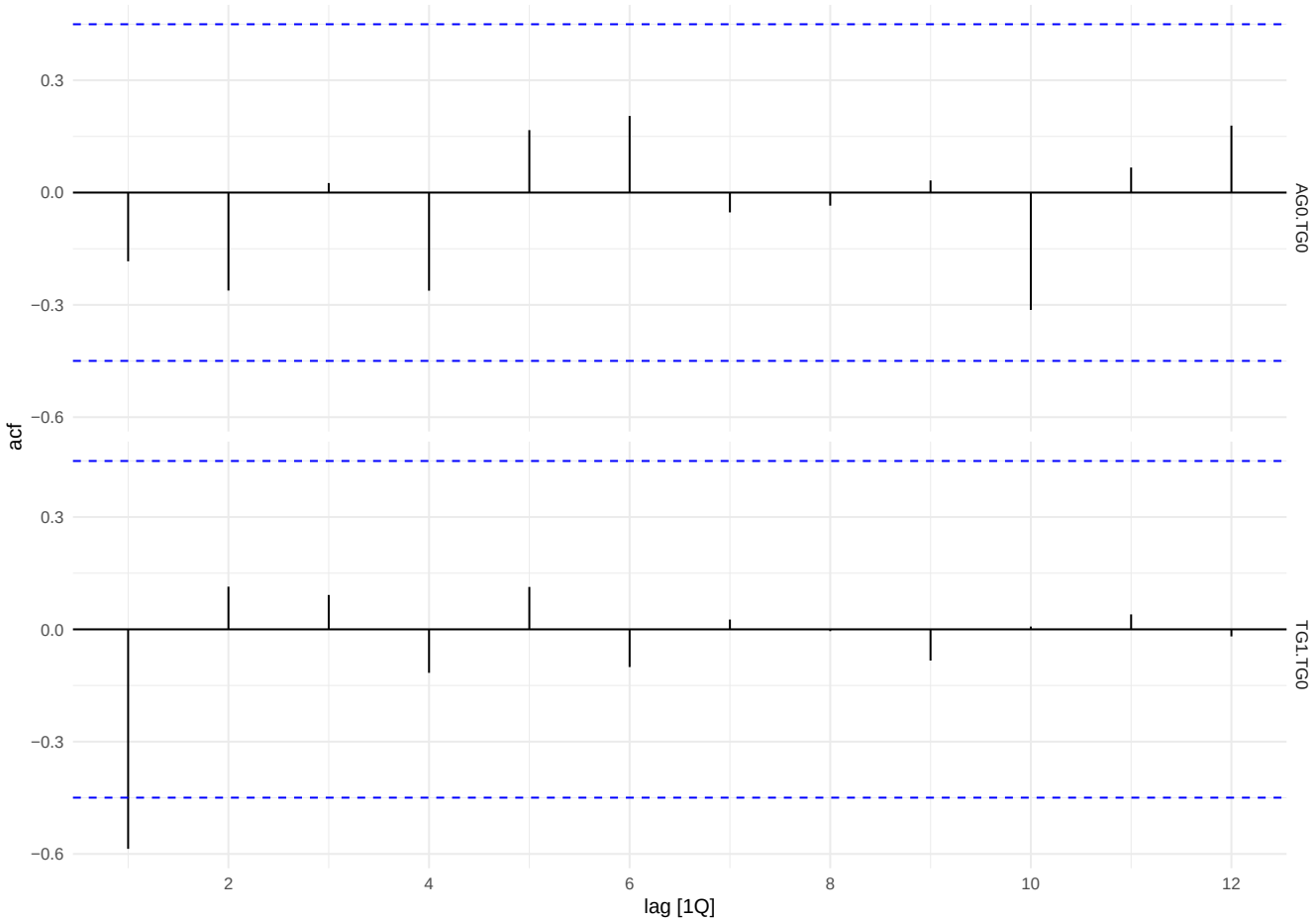
- On peut représenter simplement la fonction d'autocorrélation obtenue avec la fonction `ACF()`

```
nelplo_tsbl %>% filter(key=="cpi") %>%
  ACF(value) %>% autoplot()
```



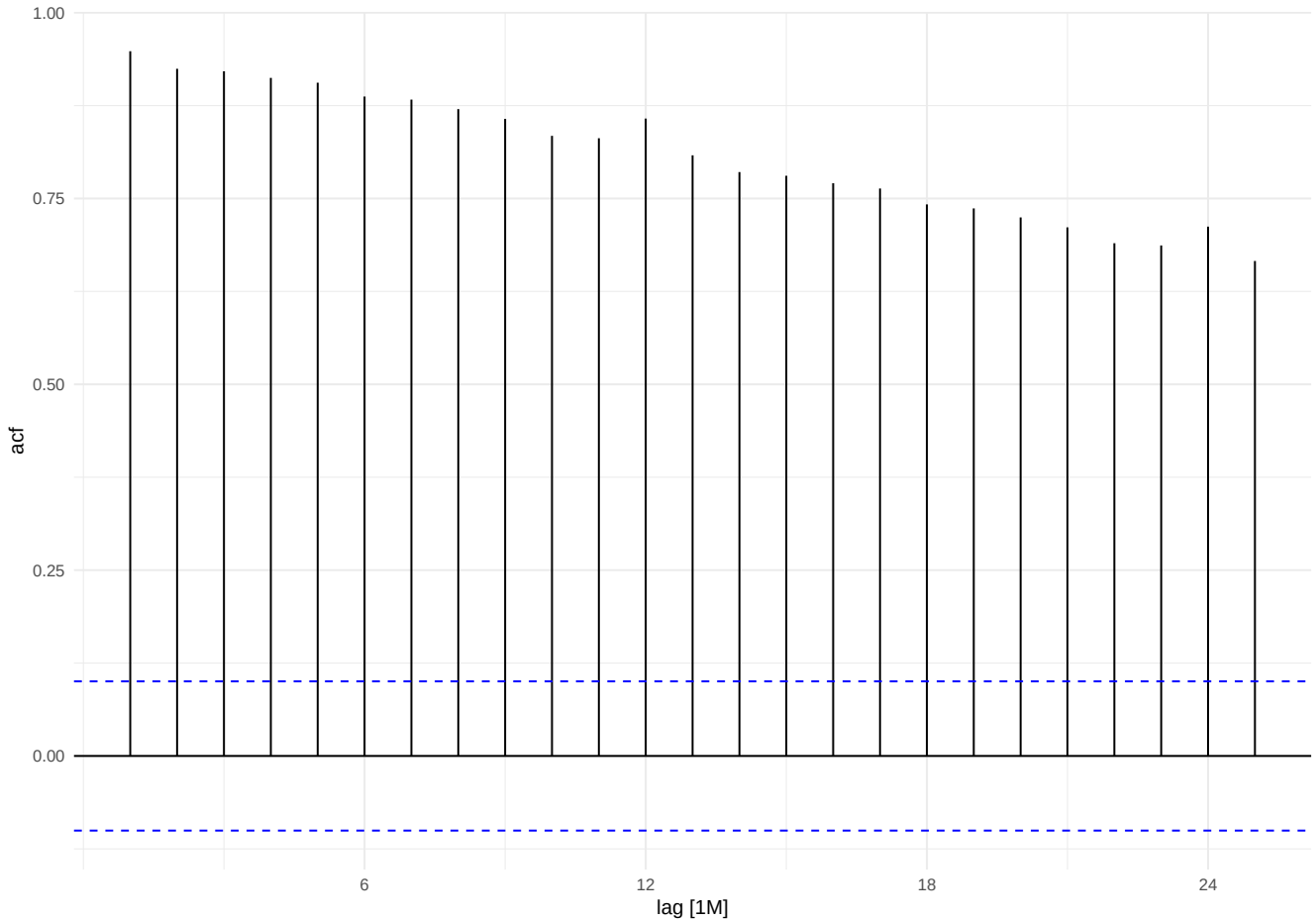
Représentation de la fonction d'autocorrélation - Données growthofmoney

```
gmoney_tsbl %>% ACF() %>% autoplot()
```



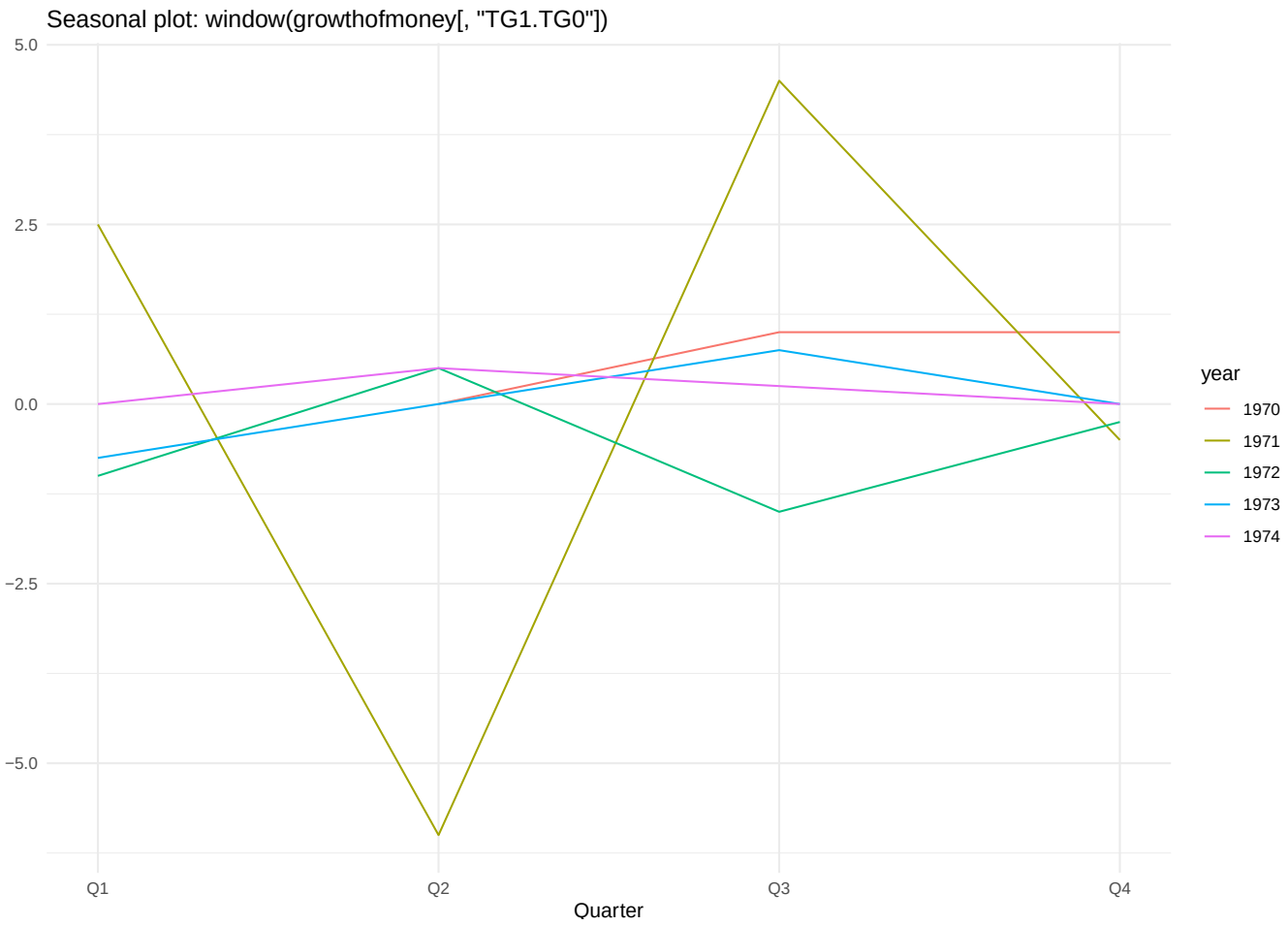
Autocorrélation - Données `retail`

```
retail_tsbl %>% ACF() %>% autoplot()
```



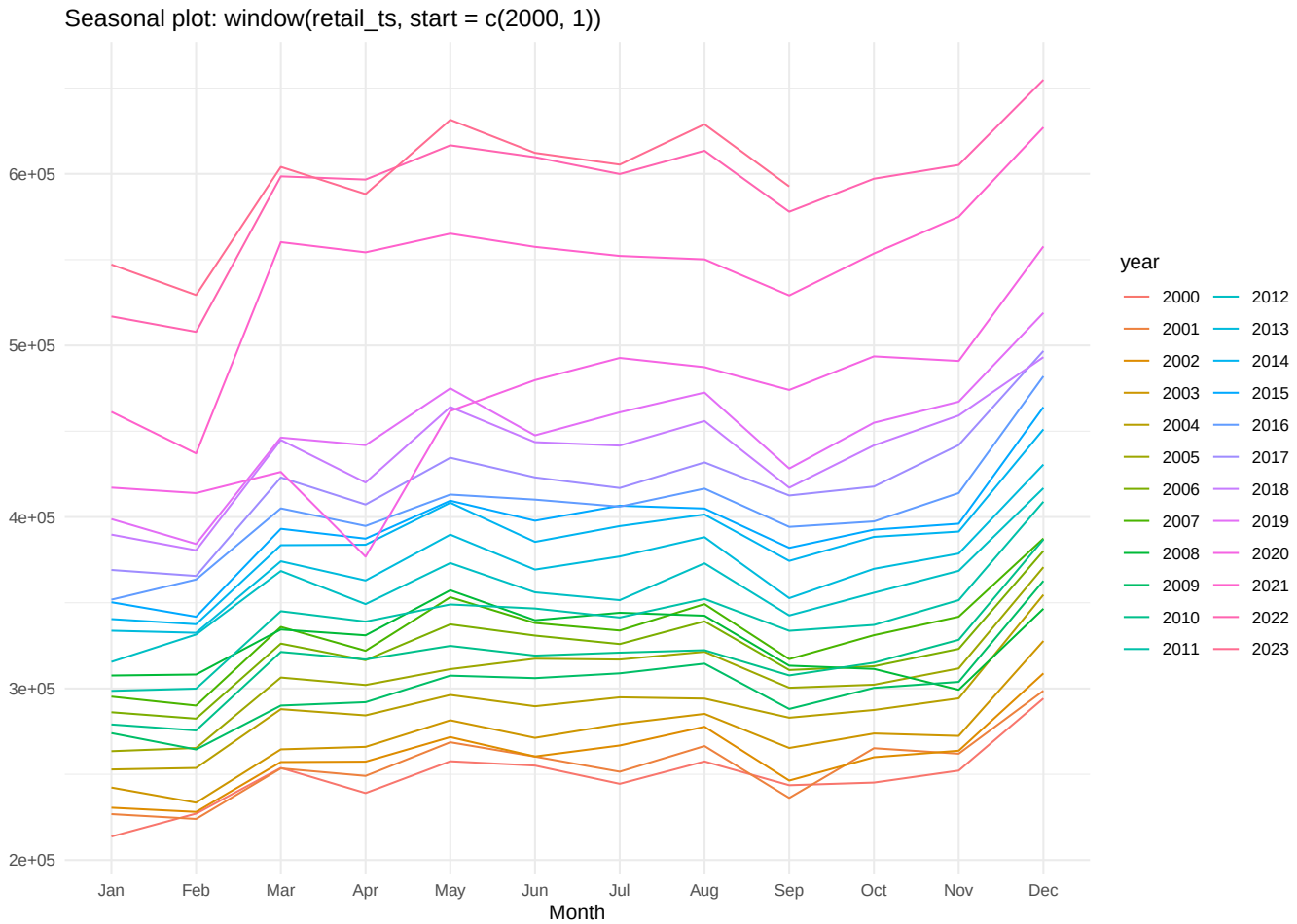
Season plot - Données growthofmoney

```
library(forecast)
ggseasonplot(window(growthofmoney[,"TG1.TG0"]))
```



Season plot - Données `retail`

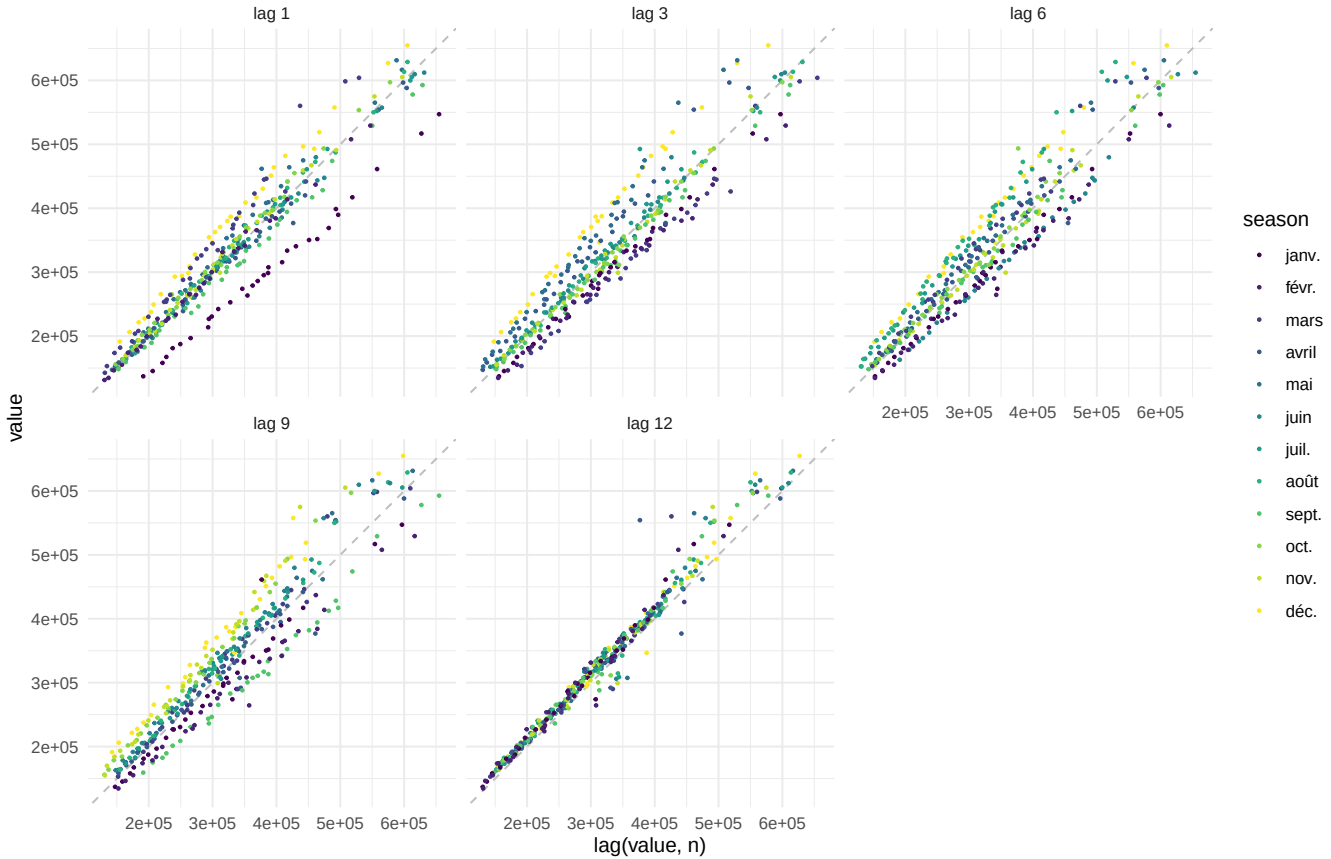
```
ggseasonplot(window(retail_ts, start=c(2000,1)))
```



Lag plot - Données `retail`

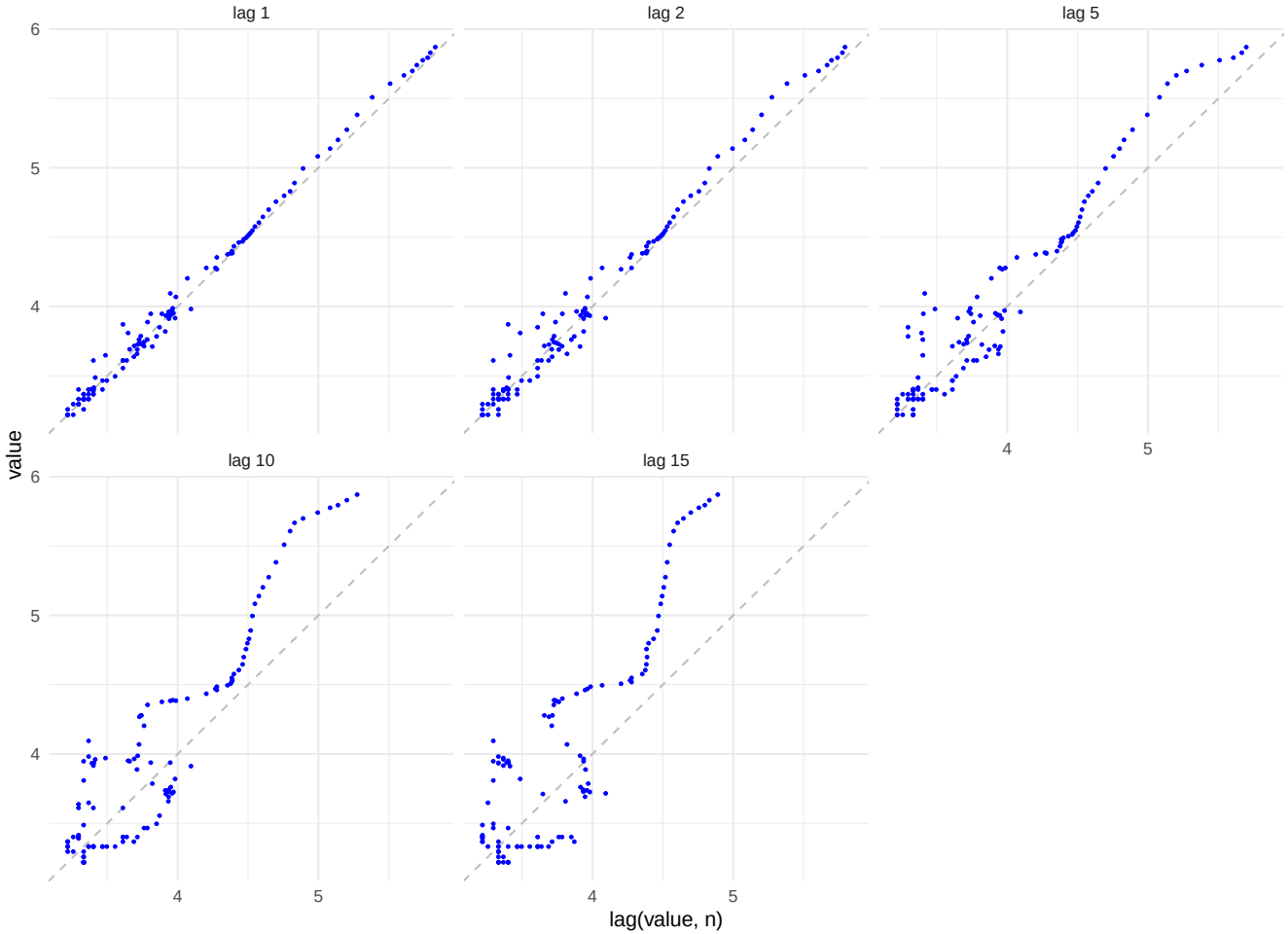
- Le 'lag plot' représente la série temporelle et ses valeurs précédentes

```
retail_tsbl %>% gg_lag(y=value, geom="point", size=0.5, lags=c(1,3,6,9,12))
```



Lag plot - Données Nelson-Plosser- Série cpi

```
nelplo_tsbl %>% filter(key=='cpi') %>%
  gg_lag(y=value, geom="point", size=0.5, colour="blue", lags=c(1,2,5,10,15))
```



Exercice

- Représentez les séries des données PIB (chronogramme, fonction d'autocorrélation, lag plot)

Section 6

Régression Linéaire

Le modèle de régression linéaire simple

- Modèle de régression linéaire simple (une seule variable indépendante):

$$y_i = \beta_1 + \beta_2 \cdot x_i + \epsilon_i$$

  - Les observations sont indicées par i, $(i = 1, \ldots, N)$
  - y_i est la variable **expliquée** (dépendante)
  - x_i est la variable **explicative** (indépendante)
  - β_1 et β_2 sont les **paramètres** (à estimer)
  - ϵ_i est le **résidu** (écart aléatoire, erreur)
- L'équation de la droite de régression est déterminée par la pente (slope) (β_0) et l'intercept (β_1) :

$$E(y|x) = \beta_1 + \beta_2 \cdot x$$

Hypothèses de base du modèle linéaire

- On parle de Moindres Carrés Ordinaires (MCO) ou Ordinary Least Square model (OLS) car l'objectif lors de l'estimation est de minimiser la somme des erreurs au carré $\sum_i \epsilon_i^2$
- Les hypothèses de base concernent la distribution de probabilité des résidus ϵ_i:
  - Hypothèse 1 : ϵ_i suit une distribution normale $N(\mu, \sigma^2)$
  - Hypothèse 2 : l'espérance de ϵ_i est nulle : $\forall i, E(\epsilon_i) = 0$
  - Hypothèse 3 : la variance de ϵ_i est constante (homoscédasticité): $\forall i, V(\epsilon_i) = \sigma^2$
  - Hypothèse 4 : la covariance entre deux observations est nulle: $\forall i \neq j, Cov(\epsilon_i, \epsilon_j) = 0$, il n'y a pas d'autocorrélation des résidus, ils sont sériellement indépendants

Régression linéaire simple - Données Nelson-Plosser

- On considère les données Nelson-Plosser à partir de l'année 1909 (toutes les séries complètes)
- On utilise la fonction `spread` pour passer du format "long" au format "large"

```
nelplo1909 <- nelplo_tsbl %>%
  filter(index>=1909) %>%
  spread(key = key, value=value)
nelplo1909
```

```
## # A tibble: 80 x 15
##    index   cpi   emp gnp.capita gnp.def gnp.nom gnp.real int.rate    ip
##    <dbl> <dbl> <dbl>      <dbl>   <dbl>   <dbl>    <dbl>    <dbl> <dbl>
##  1  1909  3.33  10.5       7.16    3.37    10.4     4.76     3.77  2.30
##  2  1910  3.37  10.5       7.17    3.40    10.5     4.79     3.8   2.36
##  3  1911  3.37  10.5       7.18    3.39    10.5     4.81     3.9   2.32
##  4  1912  3.40  10.5       7.22    3.43    10.6     4.87     3.9   2.46
##  5  1913  3.39  10.6       7.21    3.44    10.6     4.88     4     2.53
##  6  1914  3.40  10.5       7.14    3.45    10.6     4.83     4.1   2.46
##  7  1915  3.41  10.5       7.12    3.48    10.6     4.82     4.15  2.61
##  8  1916  3.49  10.6       7.18    3.60    10.8     4.90     4.05  2.78
##  9  1917  3.65  10.6       7.18    3.81    11.0     4.91     4.05  2.79
## 10  1918  3.81  10.7       7.29    3.96    11.2     5.02     4.75  2.77
## # i 70 more rows
## # i 6 more variables: money.stock <dbl>, nom.wages <dbl>, real.wages <dbl>,
## #   stock.prices <dbl>, unemp <dbl>, vel <dbl>
```

Régression linéaire simple - Exemple 1

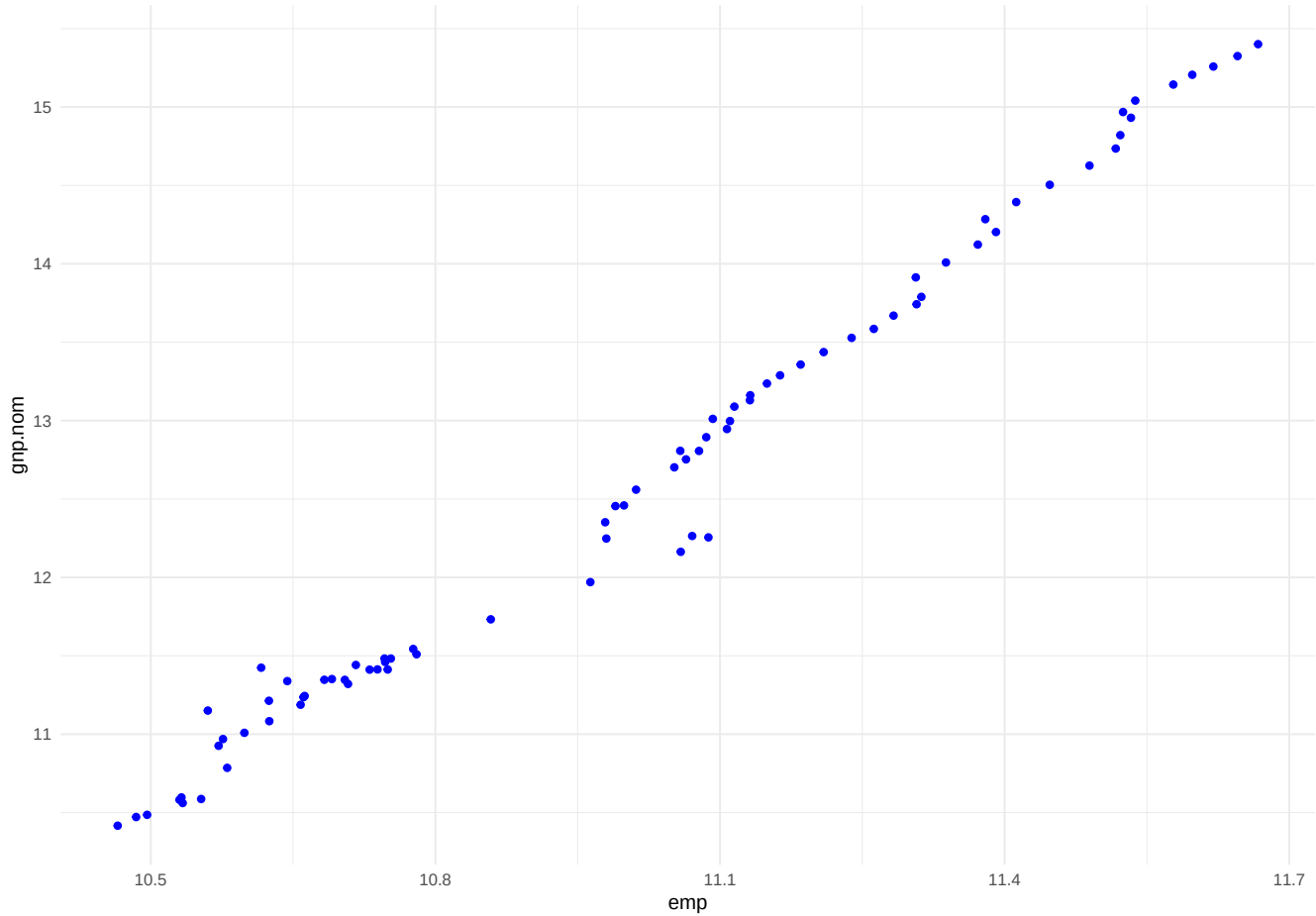
- On commence par un modèle bivarié:
  - variable expliquée y = gnp.nom (PNB, millions de dollars)
  - variable explicative x = emp (emploi, milliers de personnes)
- Note: on ne tient pas compte ici de l'aspect 'série temporelle' des données

```
nelplo1909 %>% select(index, gnp.nom, emp)
```

```
## # A tibble: 80 x 3
##    index gnp.nom   emp
##    <dbl>   <dbl> <dbl>
##  1  1909    10.4  10.5
##  2  1910    10.5  10.5
##  3  1911    10.5  10.5
##  4  1912    10.6  10.5
##  5  1913    10.6  10.6
##  6  1914    10.6  10.5
##  7  1915    10.6  10.5
##  8  1916    10.8  10.6
##  9  1917    11.0  10.6
## 10  1918    11.2  10.7
## # i 70 more rows
```

Régression linéaire simple - Visualisation

```
nelplo1909 %>%
  ggplot(aes(emp, gnp.nom)) +
    geom_point(colour="blue")
```



Régression linéaire simple - Estimation

- L'estimation des paramètres se fait avec la fonction `lm` (Linear Models)

```
mod1 <- lm(gnp.nom ~ emp, data=nelplo1909)
summary(mod1)
```

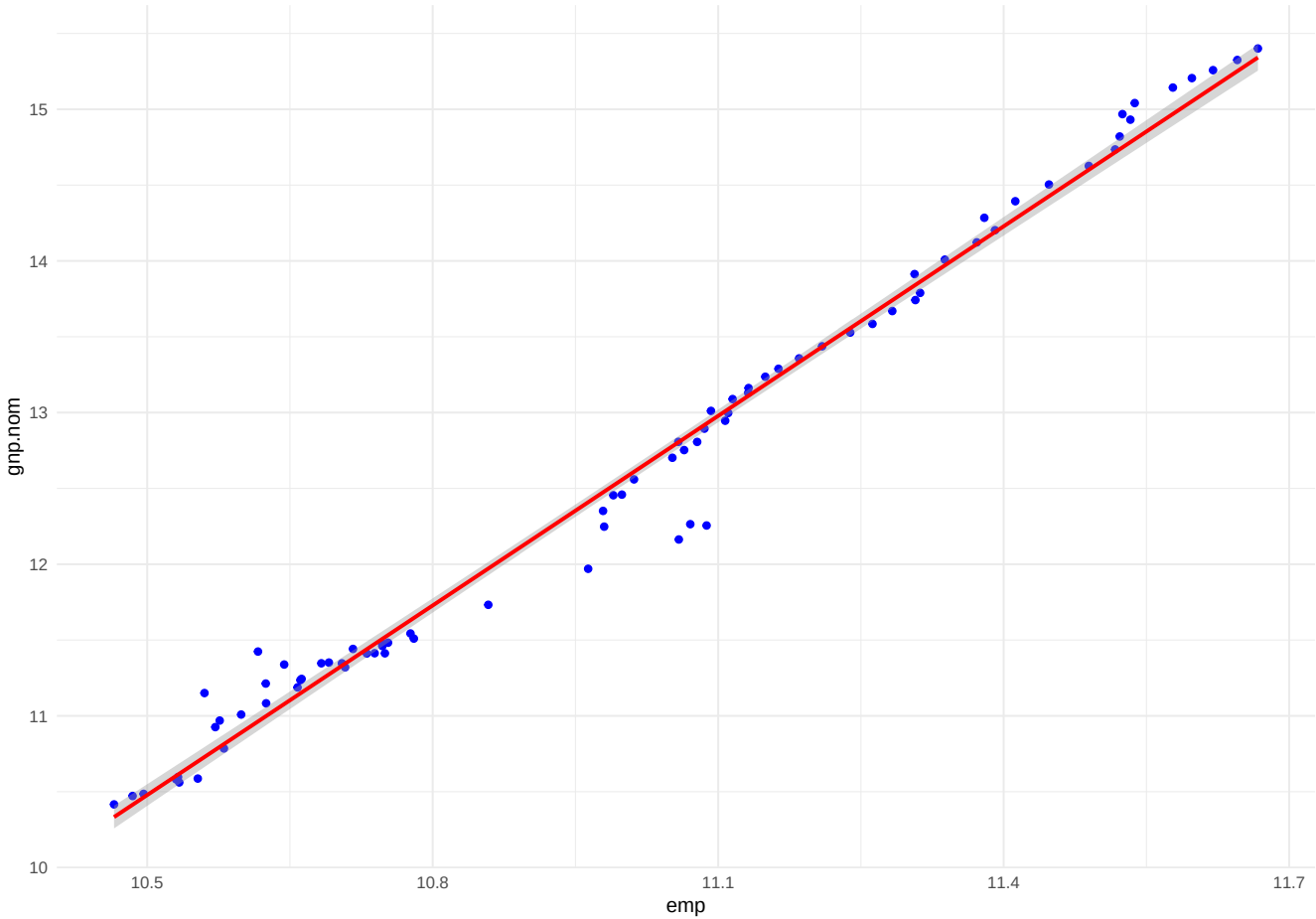
```
##
## Call:
## lm(formula = gnp.nom ~ emp, data = nelplo1909)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.67204 -0.06295  0.01739  0.08368  0.46104
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -33.27558    0.64326  -51.73   <2e-16 ***
## emp           4.16700    0.05841   71.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1814 on 78 degrees of freedom
## Multiple R-squared:  0.9849, Adjusted R-squared:  0.9847
## F-statistic:  5089 on 1 and 78 DF,  p-value: < 2.2e-16
```

Régression linéaire simple - Coefficients

- Un test avec H0: $\beta = 0$ est réalisé pour chacun des coefficients
- Le coefficient associé à la variable explicative `emp` (emploi) est statistiquement significatif (p-valeur < 0.001)
- La valeur de R^2 (variance expliquée) est élevée

Graphique de la droite de régression

```
ggplot(nelplo1909, aes(x=emp, y=gnp.nom)) +
  geom_point(colour="blue") +
  geom_smooth(method='lm', color="red")
```



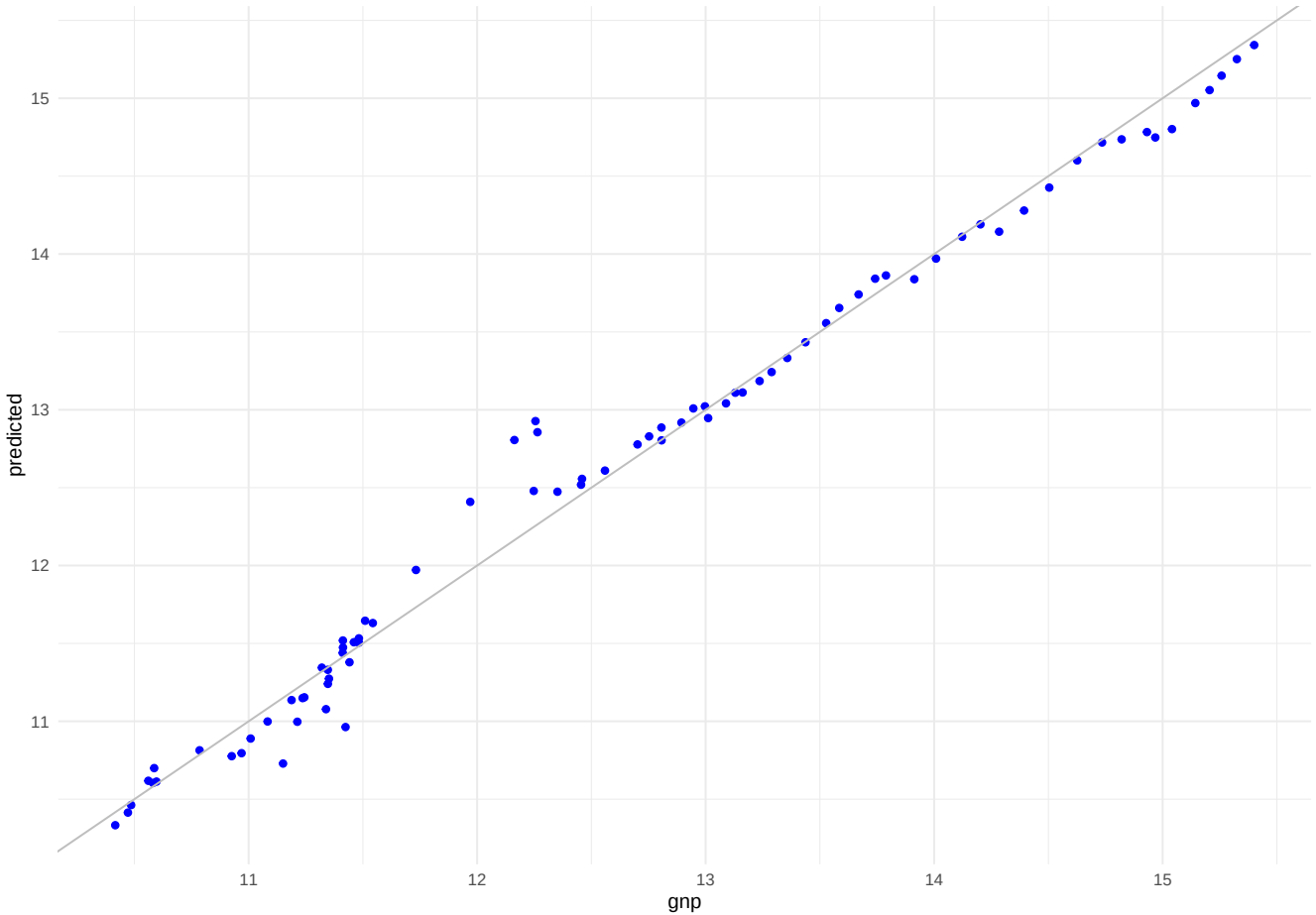
Valeurs observées, valeurs prédites, résidus

- Les valeurs prédites par le modèle se trouvent dans l'attribut `fitted.values`
- Les résidus représentent la différence valeur observée-valeur prédite, ils se trouvent dans l'attribut `residuals`

```
mod1_diag <- tibble(gnp=nelplo1909$gnp.nom, predicted=mod1$fitted.values, residual=mod1$residuals)
mod1_diag
```

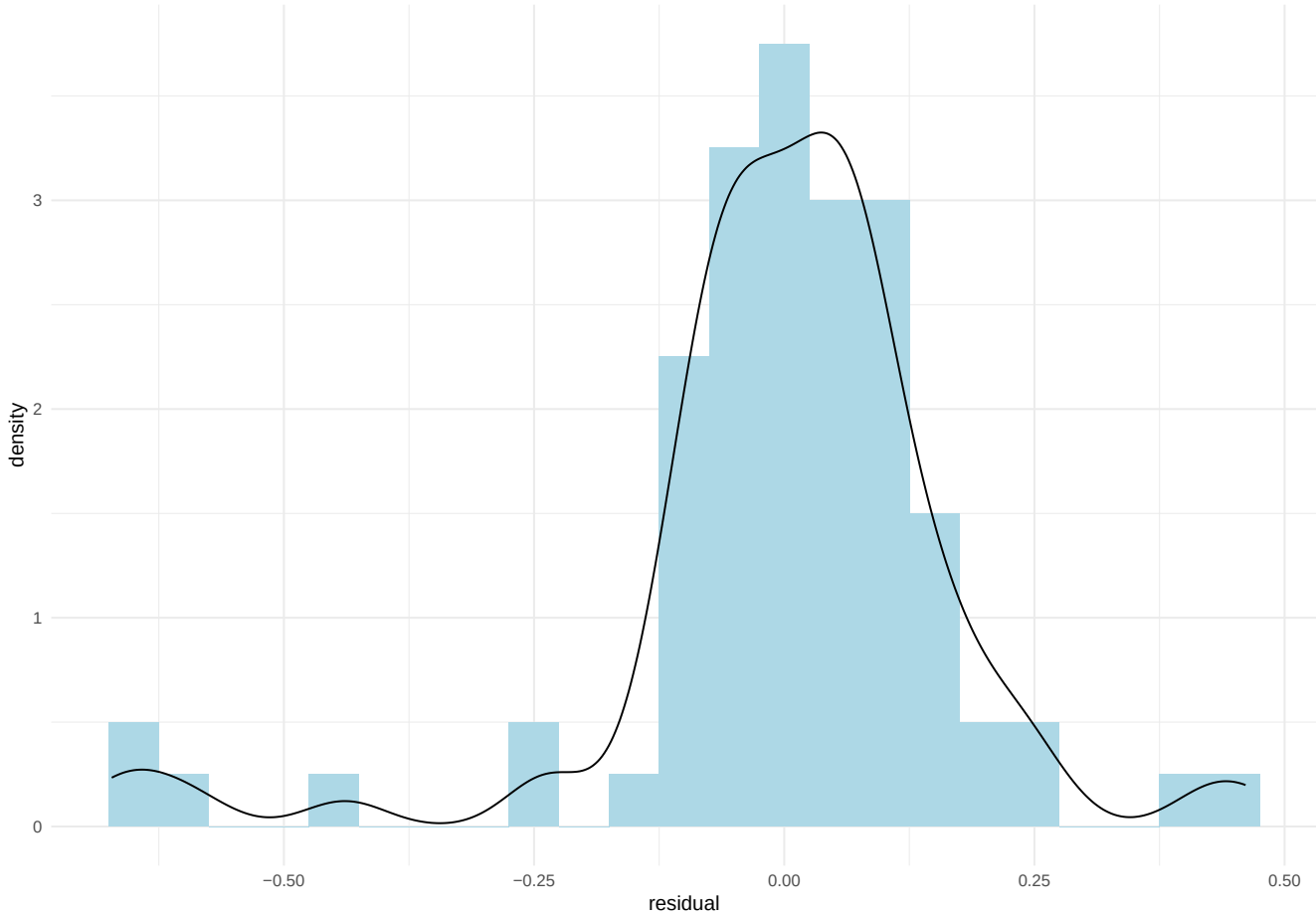
```
## # A tibble: 80 x 3
##       gnp predicted residual
##     <dbl>     <dbl>    <dbl>
##  1  10.4       10.3   0.0835
##  2  10.5       10.4   0.0577
##  3  10.5       10.5   0.0234
##  4  10.6       10.6  -0.0227
##  5  10.6       10.7  -0.113
##  6  10.6       10.6  -0.0572
##  7  10.6       10.6  -0.0163
##  8  10.8       10.8  -0.0287
##  9  11.0       10.9   0.119
## 10  11.2       11.2   0.0897
## # i 70 more rows
```

Graphique valeurs prédites x observées



Distribution des résidus

```
ggplot(mod1_diag, aes(x=residual)) +
  geom_histogram(aes(y = after_stat(density)), fill="lightblue", binwidth = 0.05) +
  geom_density(aes(x=residual))
```



Normalité des résidus

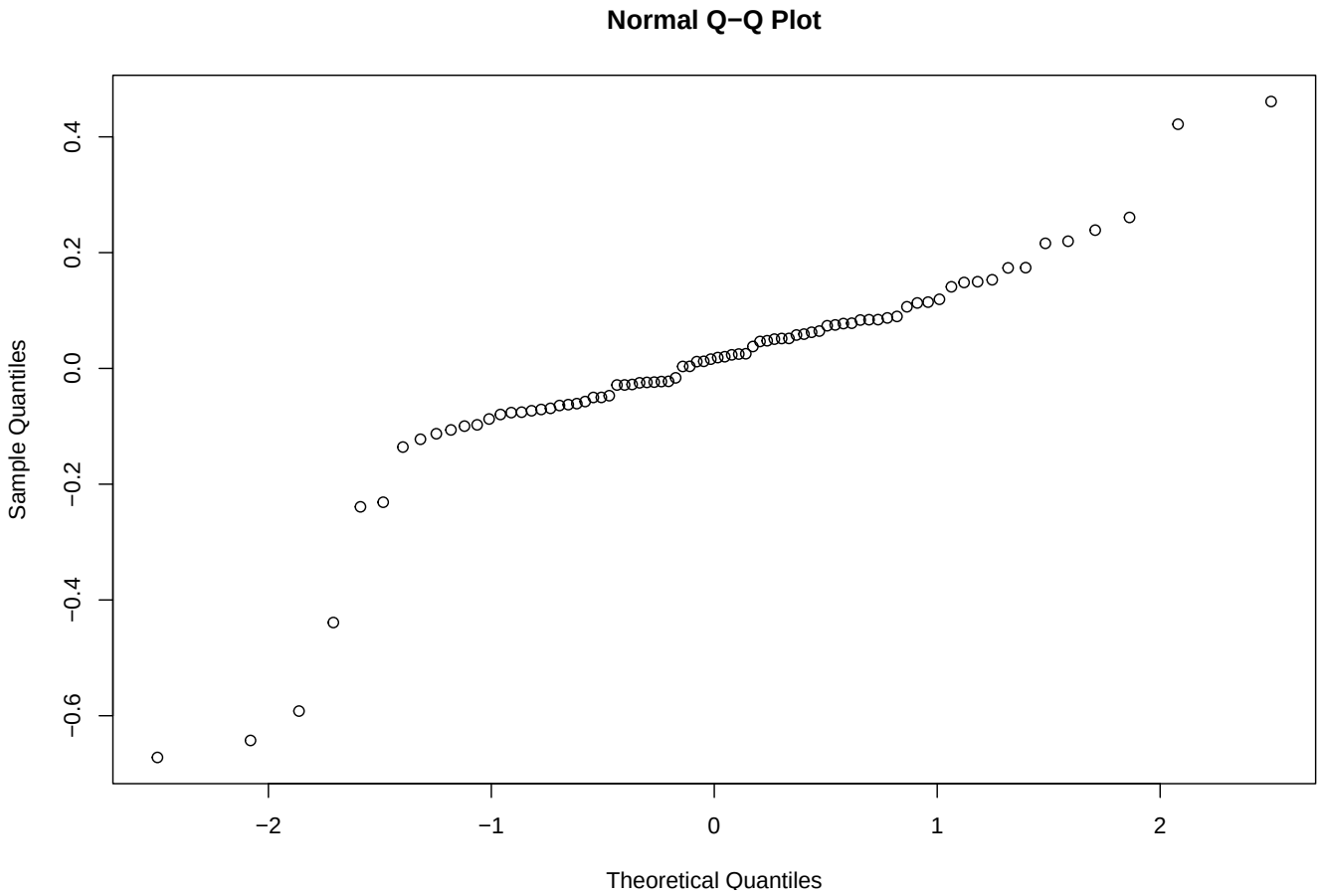
- Pour tester la normalité des résidus on utilise le test de **Shapiro-Wilk**
- Hypothèse H0: les résidus suivent une distribution normale
- La p-valeur est inférieure à 0.05, on rejette H0: les résidus ne sont pas distribués normalement

```
shapiro.test(mod1_diag$residual)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod1_diag$residual
## W = 0.84652, p-value = 1.239e-07
```

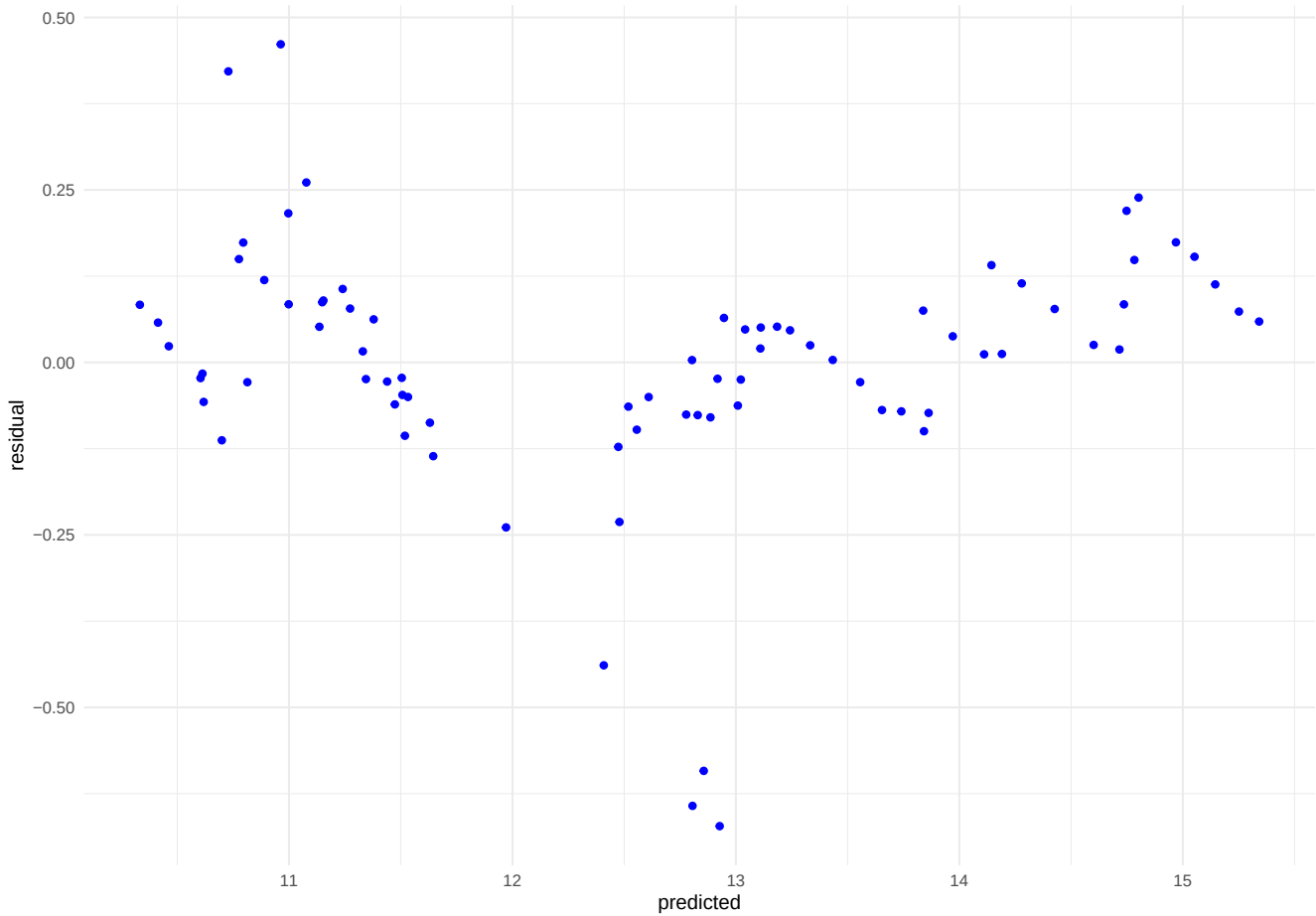
Normalité des résidus - Quantile-Quantile plot

```
qqnorm(mod1_diag$residual)
```



Résidus vs valeurs prédites

```
ggplot(mod1_diag) +
  geom_point(aes(x=predicted, y=residual), colour="blue")
```



Hétéroscedasticité

- L'homoscédasticité des résidus est une des hypothèses fondamentales du modèle OLS/MCO
- Homoscédasticité = la variance des résidus est constante:

$$Var(\epsilon_i) = \sigma^2$$

- Hétéroscédasticité = la variance des résidus n'est pas constante:

$$Var(\epsilon_i) = \sigma_i^2$$

- Si l'hypothèse d'homoscédasticité des résidus est violée, les tests d'hypothèse sur les coéfficients β du modèle OLS ne sont plus valides

Tester l'homoscédasticité

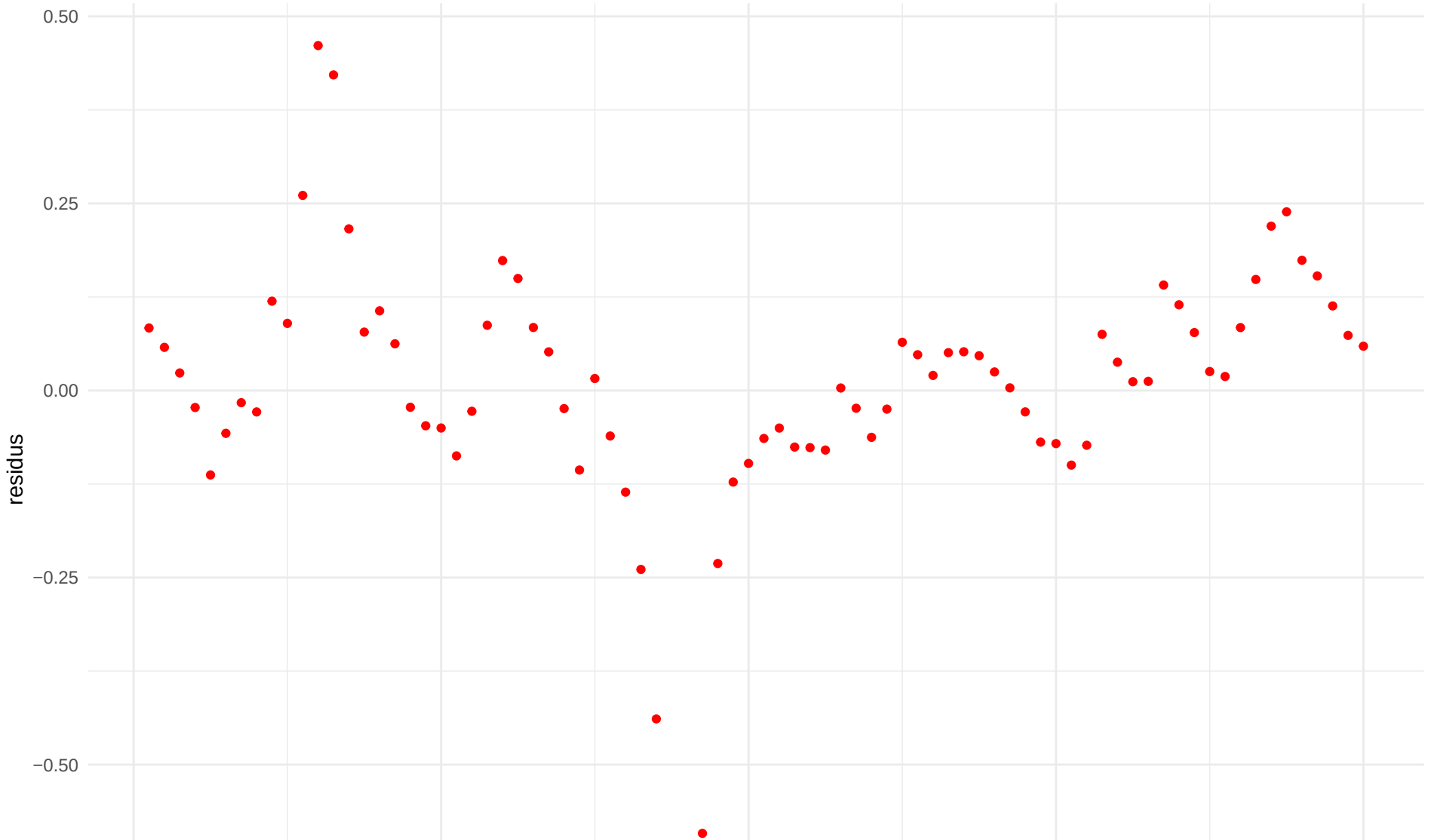
- Le test de **Breusch-Pagan-Godfrey** permet de tester l'homoscédasticité des résidus
- Hypothèse nulle (H0): homoscédasticité (les résidus ont une variance constante)
- On utilise la function `bptest()` de la librairie R `lmtest`
- La p-valeur du test est supérieure à 0.05, on accepte H0

```
library(lmtest)
bptest(mod1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod1
## BP = 0.06942, df = 1, p-value = 0.7922
```

Graphique de la série des résidus

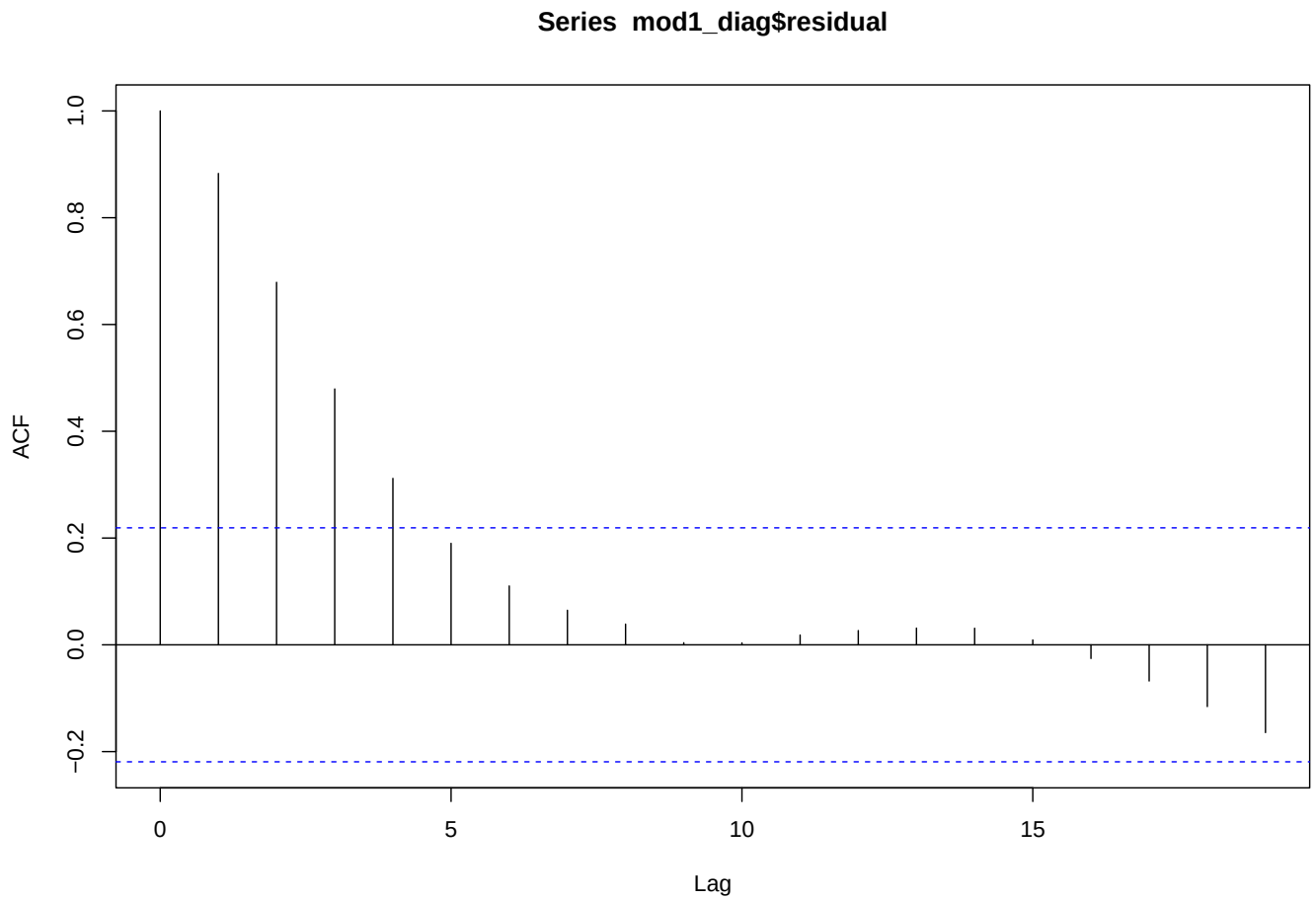
```
gdata <- tibble(index=1:nrow(mod1_diag), residus=mod1_diag$residual)
gdata %>% ggplot() + geom_point(aes(x=index, y=residus), color="red")
```



Autocorrélation des résidus - ACF

- On peut utiliser un graphique `acf` pour visualiser la fonction d'autocorrélation des résidus

```
acf(mod1_diag$residual)
```



Le test de Durbin-Watson

- Le test de **Durbin-Watson** est un test d'absence d'autocorrélation d'ordre 1 sur les résidus d'une régression linéaire
- On utilise la fonction `dwtest` de la librairie `lmtest`
- Hypothèse nulle (H0): pas d'autocorrélation des résidus

```
dwtest(mod1)
```

```
##
##  Durbin-Watson test
##
## data:  mod1
## DW = 0.23, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

- Ici on rejette l'hypothèse H0, il y a autocorrélation des résidus

Régression linéaire simple - Exemple 2

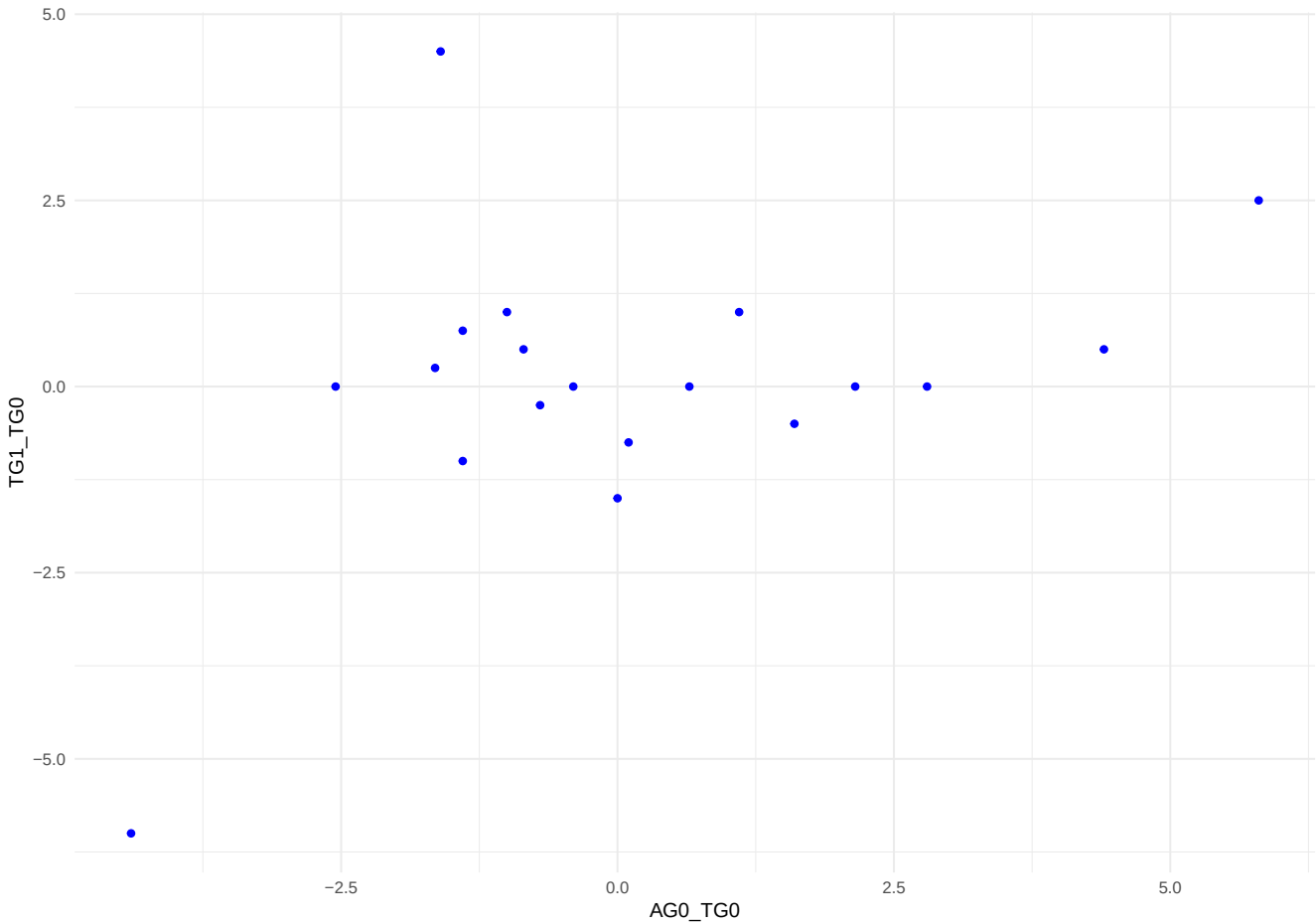
- Modèle de régression sur les données `growthofmoney`
- TG1.TG0: difference of current and preceding target for the growth rate of the money supply
- AG0.TG0: difference of actual growth rate and target growth rate for the preceding period

```
data("growthofmoney")
head(growthofmoney)
```

```
##         TG1.TG0 AG0.TG0
## 1970 Q2     0.0    -0.4
## 1970 Q3     1.0    -1.0
## 1970 Q4     1.0     1.1
## 1971 Q1     2.5     5.8
## 1971 Q2    -6.0    -4.4
## 1971 Q3     4.5    -1.6
```

Régression linéaire simple - Données growthofmoney

```
gdata <- tibble(AG0_TG0=growthofmoney[,"AG0.TG0"], TG1_TG0=growthofmoney[,"TG1.TG0"])
ggplot(gdata) + geom_point(aes(x=AG0_TG0, y=TG1_TG0), color="blue")
```



Régression linéaire simple - Données growthofmoney

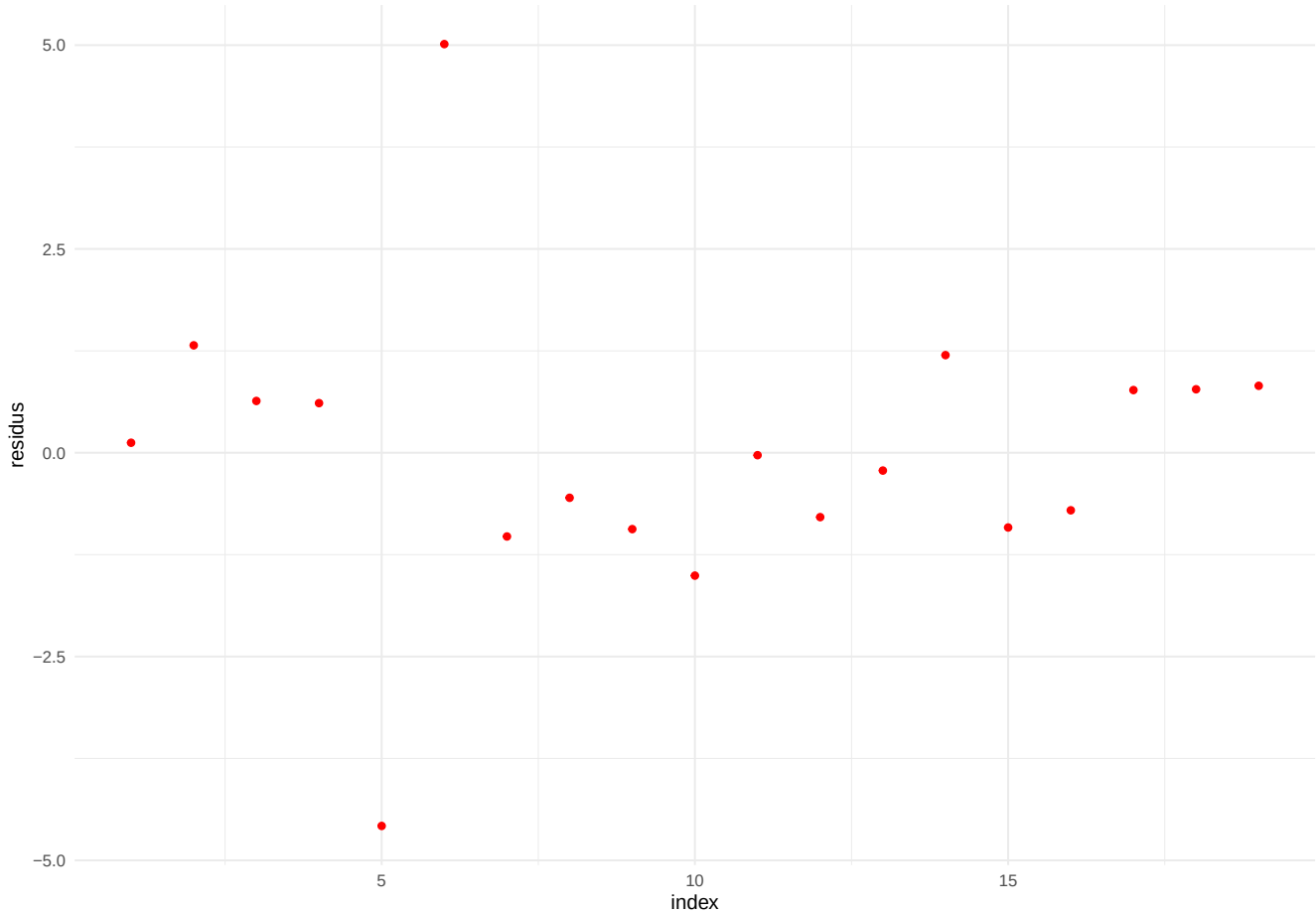
- Estimation du modèle (utilisé par Hetzel dans son article)

```
modelHetzel <- TG1.TG0 ~ AG0.TG0
gom.mod1 <- lm(modelHetzel, data=growthofmoney)
summary(gom.mod1)
```

```
##
## Call:
## lm(formula = modelHetzel, data = growthofmoney)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5779 -0.8534 -0.0299  0.7737  5.0125
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.007322   0.426070   0.017    0.986
## AG0.TG0     0.324858   0.179456   1.810    0.088 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.854 on 17 degrees of freedom
## Multiple R-squared:  0.1616, Adjusted R-squared:  0.1123
## F-statistic: 3.277 on 1 and 17 DF,  p-value: 0.08797
```

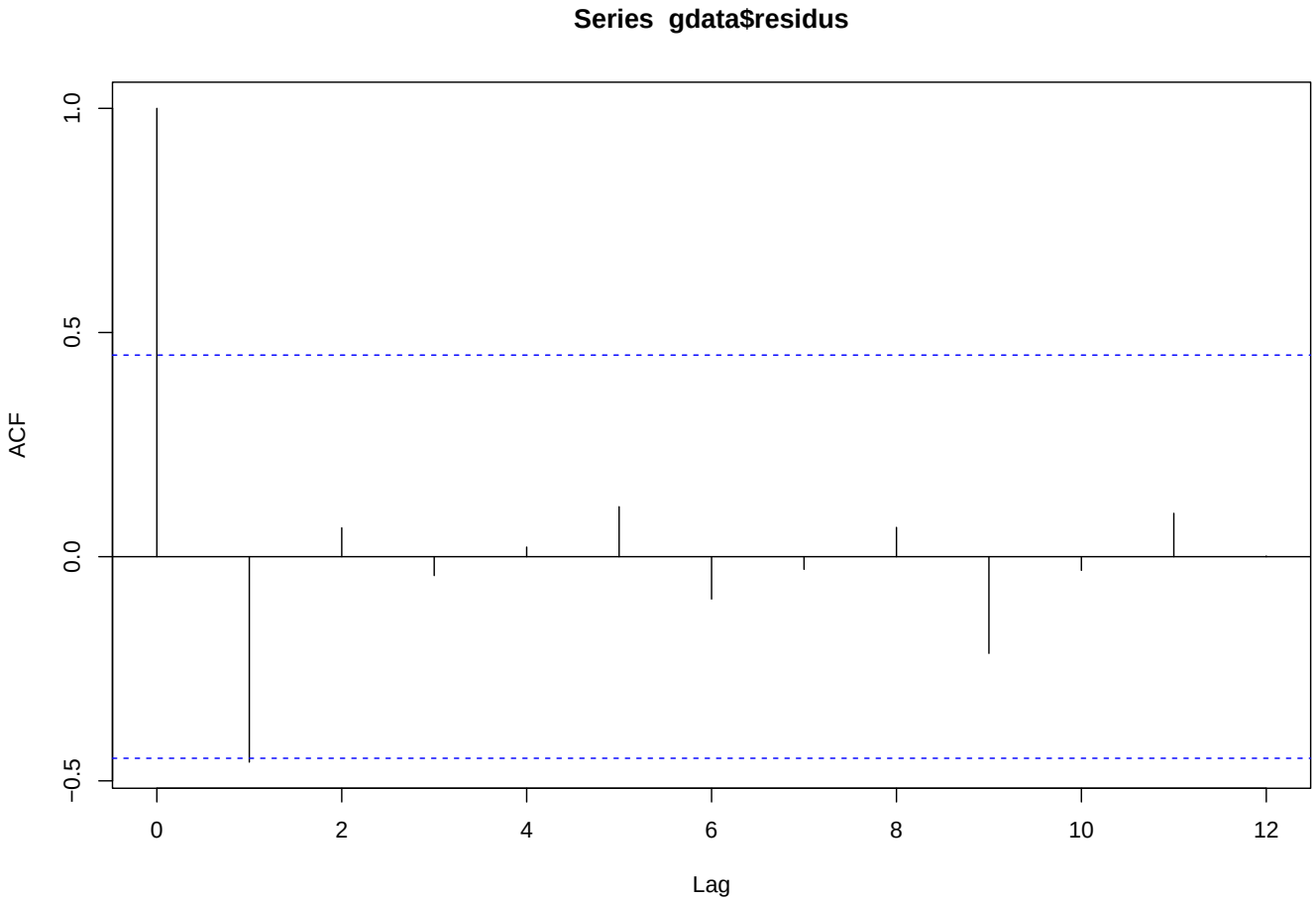
Graphique de la série des résidus - Données growthofmoney

```
gdata <- tibble(index=1:nrow(growthofmoney), residus=gom.mod1$residual)
gdata %>% ggplot() + geom_point(aes(x=index, y=residus), color="red")
```



Autocorrélation des résidus - Données growthofmoney

```
acf(gdata$residus)
```



Autocorrélation des résidus - Données growthofmoney

- La p-valeur du test de Durbin-Watson est largement supérieure à 0.05, on accepte H0, il n'y a pas d'autocorrélation des résidus

```
dwtest(modelHetzel, data=growthofmoney)
```

```
##
##  Durbin-Watson test
##
## data:  modelHetzel
## DW = 2.9046, p-value = 0.9839
## alternative hypothesis: true autocorrelation is greater than 0
```

Modèle de régression multivarié

- Modèle de régression multivarié = plusieurs variables indépendantes:

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \cdots + \beta_k \cdot x_k + \epsilon$$

Modèle de régression multivarié - Données Nelson-Plosser

- On souhaite prédire le PNB (`gnp.nom`, millions de dollars) par le taux de chômage, l'indice des prix et l'année
- On utilise la fonction `spread` pour passer du format "long" au format "large"

```
nelplo1909 <- nelplo_tsbl %>%
  filter(index>=1909) %>%
  spread(key = key, value=value)
regdata <- nelplo1909 %>%
  select(index, gnp.capita, unemp, cpi)
```

Modèle de régression multivarié - Estimation

```
mod2 <- lm(gnp.capita ~ index+unemp+cpi, data=regdata)
summary(mod2)
```

```
##
## Call:
## lm(formula = gnp.capita ~ index + unemp + cpi, data = regdata)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.178391 -0.034936 -0.001147  0.033163  0.181541
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.679e+01  1.719e+00 -15.583   <2e-16 ***
## index        1.775e-02  9.505e-04  18.680   <2e-16 ***
## unemp       -1.489e-01  1.231e-02 -12.095   <2e-16 ***
## cpi          4.099e-02  3.198e-02   1.282    0.204
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06824 on 76 degrees of freedom
## Multiple R-squared:  0.9779, Adjusted R-squared:  0.977
## F-statistic:  1119 on 3 and 76 DF,  p-value: < 2.2e-16
```

Sélection du modèle

- On élimine les variables dont le coefficient n'est pas significativement différent de 0 (p-valeur > 0.05)

```
mod3 <- lm(gnp.capita ~ index+unemp, data=regdata)
summary(mod3)
```

```
##
## Call:
## lm(formula = gnp.capita ~ index + unemp, data = regdata)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.184686 -0.035709 -0.007246  0.032664  0.184953
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.883e+01  6.466e-01  -44.59   <2e-16 ***
## index        1.890e-02  3.322e-04   56.88   <2e-16 ***
## unemp       -1.516e-01  1.219e-02  -12.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06852 on 77 degrees of freedom
## Multiple R-squared:  0.9774, Adjusted R-squared:  0.9768
## F-statistic:  1663 on 2 and 77 DF,  p-value: < 2.2e-16
```

Comparaison de modèles

- Pour comparer des modèles imbriqués, on peut utiliser un critère d'information (AIC ou BIC)

```
AIC(mod1, mod2, mod3)
```

```
## # A tibble: 3 x 2
##      df    AIC
##   <dbl>  <dbl>
## 1     3  -42.2
## 2     5 -197.
## 3     4 -197.
```

- On retient le modèle ayant le plus faible AIC, ici le modèle 3

Test de changement structurel

- Soit le modèle de régression linéaire standard:

$$y_i = x_i^T \beta_i + u_i \quad (i = 1, \ldots, N)$$

où y_i est l'observation de la variable dépendante et le vecteur $x_i = (1, x_{i2}, \ldots, x_{ik}$ l'obsevation des variables indépendantes au temps i

- Les tests de changement structurel proposent de tester l'hypothèse

$$H0 : \beta_i = \beta_0 \quad (i = 1, \ldots, N)$$

- Le test de **Chow** permet d'identifier un éventuel changement structurel dans les données au point fourni **a priori** par l'utilisateur
- Le rejet de l'hypotèse H0 signifie qu'un meilleur ajustement peut être obtenu avec deux droites de régression (i.e. les paramètres du modèle ne sont pas stables)

Test de Chow - Données growthofmoney

- On utilise la librairie strucchange (voir article)
- On reproduit ici l'exemple W. Krämer & H. Sonnberger (1986), The Linear Regression Model under Test. Heidelberg: Physica (voir p. 138): test de changement structurel au premier trimestre 1974
- La p-valeur est supérieure à 0.05, on rejette H0, il n'y a pas de changement structurel au premier trimestre 1974

```
sctest(modelHetzel, point=c(1973,4), data=growthofmoney, type="Chow")
```

```
##
##  Chow test
##
## data:  modelHetzel
## F = 0.37876, p-value = 0.6911
```

Test de changement structurel - Données Nelson-Plossser

- Pour la série du PNB nominal (log) on teste un changement structurel en 1933

```
rdata <- Nelson_Plosser[,c("year", "gnp.nom")]
rdata[,"logGNP"] = log(rdata[,"gnp.nom"])
rdata <- rdata[rdata$year>=1909,]
rdata <- ts(rdata, start=1909)
window(rdata, end=1916)
```

```
## Time Series:
## Start = 1909
## End = 1916
## Frequency = 1
##      year  gnp.nom   logGNP
## 1909 1909 10.41631 2.343373
## 1910 1910 10.47164 2.348670
## 1911 1911 10.48570 2.350013
## 1912 1912 10.58152 2.359109
## 1913 1913 10.58658 2.359588
## 1914 1914 10.56101 2.357169
## 1915 1915 10.59663 2.360536
## 1916 1916 10.78519 2.378174
```

Test de changement structurel - Données Nelson-Plosser

- La p-valeur du test est inférieure à 0.05, on rejette H0

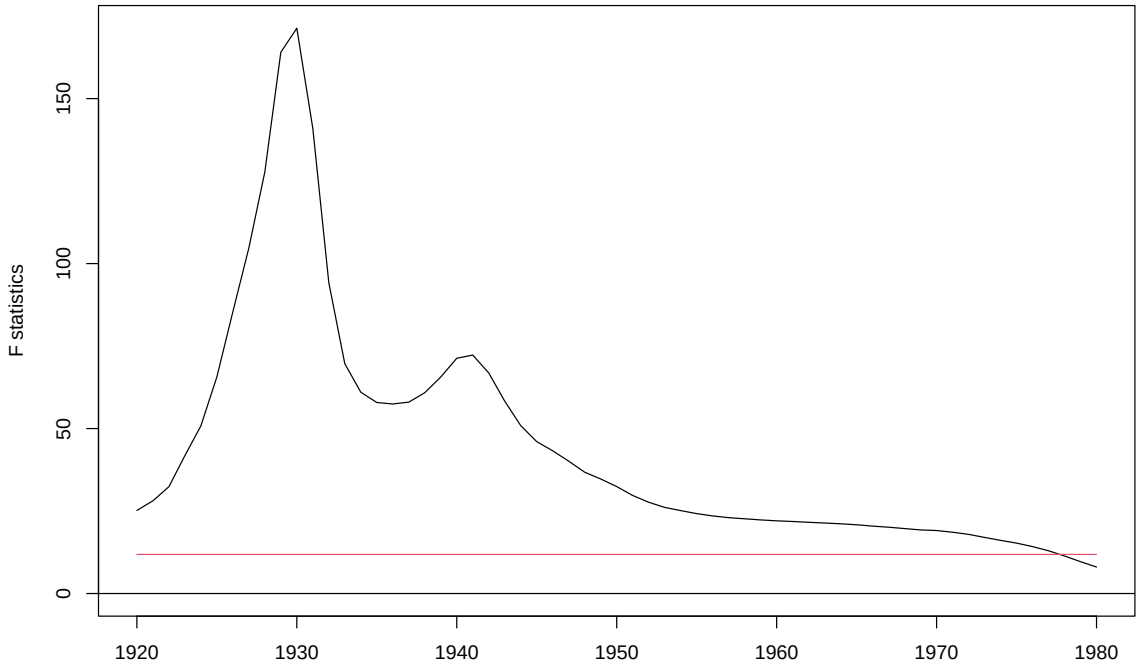
```
library(strucchange)
model <- logGNP ~ year
sctest(model, point=c(1933,1), data=rdata, type="Chow")
```

```
##
##  Chow test
##
## data:  model
## F = 34.825, p-value = 1.841e-11
```

Test de changement structurel - Extension

- La librairie `strucchange` propose une extension du test de Chow: le changement structurel est recherché sur un intervalle
- La statistique F est calculée pour chacun des points de l'intervalle
- On rejette H0 pour des valeurs élevées de F (supérieure au seuil représenté par la ligne rouge)

```
fs <- Fstats(model, from = c(1920,1), to = c(1980,1), data = rdata)
plot(fs)
```



Test de changement structurel - Extension

Exercice

- Analysez les résidus de la régression

```
mod3 <- lm(gnp.capita ~ index+unemp, data=regdata)
```

- Les résidus sont-ils normalement distribués ?
- Sont-ils autocorrélés ?
- Leur variance est-elle constante ?

Section 7

Décomposition d'une série temporelle (moyennes mobiles)

Composants d'une série temporelle

- Une série temporelle peut être décomposée en trois éléments:
  - Tendence
  - Saisonnalité
  - Résidus

Décomposition avec la librairie `feasts`

- La librarie `feasts` propose deux méthodes de décomposition (classique et STL)
- La décomposition classique utilise les moyennes mobiles, la saisonalité peut être additive ou multiplicative

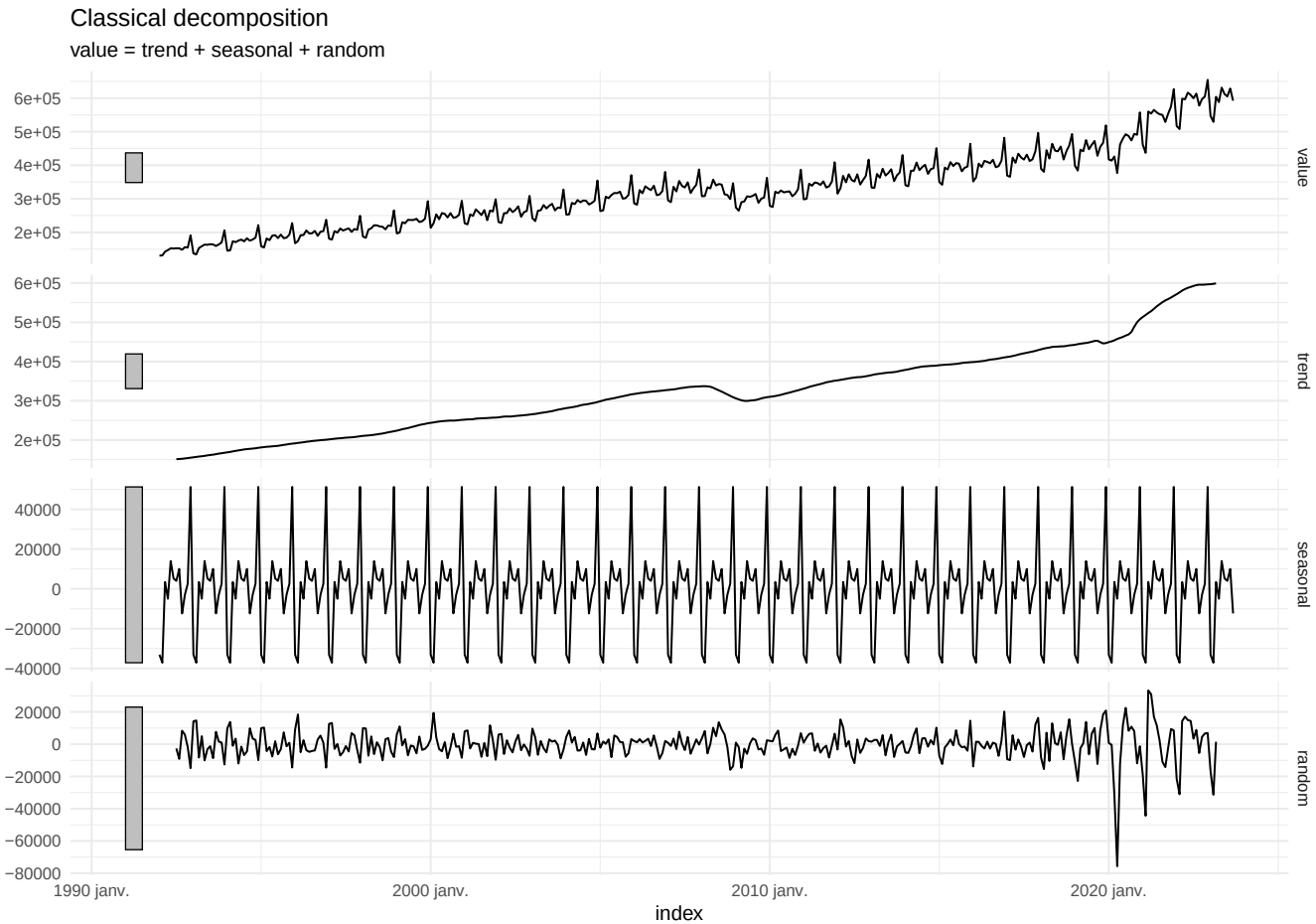
```
dcmp <- retail_tsbl %>%
  model(classical_decomposition(value))
components(dcmp)
```

```
## # A tibble: 381 x 7
##    .model                    index  value   trend seasonal random season_adjust
##    <chr>                     <mth>  <dbl>   <dbl>    <dbl>  <dbl>         <dbl>
##  1 classical_decomposit~ 1992 janv. 130683     NA   -33112.     NA        163795.
##  2 classical_decomposit~ 1992 févr. 131244     NA   -37134.     NA        168378.
##  3 classical_decomposit~  1992 mars 142488     NA     3459.     NA        139029.
##  4 classical_decomposit~ 1992 avril 147175     NA    -4965.     NA        152140.
##  5 classical_decomposit~   1992 mai 152420     NA    13966.     NA        138454.
##  6 classical_decomposit~  1992 juin 151849     NA     5222.     NA        146627.
##  7 classical_decomposit~ 1992 juil. 152586 151200.    4004. -2619.        148582.
##  8 classical_decomposit~  1992 août 152476 151599.   10017. -9140.        142459.
##  9 classical_decomposit~ 1992 sept. 148158 152172.  -12342.  8329.        160500.
## 10 classical_decomposit~  1992 oct. 155987 153087.   -2997.  5897.        158984.
## # i 371 more rows
```

Décomposition avec la librairie `feasts` - Graphique

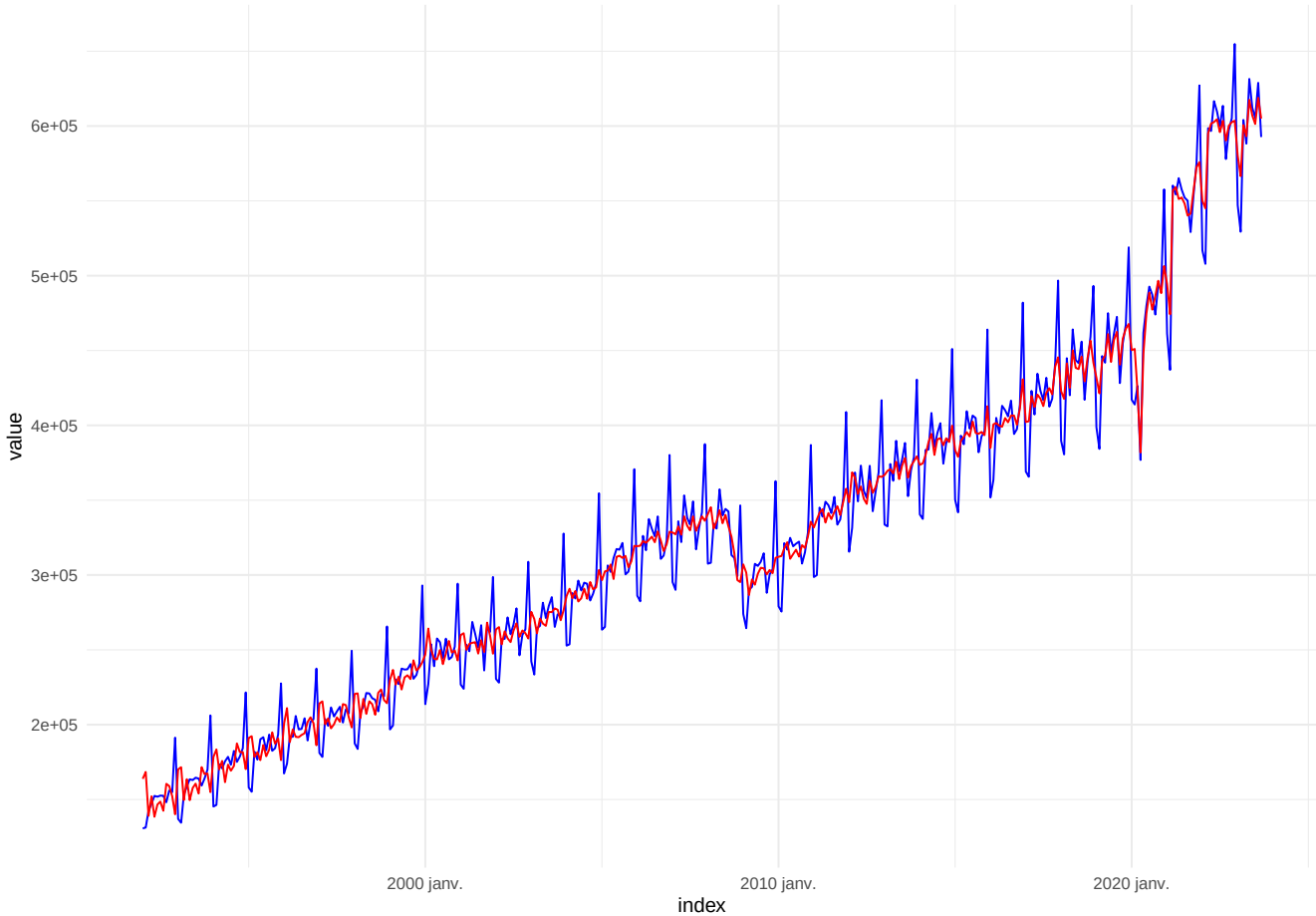
```
components(dcmp) %>% autoplot()
```

```
## Warning: Removed 6 rows containing missing values (`geom_line()`).
```



Décomposition avec la librairie `feasts` - Graphique

```
components(dcmp) %>% ggplot() +
  geom_line(aes(x=index,y=value), colour="blue") +
  geom_line(aes(x=index,y=season_adjust), colour="red")
```



Section 8

Transformation des données et stabilisation de la variance

Tendance linéaire

- Une série temporelle dont l'évolution est une fonction **déterministe** du temps est non-sationnaire
- Une série dont l'évolution autour d'une fonction déterministe du temps est stationnaire est dite stationnaire à une tendance près (trend stationary)
- On peut décrire une tendance linéaire par le modèle

$$y_t = \beta_0 + \beta_1 \cdot t + \epsilon_t$$

- y_t est non-stationnaire si $\beta 1 \neq 0$, la moyenne de y_t est

$$\mu_t = \beta_0 + \beta_1 \cdot t$$

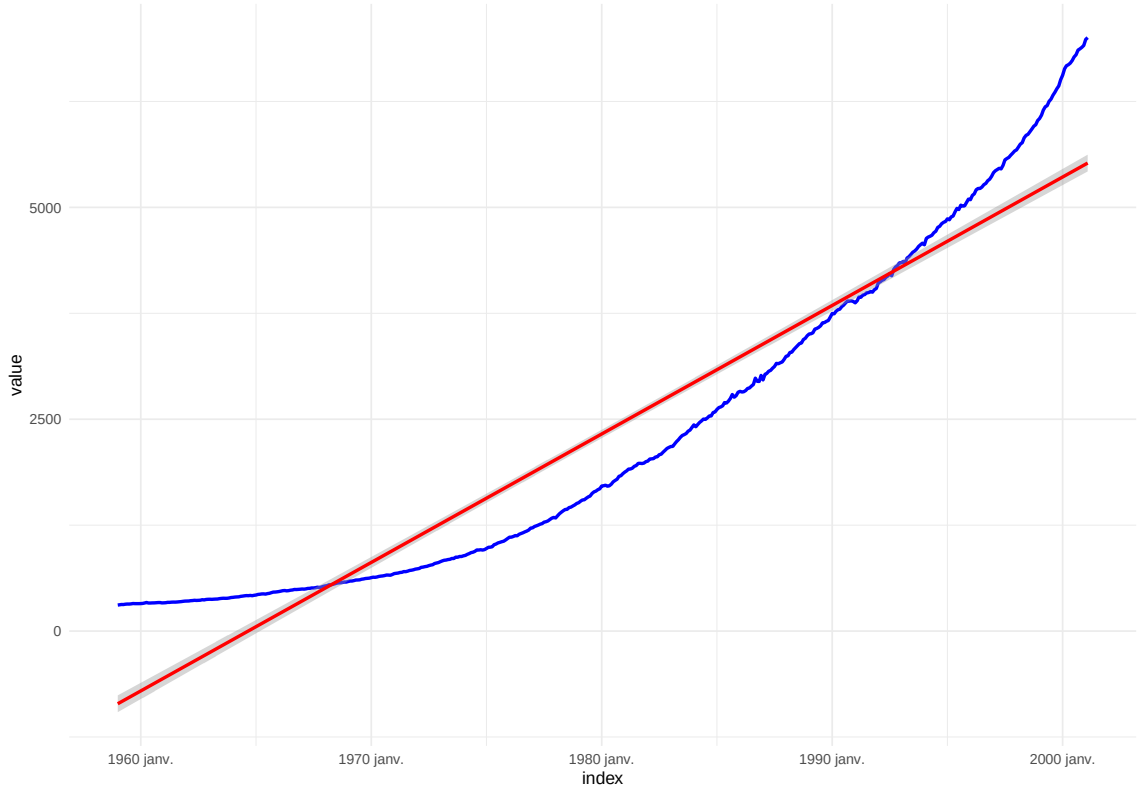
Tendance linéaire - Test

- On peut tester l'hypothèse $\beta 1 \neq 0$ (existence d'une tendance) en ajustant un modèle MCO sur les données (cette approche est valide si les erreurs ϵ_t sont un bruit blanc non corrélé)
- On peut ajuster un modèle pour estimer β_0 et β_1 puis analyser les résidus comme un processus stationnaire $\epsilon_t = y_t - \beta_0 - \beta_1 \cdot t$

Tendance linéaire - Exemple

- Données `USIncExp`, dépenses aggrégées de consommation en millions de dollars

```
USIncExp %>% as_tsibble() %>% filter(key=='expenditure') %>%
  ggplot(aes(x = index, y = value)) +
  geom_line(color = "blue", size = 1) +
  geom_smooth(method='lm', color="red")
```



Tendance linéaire - Estimation du modèle

- Estimation du modèle par MCO

```
regdata <- USIncExp %>% as_tsibble() %>% filter(key=='expenditure')
summary(lm(value ~ index, data=regdata))
```

```
##
## Call:
## lm(formula = value ~ index, data = regdata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -688.01 -507.37  -97.57  411.41 1480.36
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.101e+02  3.277e+01   24.72   <2e-16 ***
## index       4.151e-01  5.686e-03   73.00   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 568.7 on 504 degrees of freedom
## Multiple R-squared:  0.9136, Adjusted R-squared:  0.9134
## F-statistic:  5329 on 1 and 504 DF,  p-value: < 2.2e-16
```

- Le coefficient associé au temps (index) est significativement différent de 0 (p-valeur < 0.05)

Tendance exponentielle

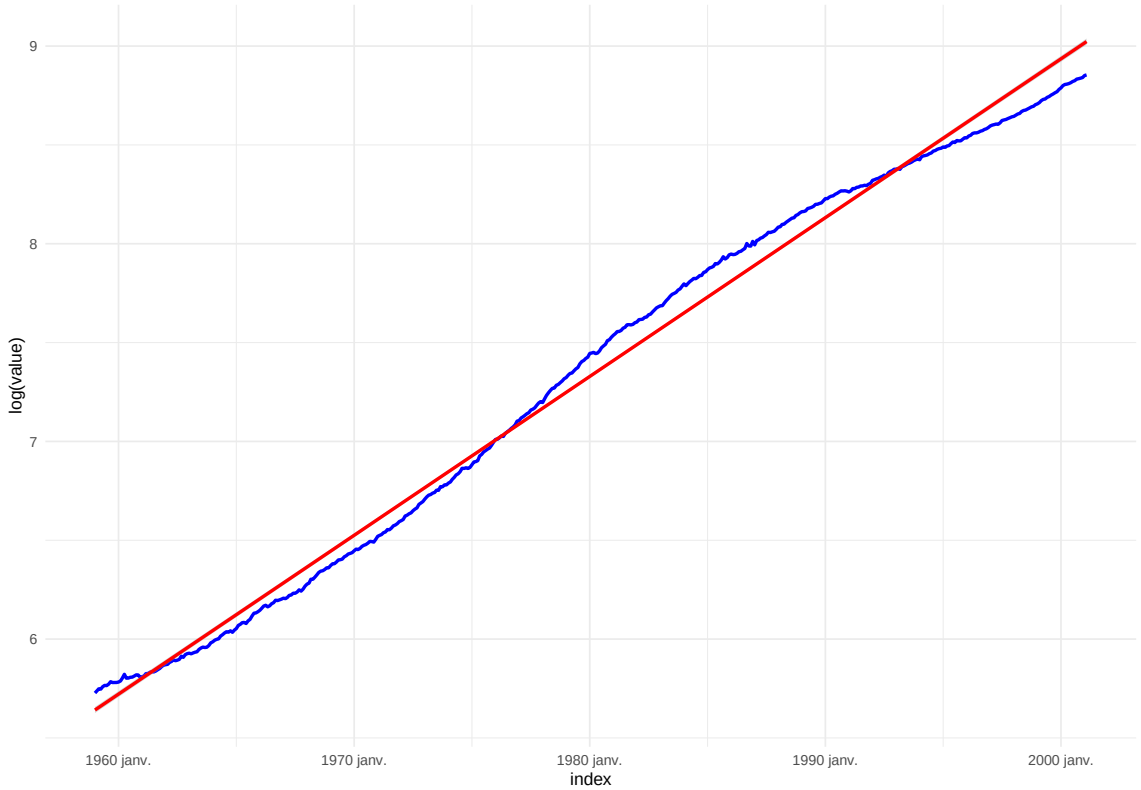
- Si la tendance est exponentielle, on peut ramener à une tendance linéaire en utilisant le log:

$$\log(y_t) = \beta_0 + \beta_1 \cdot t + \epsilon_t$$

- Note: β_1 dans le modèle à tendance exponentielle est le taux de croissance annuel moyen (si t est exprimé en années)

Tendance exponentielle - Echelle logarithmique

```
USIncExp %>% as_tsibble() %>% filter(key=='expenditure') %>%
  ggplot(aes(x = index, y = log(value))) +
  geom_line(color = "blue", size = 1) +
  geom_smooth(method='lm', color="red")
```



Tendance exponentielle - Modèle linéaire

```
regdata <- USIncExp %>% as_tsibble() %>%
  filter(key=='expenditure') %>%
  mutate(value=log(value))
summary(lm(value ~ index, data=regdata))
```

```
##
## Call:
## lm(formula = value ~ index, data = regdata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.16858 -0.07288 -0.01324  0.09148  0.15056
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.525e+00  5.134e-03    1271   <2e-16 ***
## index       2.200e-04  8.908e-07     247   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08909 on 504 degrees of freedom
## Multiple R-squared:  0.9918, Adjusted R-squared:  0.9918
## F-statistic: 6.099e+04 on 1 and 504 DF,  p-value: < 2.2e-16
```

Utilisation de la transformation logarithmique

- On utilise souvent le logarithme (naturel) d'une série temporelle pour l'analyse
- Par exemple dans l'article original de Nelson et Plosser, toutes les séries à l'exception de la série `stock.price` (prix des actions) sont transformées en log: "*The tendency of economic time series to exhibit variation that increases in mean and dispersion in proportion to absolute level motivates the transformation to natural logs and the assumption that trends are linear in the transformed data.*"
- L'utilisation du log d'une série:
  - diminue donc l'hétéroscédasticité (stabilisation de la variance)
  - transforme une tendance exponentielle en tendance linéaire

Désaisonnalisation

- De nombreuses séries économiques présentent des comportements périodiques, rendant difficile la comparaison de deux instants successifs
- Cela peut être le cas particulièrement lorsque la série est trimestrielle ou mensuelle (données `retail`)
- Le recours à une **désaisonnalisation** permet d'obtenir des séries dites corrigées des variations saisonnières (CVS)

Désaisonnalisation par la régression linéaire

- On inclus la saisonnalité dans un modèle linéaire tendenciel avec des variables `dummy` (0 ou 1) représentant les mois, trimestres, etc US.
- Par exemple pour des trimestres on ajoute $4 - 1 = 3$ variables `dummy` (modèle avec constante):

$$y_t = \beta_0 + \delta_1 \cdot Q1_t + \delta_2 \cdot Q2_t + \delta_3 \cdot Q3_t + \beta_1 \cdot t + \epsilon_t$$

Désaisonnalisation - Exemple

- Données `retail` sur le commerce de détail
- On créé une variable catégorielle (factor) pour le mois
- Le temps est un index t de 1 à 381 (nombre de mois dans la série)
- Dans la formule on ajoute -1 pour un modèle sans constante (intercept)

```
retail$mois <- factor(month(retail$DATE), labels=month(1:12, label=TRUE))
retail$t <- c(1:nrow(retail))
head(retail)
```

```
## # A tibble: 6 x 4
##   DATE       RSXFSN mois      t
##   <date>      <dbl> <fct> <int>
## 1 1992-01-01 130683 janv      1
## 2 1992-02-01 131244 févr      2
## 3 1992-03-01 142488 mars      3
## 4 1992-04-01 147175 avril     4
## 5 1992-05-01 152420 mai       5
## 6 1992-06-01 151849 juin      6
```

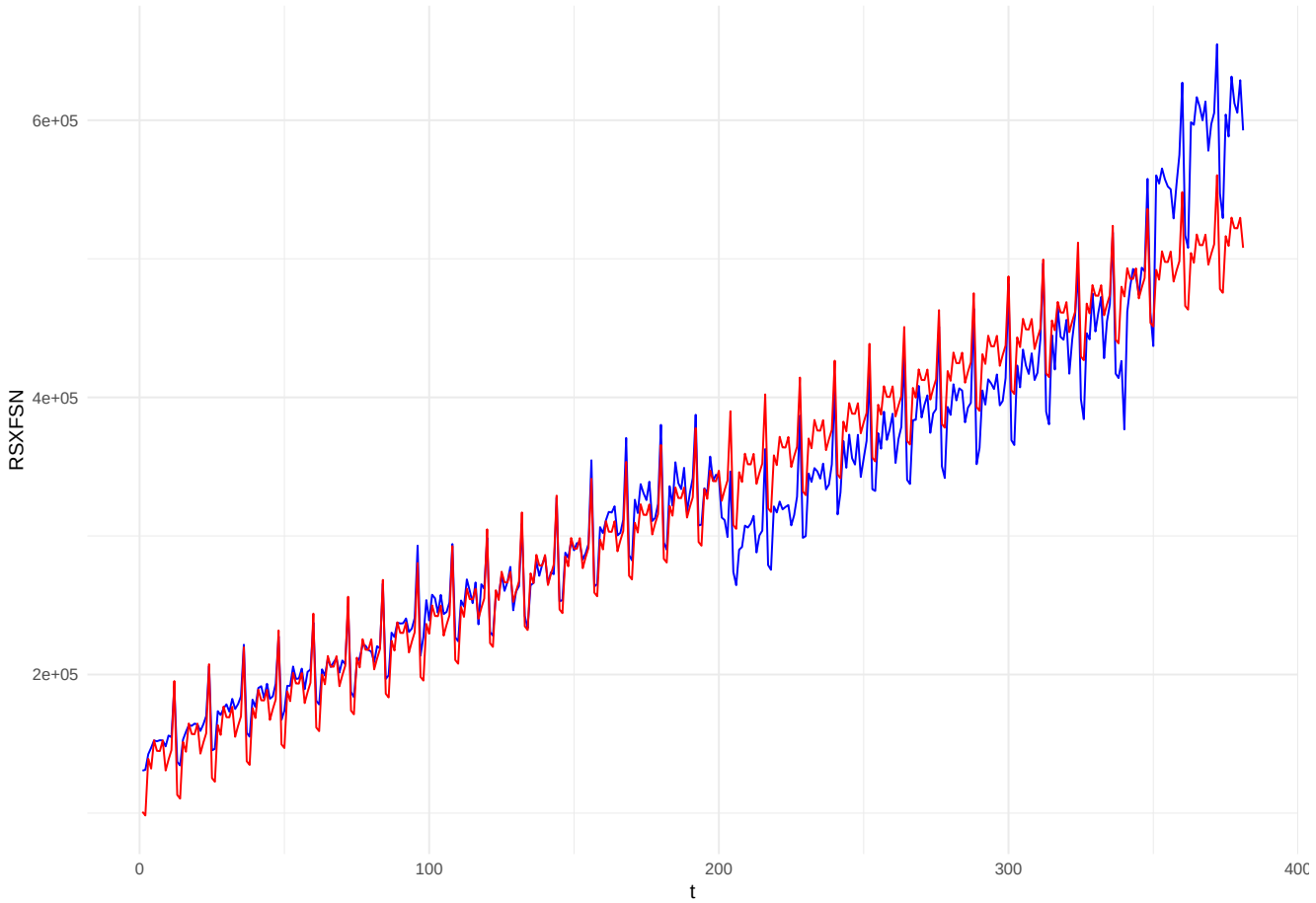
Désaisonnalisation - Résultat

```
modst <- lm(RSXFSN ~ t+ mois-1, data=retail)
summary(modst)
```

```
##
## Call:
## lm(formula = RSXFSN ~ t + mois - 1, data = retail)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -95922 -22769    753  10775 101781
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## t           1013.92      14.43   70.25   <2e-16 ***
## moisjanv   99954.45    6105.34   16.37   <2e-16 ***
## moisfévr   96272.09    6111.74   15.75   <2e-16 ***
## moismars  136132.45    6118.16   22.25   <2e-16 ***
## moisavril 128055.40    6124.61   20.91   <2e-16 ***
## moismai   147466.57    6131.09   24.05   <2e-16 ***
## moisjuin  138825.03    6137.59   22.62   <2e-16 ***
## moisjuil  137696.67    6144.12   22.41   <2e-16 ***
## moisaoût  144302.31    6150.68   23.46   <2e-16 ***
## moissept  121614.92    6157.26   19.75   <2e-16 ***
## moisoct   128425.61    6203.09   20.70   <2e-16 ***
## moisnov   134270.91    6209.48   21.62   <2e-16 ***
## moisdéc   183151.15    6215.90   29.46   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 30980 on 368 degrees of freedom
## Multiple R-squared:  0.9923, Adjusted R-squared:  0.992
## F-statistic:  3652 on 13 and 368 DF,  p-value: < 2.2e-16
```

Désaisonnalisation avec la régression linéaire: Prédiction

```
retail$prediction <- predict.lm(modst)
ggplot(retail)+
  geom_line(mapping=aes(x=t,y=RSXFSN),color="blue")+
  geom_line(mapping=aes(x=t,y=prediction), color="red")
```



Désaisonnalisation avec la régression linéaire: Calcul

- Creation des variables indicatrices pour le mois (dummies)

```
retail_1992_2022 <- retail %>% filter(year(DATE)<2023)
annees = nrow(retail_1992_2022)/12
t=1:annees

for (i in 1:12)
{
  su=rep(0,times=12)
  su[i]=1
  s=rep(su,times=annees)
  assign(paste("s",i,sep=""),s)
}
cbind(retail_1992_2022[,"RSXFSN"],s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12)[1:12,]
```

```
##              s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
##  [1,] 130683  1  0  0  0  0  0  0  0  0   0   0   0
##  [2,] 131244  0  1  0  0  0  0  0  0  0   0   0   0
##  [3,] 142488  0  0  1  0  0  0  0  0  0   0   0   0
##  [4,] 147175  0  0  0  1  0  0  0  0  0   0   0   0
##  [5,] 152420  0  0  0  0  1  0  0  0  0   0   0   0
##  [6,] 151849  0  0  0  0  0  1  0  0  0   0   0   0
##  [7,] 152586  0  0  0  0  0  0  1  0  0   0   0   0
##  [8,] 152476  0  0  0  0  0  0  0  1  0   0   0   0
##  [9,] 148158  0  0  0  0  0  0  0  0  1   0   0   0
## [10,] 155987  0  0  0  0  0  0  0  0  0   1   0   0
## [11,] 154824  0  0  0  0  0  0  0  0  0   0   1   0
## [12,] 191347  0  0  0  0  0  0  0  0  0   0   0   1
```

Désaisonnalisation avec la régression linéaire: Calcul

- Pour obtenir les données CVS on extrait les coefficients

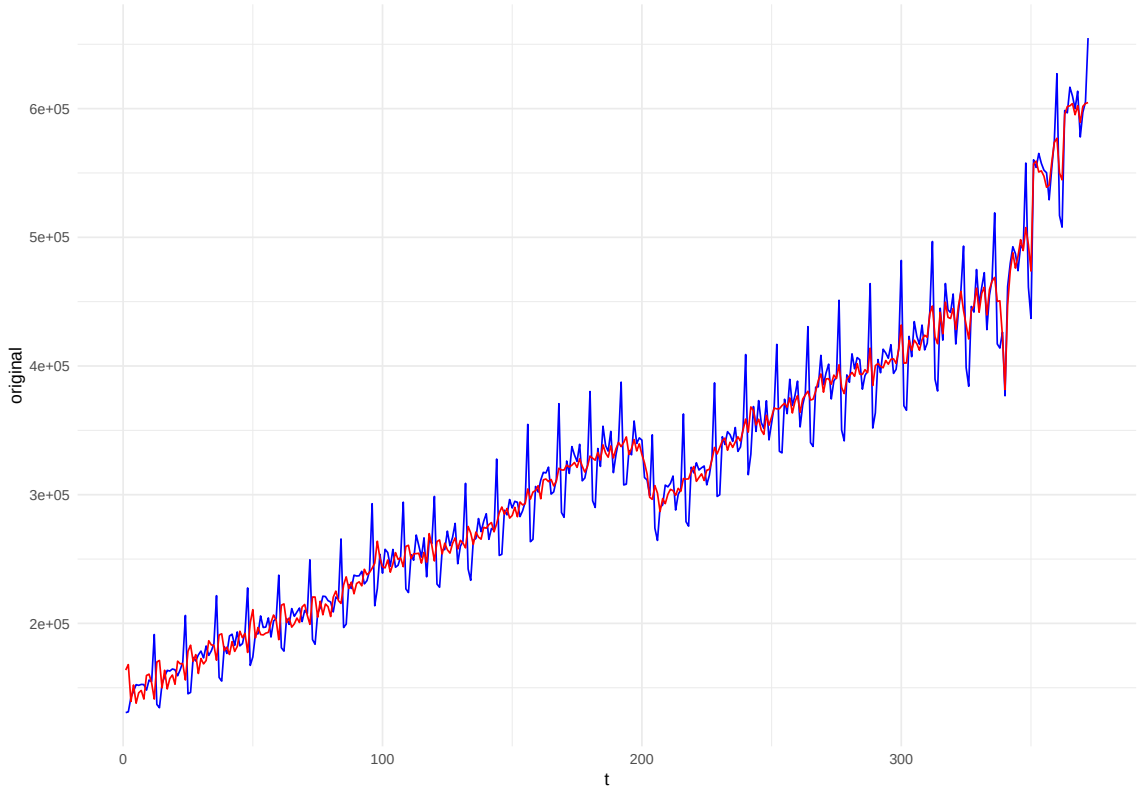
```
coefst <- modst$coefficients
coefst
```

```
##          t   moisjanv   moisfévr   moismars  moisavril    moismai   moisjuin
##   1013.922  99954.447  96272.087 136132.447 128055.400 147466.572 138825.026
##   moisjuil   moisaoût   moissept    moisoct    moisnov    moisdéc
## 137696.666 144302.307 121614.917 128425.611 134270.915 183151.155
```

```
a <- mean(coefst[2:13])
b <- coefst[1]
c <- coefst[2:13]-mean(coefst[2:13])
y_cvs <- retail_1992_2022$RSXFSN-(c[1]*s1+c[2]*s2+c[3]*s3+c[4]*s4+c[5]*s5+c[6]*s6+c[7]*s7+c[8]*s8+c[9]*s9+c[10]
```

Désaisonnalisation avec la régression linéaire: Données CVS

```
gdata <- tibble(t=retail_1992_2022$t, original=retail_1992_2022$RSXFSN, cvs = y_cvs)

ggplot(gdata)+
  geom_line(mapping=aes(x=t,y=original),color="blue")+
  geom_line(mapping=aes(x=t,y=cvs), color="red")
```



Désaisonnalisation par la régression linéaire (librairie `forecast`)

- On peut également utiliser la fonction `tslm` de la librairie `forecast`
- Cette fonction ajuste un modèle linéaire incluant la saisonalité et la tendance (et éventuellement la tendance au carré)
- Modèle linéaire avec saisonnalité et tendance - données `retail`

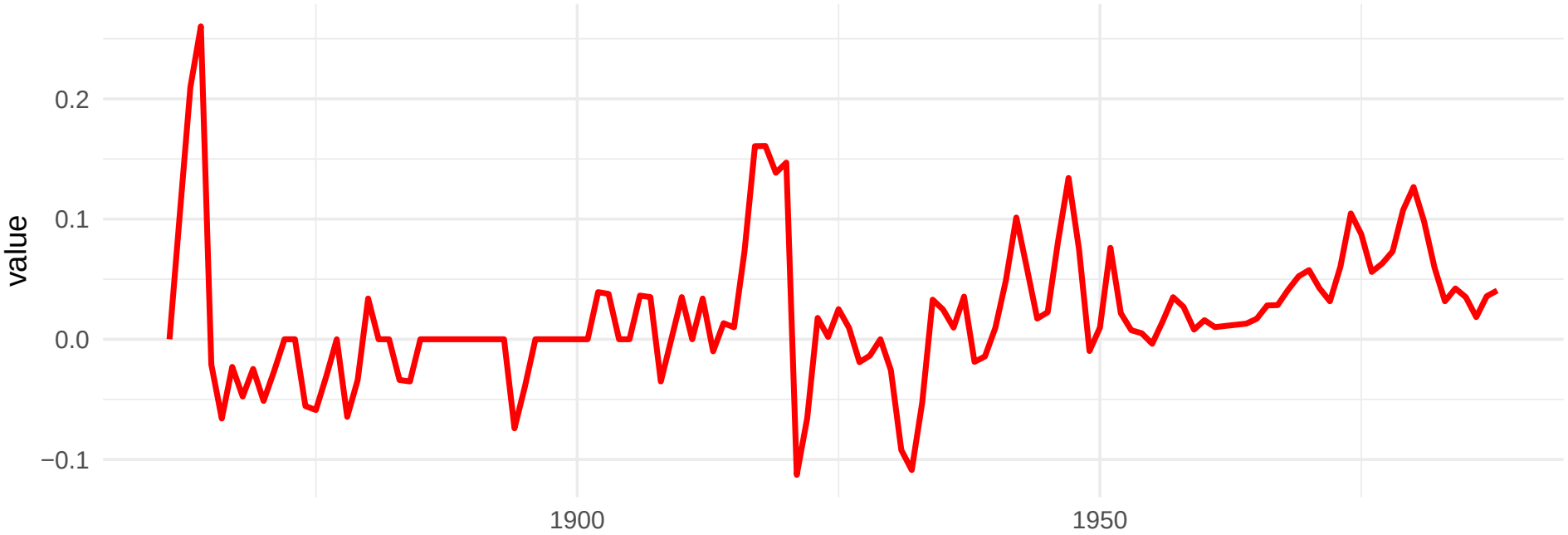
```
library(forecast)
bhat = tslm(retail_ts~trend+I(trend^2)+season)
summary(bhat)
```

```
##
## Call:
## tslm(formula = retail_ts ~ trend + I(trend^2) + season)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -111154  -19147   -5324   17916   72473
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.371e+05  5.928e+03  23.123  < 2e-16 ***
## trend        4.276e+02  4.859e+01   8.799  < 2e-16 ***
## I(trend^2)   1.535e+00  1.232e-01  12.459  < 2e-16 ***
## season2     -3.672e+03  6.501e+03  -0.565 0.572588
## season3      3.620e+04  6.501e+03   5.568 5.00e-08 ***
## season4      2.812e+04  6.501e+03   4.326 1.96e-05 ***
## season5      4.754e+04  6.501e+03   7.312 1.67e-12 ***
```

Stationarisation

- Une série est dite intégrée d'ordre d, notée $I(d)$, s'il faut la différencier d fois pour obtenir une série stationnaire
- La fonction `diff()` permet d'obtenir la série intégrée
- La série intégrée débute au temps $t + 1$

```
cpi_diff <- diff(NelPlo[,"cpi"],1)

cpi_diff %>% as_tsibble() %>%
  ggplot(aes(x = index, y = value)) +
    geom_line(colour="red", size = 1)
```



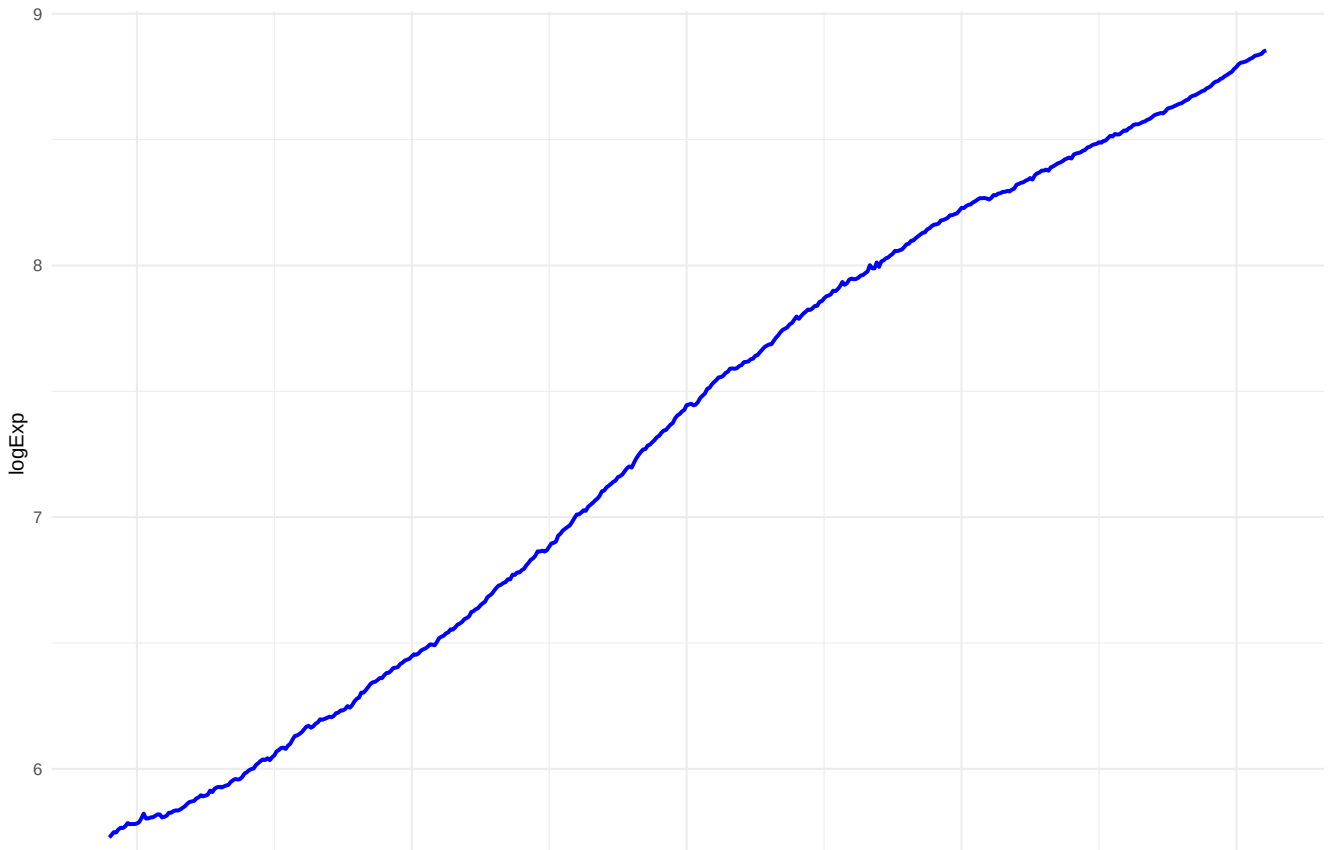
Test de non-stationarité - ADF

- Le test ADF (Augmented Dickey-Fuller) permet de tester la non-stationarité d'une série
- Avant de réaliser un test de non-stationnarité, il faut examiner le chronogramme de la série pour voir si la série présente une tendance, soit stochastique (marche aléatoire avec dérive), soit déterministe (série stationnaire à un trend déterministe près)

Série présentant une tendance

- La série du log des dépences de consommation aux USA présente une tendance

```
USExp <- USIncExp %>% as_tsibble() %>% filter(key=='expenditure') %>%
  mutate(logExp=log(value))
ggplot(USExp, aes(x = index, y = logExp)) +
  geom_line(color = "blue", size = 1)
```



Test ADF avec la fonction `urf.df` (librairie `urca`)

- Nous allons tester les hypothèses:
  - Hypothèse nulle H0: la série est non stationnaire avec dérive
  - Hypothèse alternative H1: la série est stationnaire avec trend déterministe
- Le test considère un modèle autorégressif d'ordre p (p repésenté par le lag est inconnu)
- On choisi pour commencer une valeur de p élevée et on retient la première valeur du lag fortement significative

```
library(urca)
exp.df <- ur.df(y=as.data.frame(USExp)[,"logExp"], lags=12, type='trend')
exp.df
```

```
##
## ###############################################################
## # Augmented Dickey-Fuller Test Unit Root / Cointegration Test #
## ###############################################################
##
## The value of the test statistic is: -0.4615 5.0639 0.7187
```

Test ADF: Résultats

- La première valeur p fortement significative est 8

```
summary(exp.df)
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression trend
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0208491 -0.0030259 -0.0000296  0.0034857  0.0233391
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.130e-02  1.613e-02   0.701 0.483735
## z.lag.1     -1.339e-03  2.901e-03  -0.461 0.644675
## tt           7.074e-06  1.965e-05   0.360 0.719044
## z.diff.lag1 -2.085e-01  4.567e-02  -4.565 6.37e-06 ***
## z.diff.lag2 -4.538e-02  4.627e-02  -0.981 0.327235
## z.diff.lag3 -1.030e-02  4.614e-02  -0.223 0.823428
## z.diff.lag4 -2.322e-03  4.589e-02  -0.051 0.959664
## z.diff.lag5  6.135e-02  4.562e-02   1.345 0.179309
## z.diff.lag6  1.505e-01  4.517e-02   3.332 0.000931 ***
## z.diff.lag7  1.621e-01  4.515e-02   3.590 0.000365 ***
```

Test ADF: Résultats

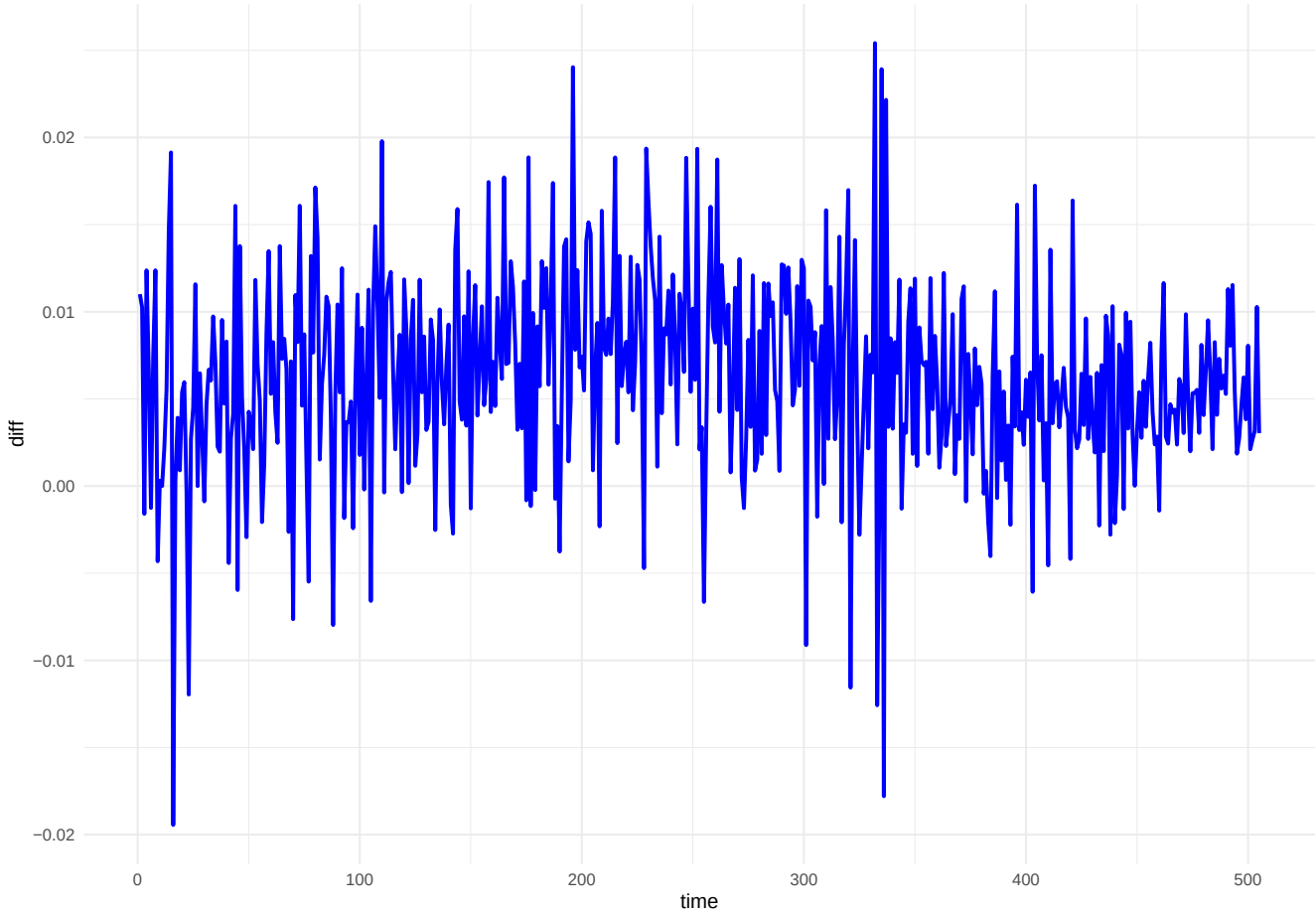
- Résultat du test avec $p = 8$
- La statistique qui nous intéresse est `tau3`, la valeur est supérieure au seuil de 10%, on retient l'hypothèse H0, la série ne présente pas de tendance déterministe et peut être rendue stationaire par différentiation

```
exp.df <- ur.df(y=as.data.frame(USExp)[,"logExp"], lags=8, type='trend')
summary(exp.df)
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression trend
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0205542 -0.0030498 -0.0001604  0.0036908  0.0222845
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.117e-02  1.609e-02   0.694 0.487809
## z.lag.1     -1.236e-03  2.883e-03  -0.429 0.668264
## tt           7.040e-06  1.949e-05   0.361 0.718134
```

Série différenciée des dépenses de consommation aux USA

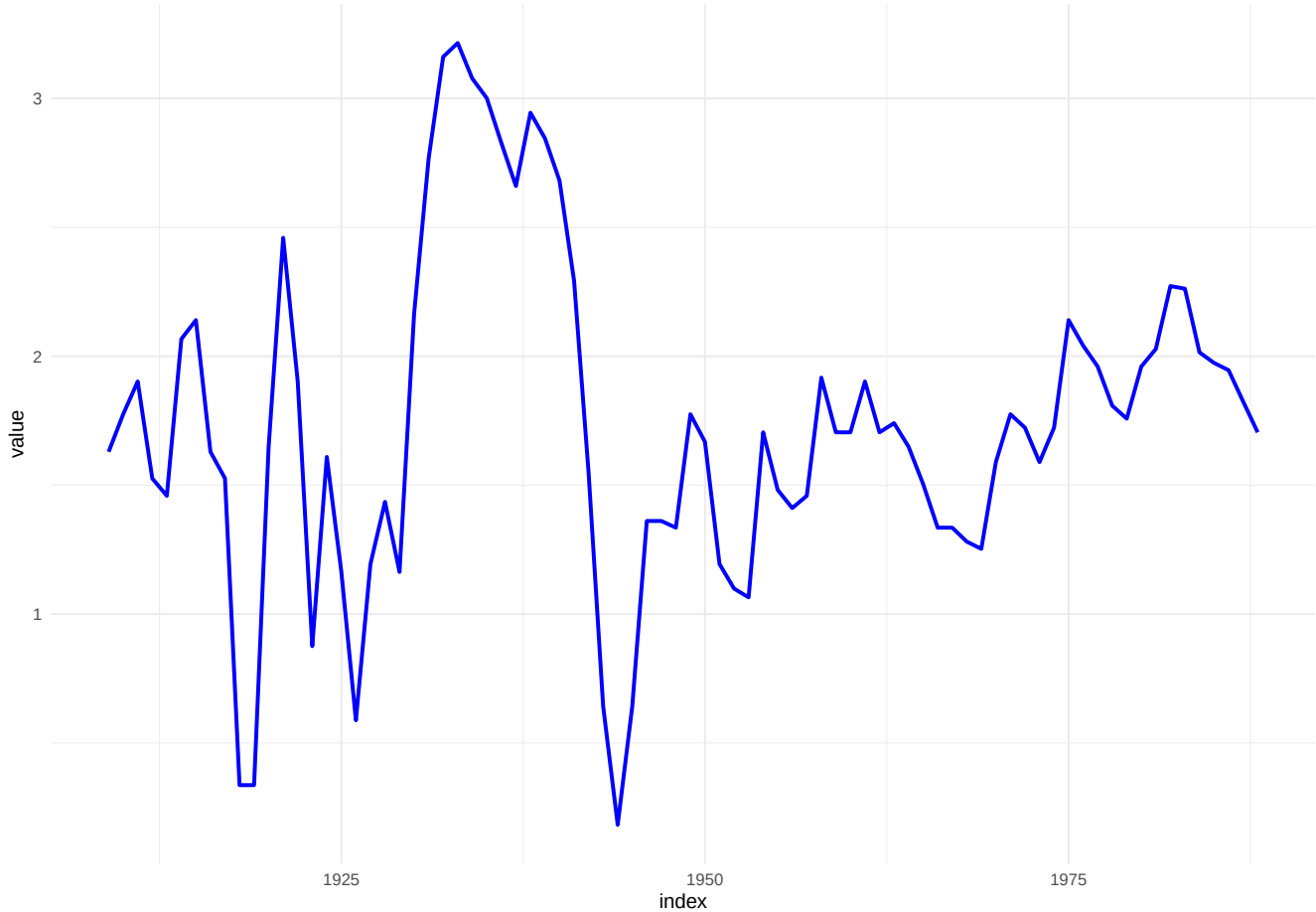
```
USExp_diff <- tibble(time=1:(nrow(USExp)-1), diff= diff(as.data.frame(USExp)[, "logExp"]))
ggplot(USExp_diff, aes(x = time, y = diff)) +
  geom_line(color = "blue", size = 1)
```



Série ne présentant pas de tendance

- La série du taux de chômage aux USA ne présente pas de tendance

```
Unemp <- nelplo_tsbl %>% filter(key=='unemp' & index >= 1909)
ggplot(Unemp, aes(x = index, y = value)) +
  geom_line(color = "blue", size = 1)
```



Série ne présentant pas de tendance - Test ADF

- Après un premier test avec `lags=6`, le premier lag singificatif est 1
- On regarde la statistique `tau2`, ici elle est inférieure à la valeur critique à 1% donc on rejete H0 et on conclut à la stationarité de la série

```
unemp.df <- ur.df(y=as.data.frame(Unemp)[,"value"], lags=1, type='drift')
summary(unemp.df)
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression drift
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + z.diff.lag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.21622 -0.15984 -0.01542  0.20900  0.92354
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.48815    0.13250   3.684 0.000431 ***
## z.lag.1     -0.28152    0.07201  -3.909 0.000201 ***
## z.diff.lag   0.30916    0.10978   2.816 0.006208 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Test de stationarité - KPSS

- Le test KPSS proposé par Kwiatkowski est disponible dans les librairies `urca` et `tseries`
- Hypothèse nulle H0: la série est stationnaire (soit à une tendance près, soit à une moyenne non nulle près)
- Le test suppose que la série est la somme d'une marche aléatoire R_t, d'un trend déterministe et d'une erreur stationnaire U_t:

$$y_t = R_t + \beta_1 + \beta_2 t + U_t$$

où $R_t = R_{t-1} + z_t$

- Pour tester que la série y_t est stationnaire à une tendance près, l'hypothèse nulle est $\sigma_z^2 = 0$

Test de stationarité - KPSS - Série USExp

- On teste ici l'hypothèse H0: la série des dépenses de consommation aux USA est stationaire à une tendance près (option 'type="tau"')

```
ktest <- ur.kpss(USExp$logExp, type="tau", lags="long")
summary(ktest)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: tau with 17 lags.
##
## Value of test-statistic is: 0.488
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.119 0.146  0.176 0.216
```

- La valeur de la statistique est nettement supérieure à la valeur critique au seuil de 1%, on rejette H0, la série ne présente pas de tendance linéaire

Test de stationarité - KPSS - Série USExp

- On teste maintenant l'hypothèse H0: la série des dépenses de consommation aux USA est stationaire de moyenne constante (option 'type="mu"')

```
ktest <- ur.kpss(USExp$logExp, type="mu", lags="long")
summary(ktest)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 17 lags.
##
## Value of test-statistic is: 2.9218
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

- La valeur de la statistique est nettement supérieure à la valeur critique au seuil de 1%, on rejette H0, la série n'est pas stationnaire

Test de stationarité - KPSS - Série USExp intégrée

- On teste maintenant la stationarité de la série intégrée

```
ktest <- ur.kpss(USExp_diff$diff, type="mu", lags="long")
summary(ktest)
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 17 lags.
##
## Value of test-statistic is: 0.5706
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

- La valeur de la statistique se situe entre les valeurs critiques à 2.5% et 5%

Exercices

- La série du PIB français présente t-elle une tendance ?
- Cette tendance est-elle déterministe ou stochastique ?
- Appliquez le test de stationarité approprié

Section 9

Modélisation de séries temporelles stationnaires

Introduction

- Les modèles AR (AutoRegressive), MA (Moving Average) et ARMA (AutoRegressive Moving Average) sont des modèles fondamentaux pour étudier et décrire le comportement des séries temporelles
- Ces modèles ont été popularisés par George Box et Gwilym Jenkins
- Leur principale limitation est qu'ils ne peuvent modéliser que des séries stationnaires, cependant, on peut transformer des séries non-stationnaires par différenciation pour les étudier avec ce cadre (modèles ARIMA)

Modèle autorégressif (AR)

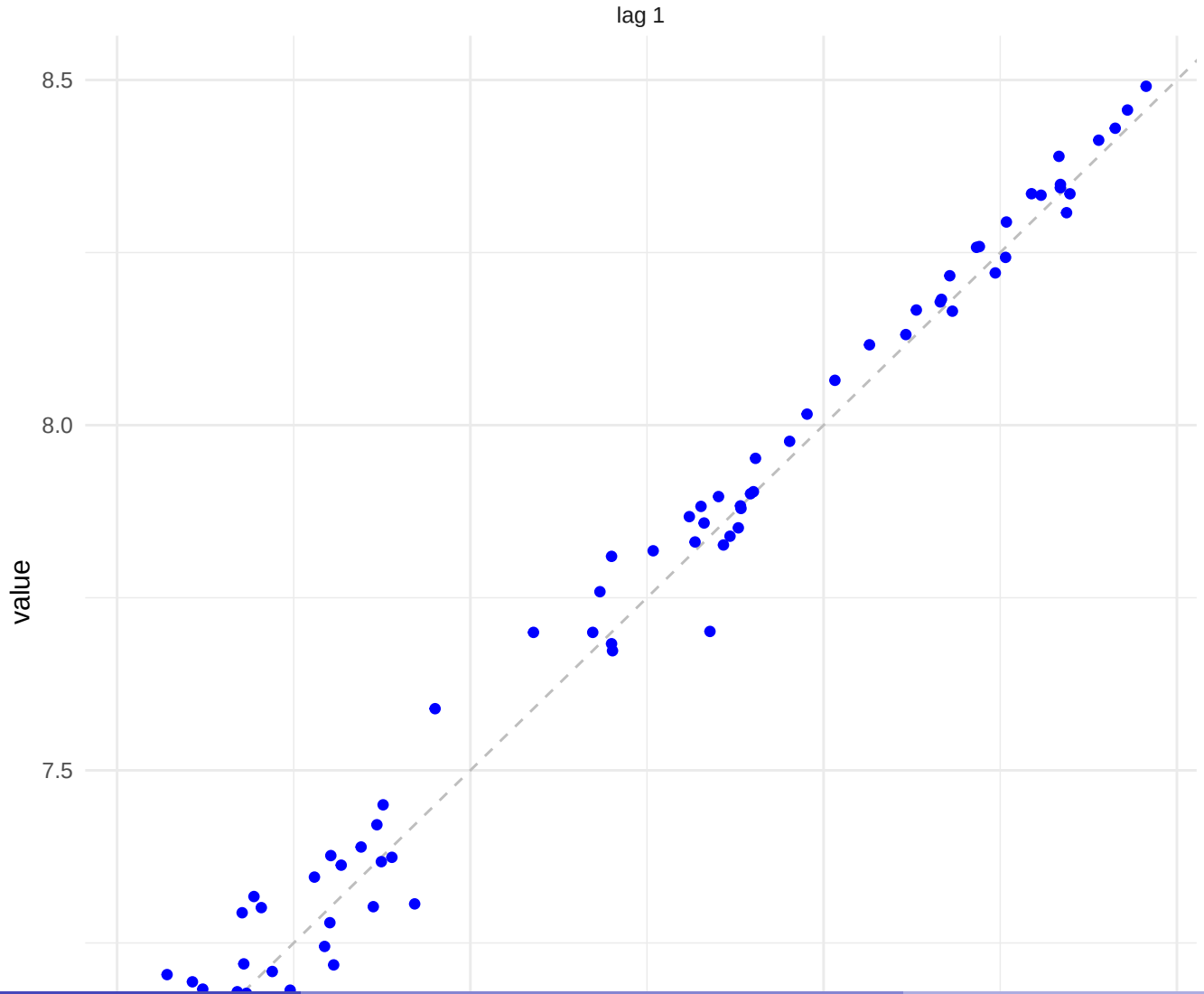
- Dans un modèle auto régressif, les variables explicatives sont des valeurs passées (lags) de la variable expliquée
- Dans un modèle AR(1) (autorégressif d'ordre 1), y est retardé d'une période

$$y_t = \alpha + \beta_1 \cdot y_{t-1} + \epsilon_t$$

avec $\epsilon_t \sim N(0, \sigma^2)$ - Pour un modèle AR(1) stationnaire, $|\beta_1| < 1$ - Propriétés d'un processus AR(1): - Moyenne de y_t: $\mu = \frac{\alpha}{1-\beta_1}$ - Variance: $Var(x_t) = \frac{\sigma_w^2}{1-\beta_1^2}$ - Corrélation: $\rho_h = \beta_1^h$

Modèle autorégressif - Données Nelson-Plosser

- On peut visualiser y x y_{t-1} à l'aide de la fonction `gg_lag()`



Modèle autorégressif - Données Nelson-Plosser - ACF

- Fonction d'autocorrélation avec la fonction `ACF`:

```
## # A tibble: 18 x 3
##    key             lag   acf
##    <chr>       <cf_lag> <dbl>
##  1 gnp.capita       1Y 0.962
##  2 gnp.capita       2Y 0.920
##  3 gnp.capita       3Y 0.878
##  4 gnp.capita       4Y 0.839
##  5 gnp.capita       5Y 0.800
##  6 gnp.capita       6Y 0.765
##  7 gnp.capita       7Y 0.732
##  8 gnp.capita       8Y 0.694
##  9 gnp.capita       9Y 0.660
## 10 gnp.capita      10Y 0.626
## 11 gnp.capita      11Y 0.592
## 12 gnp.capita      12Y 0.555
## 13 gnp.capita      13Y 0.524
## 14 gnp.capita      14Y 0.496
## 15 gnp.capita      15Y 0.464
## 16 gnp.capita      16Y 0.433
## 17 gnp.capita      17Y 0.402
## 18 gnp.capita      18Y 0.370
```

- La corrélation entre y et y_{t-1} est forte

Modèle autorégressif - Estimation - Données Nelson-Plosser

- On peut estimer le modèle AR(1) avec la fonction `lm()` (modèle de régression linéaire):

```
rdata <- nelplo1909p %>% filter(key=="gnp.capita")
summary(lm(value ~ lag(value), data=rdata))
```

```
##
## Call:
## lm(formula = value ~ lag(value), data = rdata)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.184309 -0.033036  0.007268  0.034030  0.122221
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.01981    0.11805   0.168    0.867
## lag(value)   0.99963    0.01526  65.523   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05924 on 76 degrees of freedom
##   (1 observation effacée parce que manquante)
## Multiple R-squared:  0.9826, Adjusted R-squared:  0.9824
## F-statistic:  4293 on 1 and 76 DF,  p-value: < 2.2e-16
```

Le coefficient associé [illegible] est significatif. La valeur de R^2 est élevée

Modèle moyenne mobile

- y_t est un processus moyenne mobile d'ordre q noté MA(q) si

$$y_t = \mu + \epsilon_t + \theta_1 \cdot \epsilon_{t-1} + \ldots + \theta_q \cdot \epsilon_{t-q}$$

- On considère que le processus est la résultante d'une combinaison linéaire de perturbations decorrélées (un bruit blanc et son passé)
- Un MA(q) est toujours stationnaire quelles que soient les valeurs des θ
- Les propriétés d'un modèle MA(1)

$$y_t = \mu + \epsilon_t + \theta_1 \cdot \epsilon_{t-1}$$

- $E[y_t] = \mu$
- $Var(y_t) = \sigma_\epsilon^2(1 + \theta_1^2)$
- ACF is $\rho_1 = \theta_1/(1 + \theta_1^2)$ and $\rho_h = 0$ for $h \geq 2$

Modèle ARMA

- Un processus ARMA (Auto Regressive Moving Average) est une synthèse des processus AR et MA
- y_t obéit à un modèle ARMA(p, q) s'il est stationnaire et vérifie :

$$y_t = c + \phi_1 \cdot y_{t-1} + \ldots + \phi_p \cdot y_{t-p} + \epsilon_t + \theta_1 \cdot \epsilon_{t-1} + \ldots + \theta_q \cdot \epsilon_{t-q}$$

Identification d'un modèle ARMA

- Soit la trajectoire observée $y_1, \ldots, y_t$ d'une série y_t, éventuellement transformée par passage en log
- Si cette trajectoire peut-être considérée comme stationnaire, on peut lui ajuster un modèle ARMA(p, q) (on ne traite pas ici des séries présentant une saisonnalité)
- La première étape consiste à choisir les ordres p et q
- Le choix de p et q n'est pas unique, il faut comparer plusieurs modèles
- Le premier critère pour juger de la qualité d'un modèle est la blancheur du résidu obtenu (voir la section définitons)

Modèle ARMA - Série des taux de chômage aux USA

- Nous avons vu que la série des taux de chômage aux USA peut-être considérée comme stationnaire
- On commence par examiner sa fonction d'autocorrélation

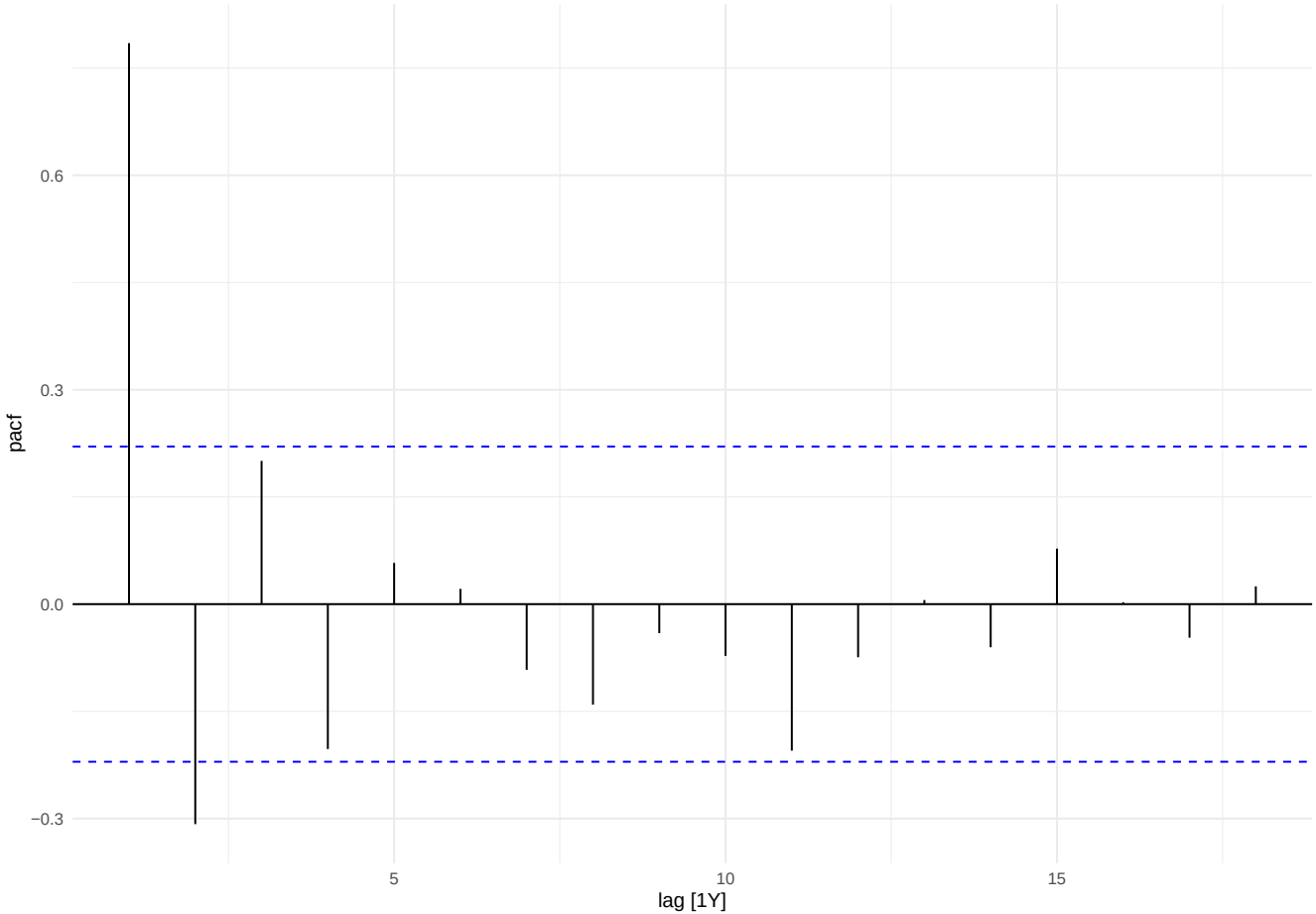
```
nelplo1909p %>% filter(key=="unemp") %>% ACF(value) %>% autoplot()
```

Fonction d'autocorrélation partielle

- Le coefficient d'autocorrélation partielle $\phi k, k$ représente l'apport d'explication de y_{t-k} à y_t, **toutes choses égales par ailleurs** ()
- La fonction d'autocorrélation partielle permet d'identifier l'ordre p d'un processus AR(p)

Fonction d'autocorrélation partielle avec PACF()

```
nelplo1909p %>% filter(key=="unemp") %>%
  PACF(value) %>% autoplot()
```



Estimation du modèle (1)

- On commence par un modèle ARMA(2,2)
- L'erreur standard est élevée par rapport à la valeur du coefficient

```
regdata <- nelplo1909p %>% filter(key=="unemp")
arma.mod1 <- arima(regdata[,"value"], c(2,0,2))
summary(arma.mod1)
```

```
##
## Call:
## arima(x = regdata[, "value"], order = c(2, 0, 2))
##
## Coefficients:
##          ar1      ar2     ma1      ma2  intercept
##       0.7228  -0.0376  0.4009  -0.1253     1.7357
## s.e.  0.9293   0.5663  0.9222   0.5102     0.1598
##
## sigma^2 estimated as 0.1296:  log likelihood = -32.05,  aic = 76.11
##
## Training set error measures:
##                       ME      RMSE       MAE       MPE     MAPE     MASE
## Training set 0.0004553661 0.3599465 0.2635755 -10.65611 24.60626 0.896286
##                     ACF1
## Training set 0.01310071
```

Estimation du modèle (2)

- On ajuste un modèle ARMA(1,1)
- L'erreur standard des coefficients a diminué fortement

```
regdata <- nelplo1909p %>% filter(key=="unemp")
arma.mod2 <- arima(regdata[,"value"], c(1,0,1))
summary(arma.mod2)
```

```
##
## Call:
## arima(x = regdata[, "value"], order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.5895  0.5555     1.7356
## s.e.  0.1032  0.1103     0.1503
##
## sigma^2 estimated as 0.13:  log likelihood = -32.19,  aic = 72.38
##
## Training set error measures:
##                       ME      RMSE       MAE       MPE     MAPE      MASE
## Training set 0.0005107137 0.3605539 0.2665423 -10.46991 24.69032 0.9063747
##                      ACF1
## Training set -0.006430895
```

Comparaison des modèles

- On retient le second modèle avec moins de paramètres (p=1 et q=1)

```
AIC(arma.mod1, arma.mod2)
```

```
## # A tibble: 2 x 2
##      df   AIC
##   <dbl> <dbl>
## 1     6  76.1
## 2     4  72.4
```

Utilisation de la fonction `auto.arima`

- La fonction `auto.arima()` recherche le meilleur modèle en utilisant un critère d'information
- Le résultat confirme le choix du modèle ARMA(1,1)

```
nelplo.arma <- auto.arima(regdata)
nelplo.arma
```

```
## Series: regdata
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1     ma1    mean
##       0.5895  0.5555  1.7356
## s.e.  0.1032  0.1103  0.1503
##
## sigma^2 = 0.1351:  log likelihood = -32.19
## AIC=72.38   AICc=72.92   BIC=81.85
```

Diagnostic du modèle - Préparation

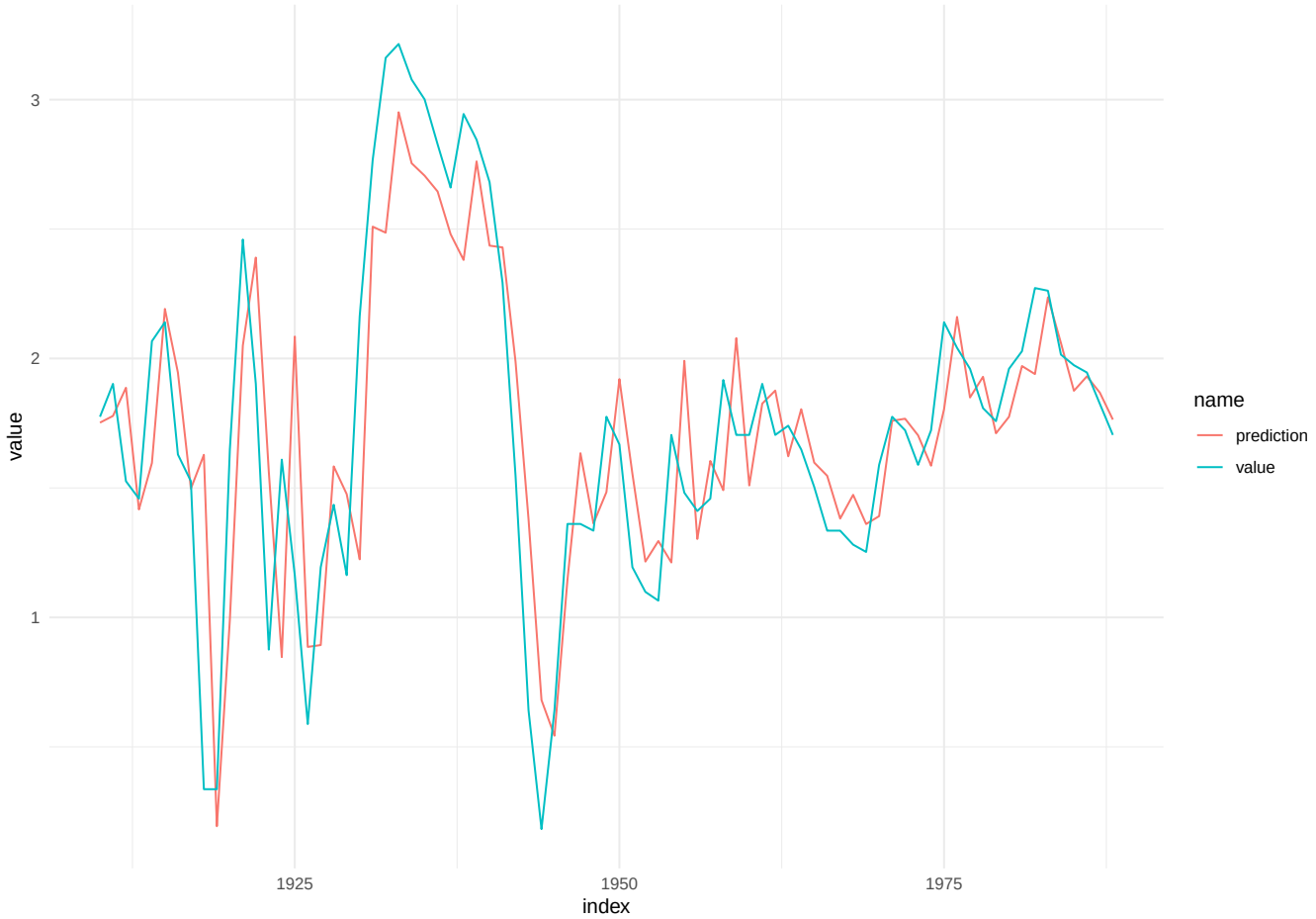
- On prépare un data frame avec les valeurs observées, les valeurs prédites par le modèle, et les résidus

```
nelplo.arma.diag <- data.frame(regdata, prediction=nelplo.arma$fitted, residus=nelplo.arma$residuals)
nelplo.arma.diag
```

```
## # A tibble: 79 x 5
##    index key   value prediction residus
##    <dbl> <chr> <dbl>      <dbl>   <dbl>
##  1  1910 unemp 1.77       1.75   0.0227
##  2  1911 unemp 1.90       1.78   0.124
##  3  1912 unemp 1.53       1.89  -0.361
##  4  1913 unemp 1.46       1.42   0.0417
##  5  1914 unemp 2.07       1.60   0.471
##  6  1915 unemp 2.14       2.19  -0.0515
##  7  1916 unemp 1.63       1.95  -0.316
##  8  1917 unemp 1.53       1.50   0.0287
##  9  1918 unemp 0.336      1.63  -1.29
## 10  1919 unemp 0.336      0.193  0.143
## # i 69 more rows
```

Diagnostic du modèle - Valeurs observées x prédites

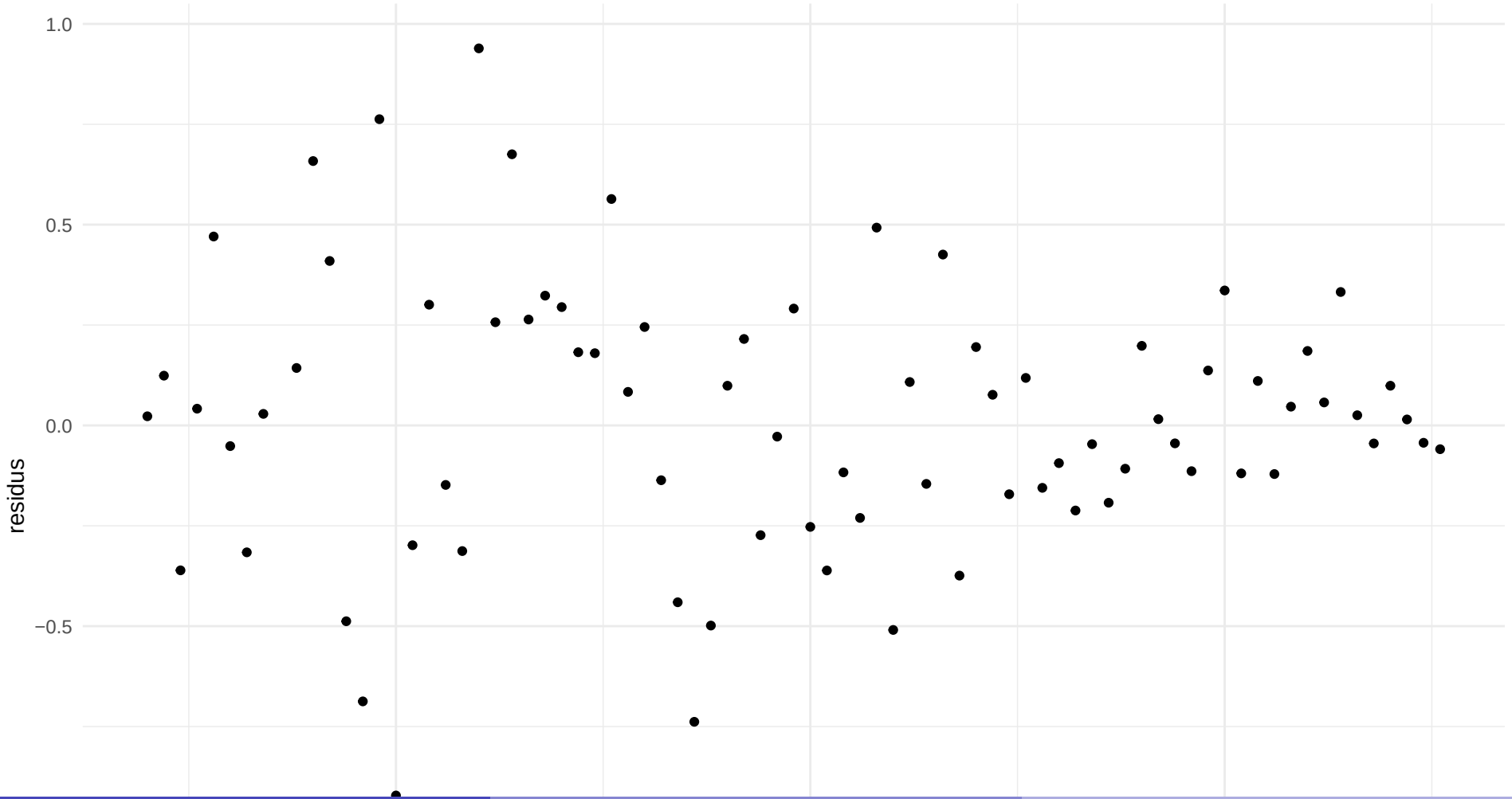
```
nelplo.arma.diag %>%
  pivot_longer(cols=3:5) %>%
  filter(name %in% c("prediction", "value")) %>%
  ggplot() + geom_line(aes(x=index, y=value, color=name))
```



Diagnostic du modèle - Résidus

```
nelplo.arma.diag %>% ggplot() +
  geom_point(aes(x=index, y=residus))
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



Projection

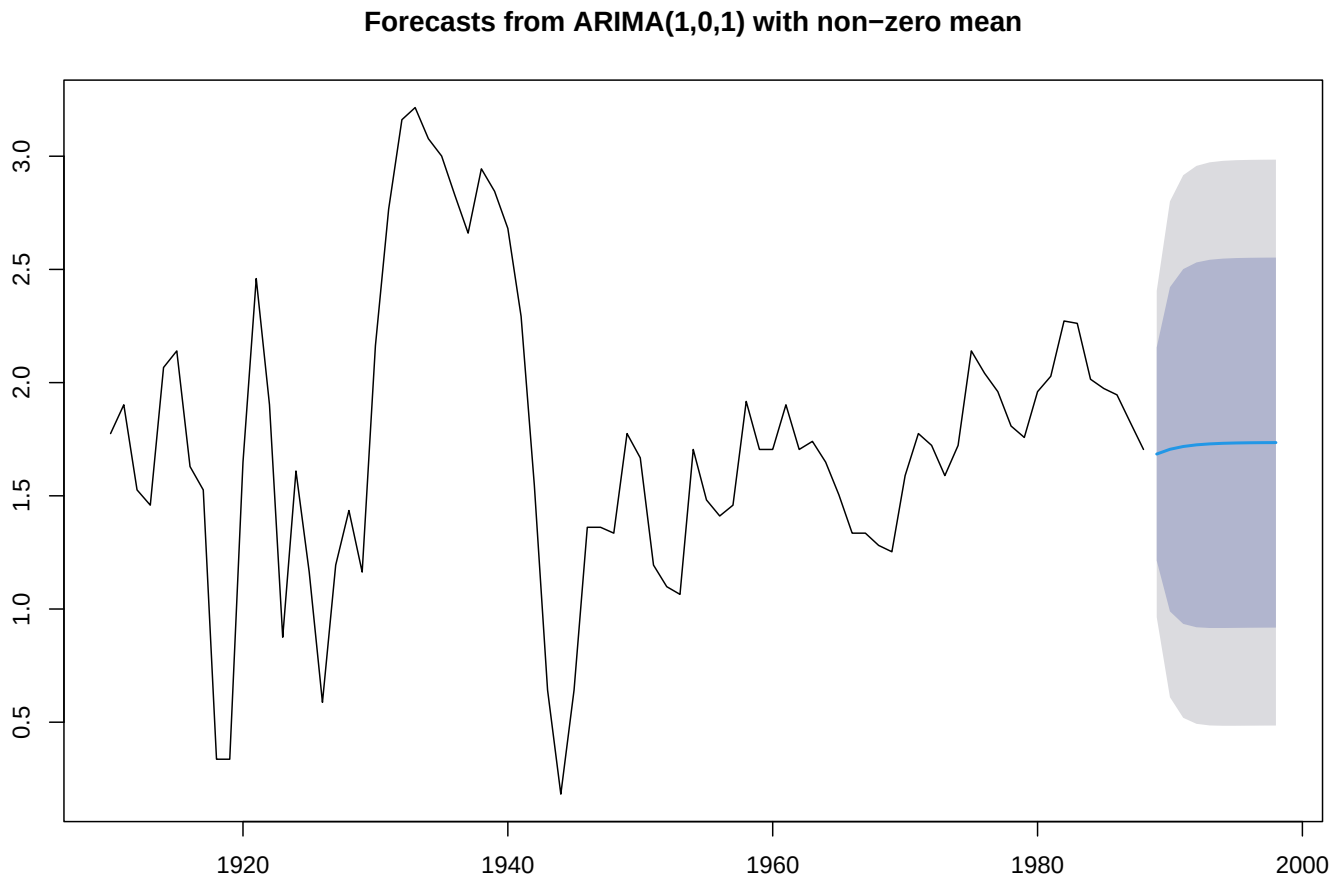
- On peut utiliser la fonction générique `forecast` pour la projection du taux de chômage

```
forecast(nelplo.arma, h=10)
```

```
##      Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 1989       1.684490 1.2133901 2.155590 0.9640049 2.404975
## 1990       1.705450 0.9892962 2.421604 0.6101872 2.800713
## 1991       1.717806 0.9342365 2.501376 0.5194397 2.916173
## 1992       1.725090 0.9194124 2.530768 0.4929124 2.957268
## 1993       1.729384 0.9161642 2.542603 0.4856717 2.973096
## 1994       1.731915 0.9160908 2.547739 0.4842196 2.979611
## 1995       1.733407 0.9166798 2.550135 0.4843304 2.982484
## 1996       1.734287 0.9172458 2.551328 0.4847304 2.983843
## 1997       1.734805 0.9176554 2.551955 0.4850823 2.984528
## 1998       1.735111 0.9179232 2.552299 0.4853301 2.984892
```

Projection - Graphique

```
plot(forecast(nelplo.arma, h=10))
```



Section 10

Modèlisation de séries non-stationnaires

Modèle ARIMA

- Un modèle ARIMA(p,d,q) est un modèle ARMA sur la série différenciée
- d est l'ordre de différentiation
- L'estimation d'un modèle ARIMA revient à un modèle ARMA sur la série différenciée

Identification d'un modèle ARIMA

- Soit la trajectoire observée $y_1, \ldots, y_t$ d'une série y_t , éventuellement obtenue après transformation d'une série initiale par passage en log
- La série n'est pas stationnaire et on veut lui ajuster un modèle ARIMA(p, d, q) (on ne traite pas ici des séries présentant une saisonnalité)
- Une fois d choisis, on est ramené à l'identification d'un ARMA sur la série différenciée
- Pour choisir d, on peut tester l'hypothèse $\mathrm{d} = 1$ contre $\mathrm{d} = 0$ avec un test ADF
- On peut également comparer les modèles avec et sans différenciation à l'aide d'un critère d'information ou de la valeur prédictive

Modèle ARIMA - Série des dépenses aggégées de consommation aux USA

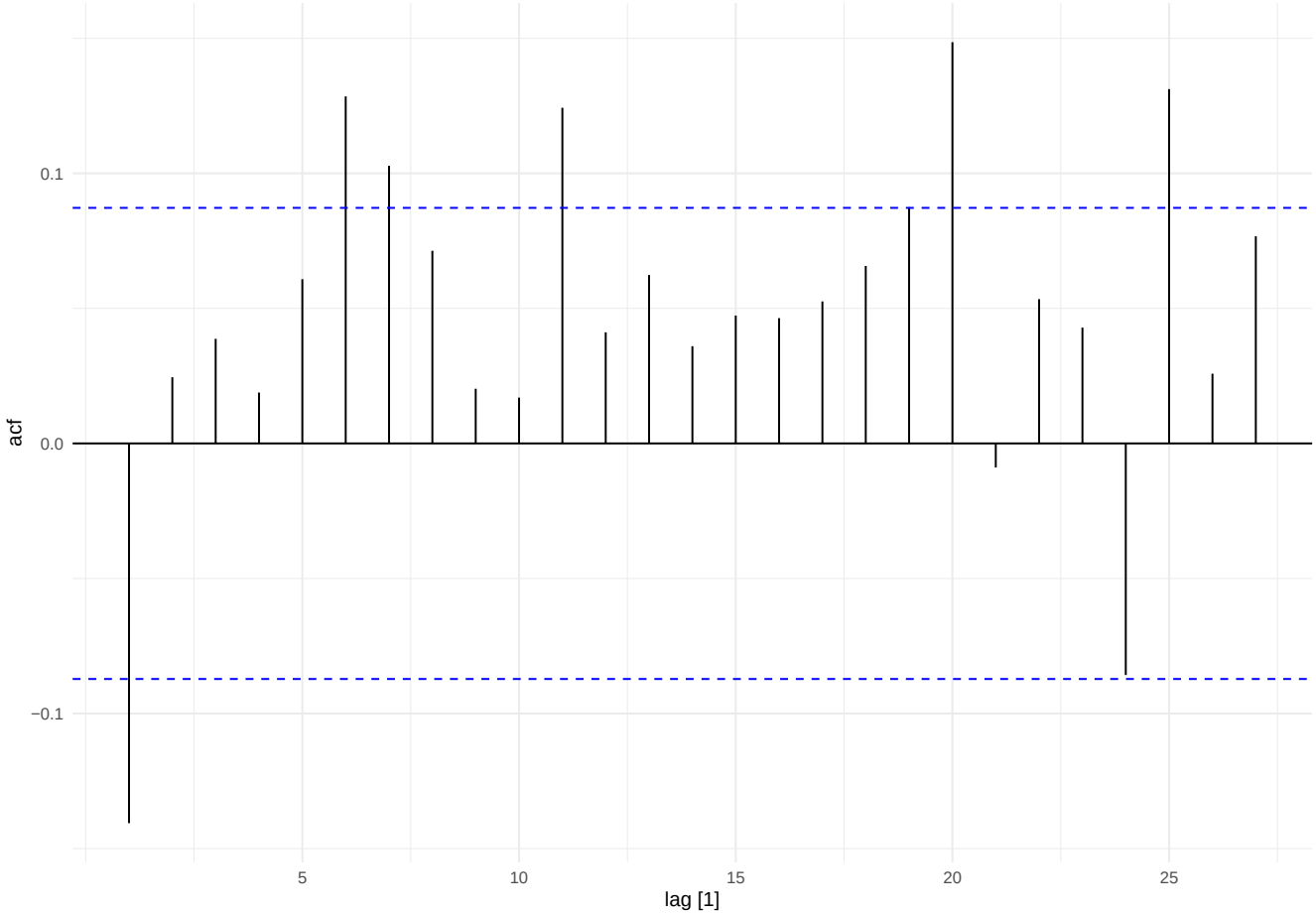
- Nous avons vu que cette série (après transformation logarithmique) est non stationnaire et qu'elle possède un trand stochastique
- On peut donc la différencier puis ajuster un modèle ARIMA
- La série différenciée a été calculée précédemment

```
USExp_diff
```

```
## # A tibble: 505 x 2
##     time      diff
##    <int>     <dbl>
##  1     1  0.0110
##  2     2  0.0103
##  3     3 -0.00160
##  4     4  0.0124
##  5     5  0.00660
##  6     6 -0.00125
##  7     7  0.00750
##  8     8  0.0124
##  9     9 -0.00431
## 10    10  0.000309
## # i 495 more rows
```

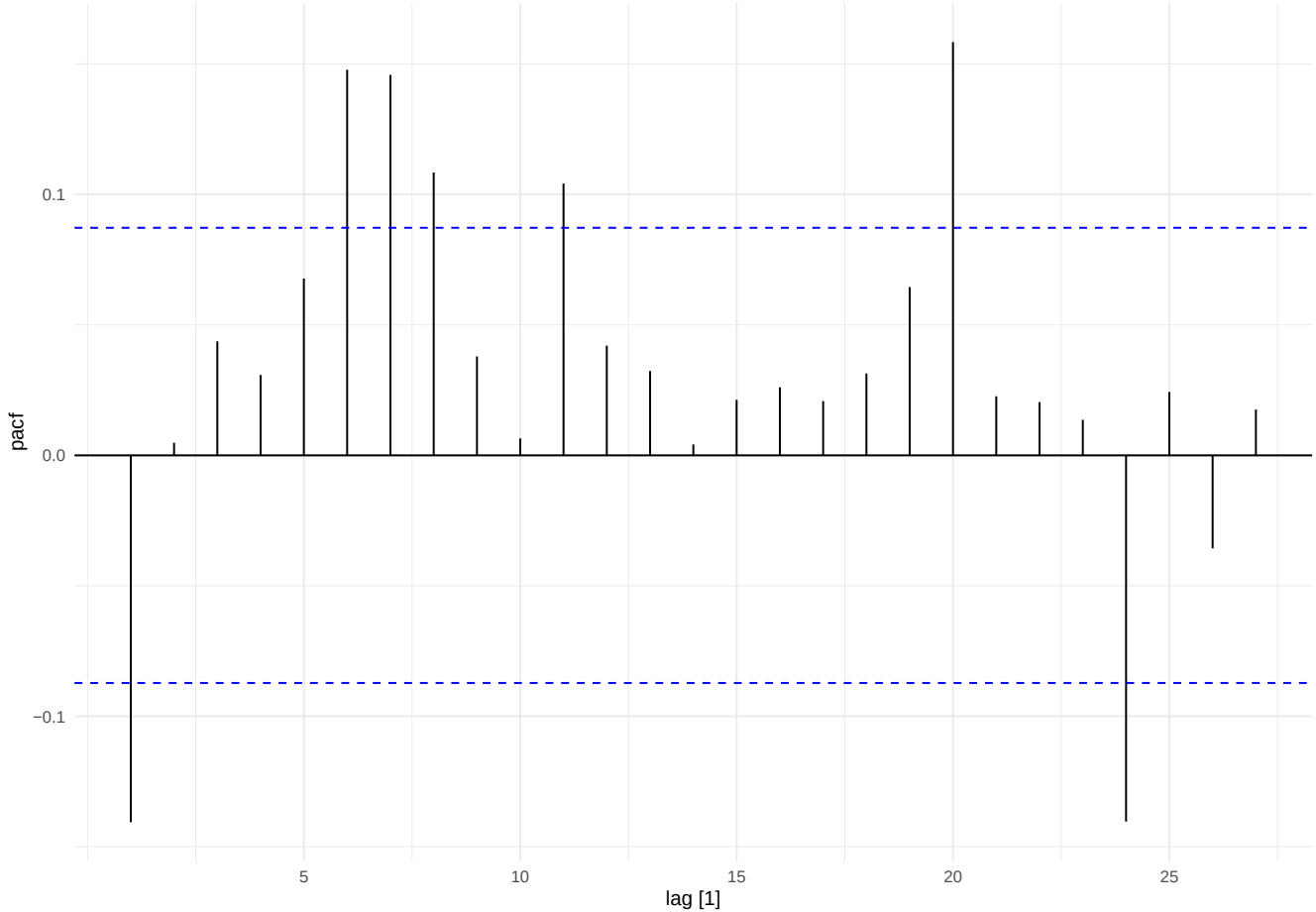
Examen de la fonction d'autocorrélation

```
USExp_diff %>% as_tsibble(index=time) %>% ACF(diff) %>% autoplot()
```



Examen de la fonction d'autocorrélation partielle

```
USExp_diff %>% as_tsibble(index=time) %>% PACF(diff) %>% autoplot()
```



Utilisation de la fonction `auto.arima`

```
regdata <- USExp[,"logExp"]
arimod <- auto.arima(regdata)
arimod
```

```
## Series: regdata
## ARIMA(2,2,2)
##
## Coefficients:
##          ar1     ar2      ma1     ma2
##       0.3384  0.0158  -1.5509  0.5725
## s.e.  0.1704  0.0679   0.1644  0.1578
##
## sigma^2 = 3.053e-05:  log likelihood = 1906.35
## AIC=-3802.69   AICc=-3802.57   BIC=-3781.58
```

Modèle ARIMA - Prédiction

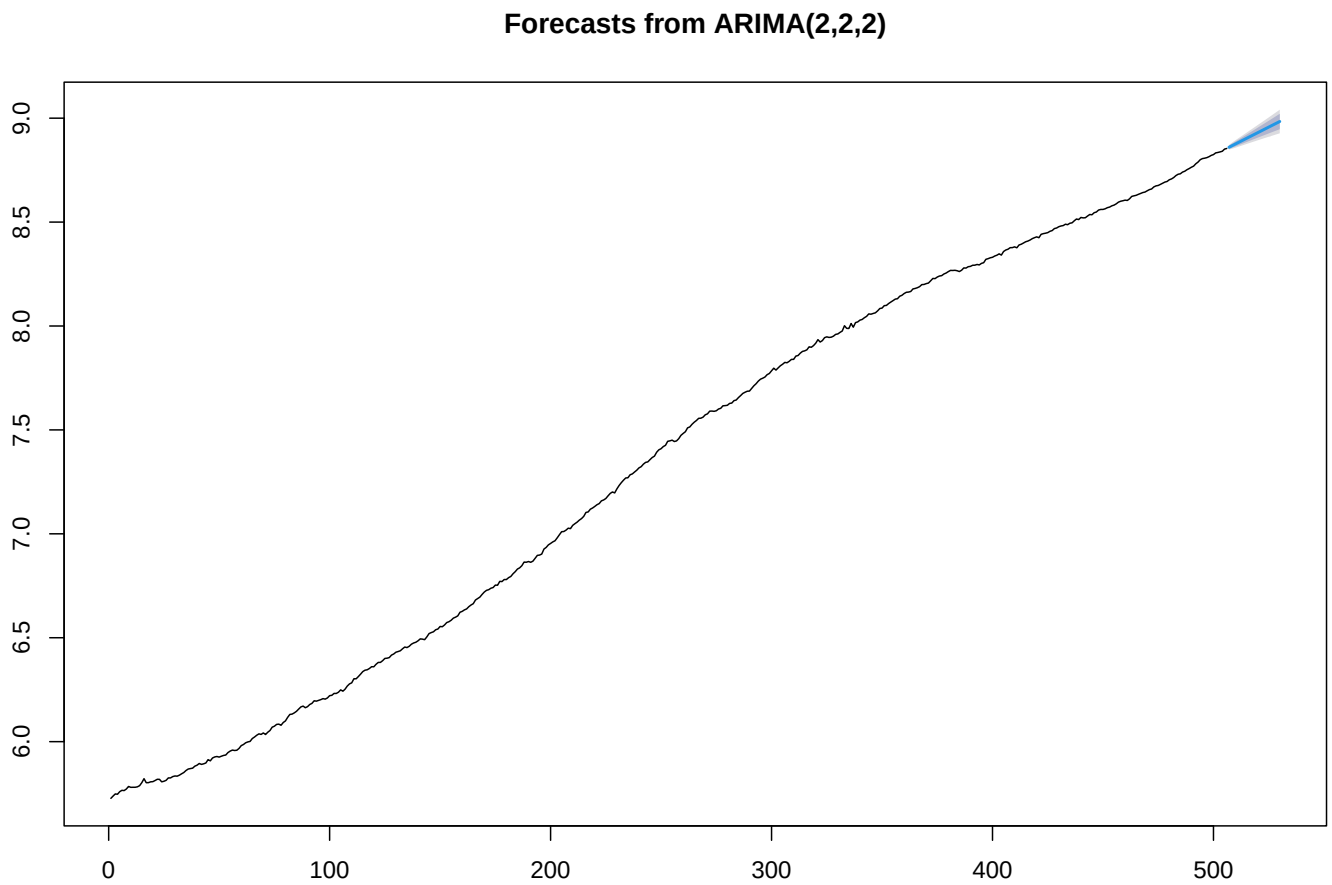
- La fonction `forecast()` produit les prédiction à partir du modèle sélectionné, avec un intervalle de confiance

```
forecast(arimod, h=24)
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 507       8.859892 8.852811 8.866972 8.849063 8.870721
## 508       8.865359 8.856346 8.874371 8.851575 8.879142
## 509       8.870780 8.860308 8.881252 8.854765 8.886796
## 510       8.876182 8.864410 8.887955 8.858178 8.894187
## 511       8.881577 8.868565 8.894590 8.861676 8.901478
## 512       8.886969 8.872742 8.901197 8.865210 8.908729
## 513       8.892361 8.876928 8.907794 8.868758 8.915963
## 514       8.897751 8.881115 8.914388 8.872308 8.923195
## 515       8.903142 8.885300 8.920984 8.875854 8.930429
## 516       8.908532 8.889479 8.927586 8.879393 8.937672
## 517       8.913923 8.893652 8.934194 8.882921 8.944925
## 518       8.919313 8.897816 8.940811 8.886435 8.952191
## 519       8.924704 8.901970 8.947438 8.889935 8.959473
## 520       8.930094 8.906114 8.954075 8.893419 8.966770
## 521       8.935485 8.910246 8.960723 8.896886 8.974084
## 522       8.940875 8.914367 8.967383 8.900335 8.981416
## 523       8.946266 8.918476 8.974055 8.903765 8.988766
## 524       8.951656 8.922573 8.980740 8.907177 8.996136
## 525       8.957047 8.926656 8.987437 8.910569 9.003525
## 526       8.962437 8.930727 8.994147 8.913941 9.010934
## 527       8.967828 8.934785 9.000870 8.917293 9.018362
## 528       8.973218 8.938829 9.007607 8.920625 9.025811
## 529       8.978609 8.942861 9.014356 8.923937 9.033280
## 530       8.983999 8.946879 9.021119 8.927229 9.040769
```

Modèle ARIMA - Prédiction - Graphique

```
plot(forecast(arimod, h=24))
```



Exercice

- Ajustez un modèle ARMA ou ARIMA sur la série du PIB français
- Prédisez le PIB pour les trimestres à venir

Section 11

Modèles multivariés

Série multivariée et processus VAR

- Les modèles VAR (Vector Autoregression) sont l'extension des modèles AR à des séries multivariées
- L'évolution d'une série est modélisée par ses valeurs passées et celles des autres séries
- La série multivariée $y_t = (y_{1t}, \ldots, y_{kt}, \ldots, y_{Kt}), k = 1, \ldots, K$ contient K variables
- Un processus VAR(p) est défini par:

$$y_t = A_1 y_{t-1} + \ldots + A_p y_{t-p} + \mu_t$$

avec $E(\mu_t) = 0$

- A_i est une matrice de $(K \times K)$ coefficients, $i = 1, \ldots, p$

Série multivariée - Données `Canada`

- On utilise la librairie `vars` (voir l'article ici)
- Les séries utilisées représentent des indicateurs macro-économiques du marché du travail au Canada (données OCDE):
  - `prod`: productivité du travail (log différence entre le GDP et nombre d'actifs)
  - `e`: nombre d'actifs (employment)
  - `U`: taux de chômage (unemployment rate)
  - `rw`: log de l'index des salaires réels (real wage index)
- Les séries s'étendent du premier trimestre 1980 aux quatrième trimestre 2004

Série multivariée - Données Canada

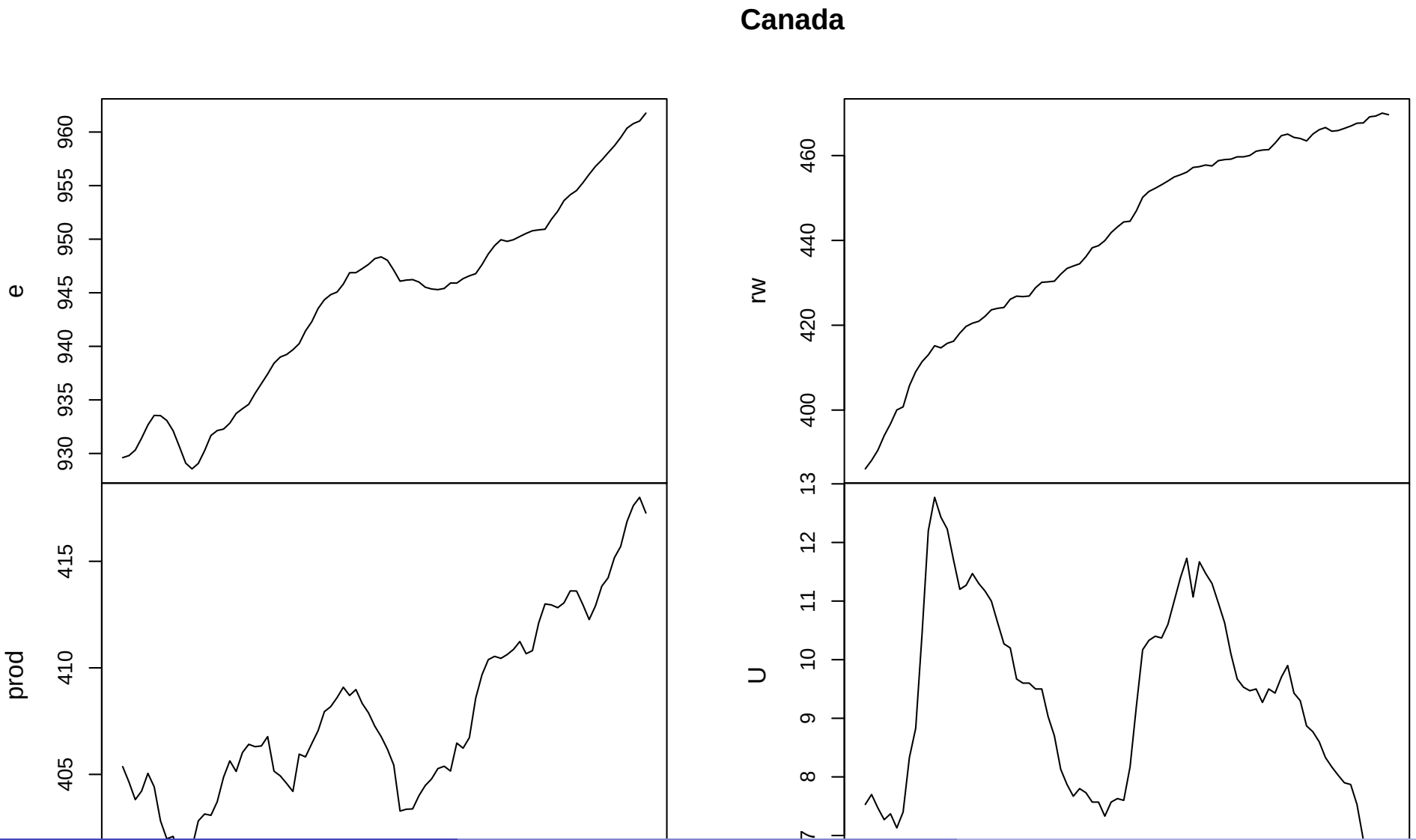
- On charge la librairie et les données (objet mts)

```
library("vars")
data("Canada")
window(Canada, start=c(1980,1), end=c(1981,4))
```

```
##                 e     prod       rw    U
## 1980 Q1 929.6105 405.3665 386.1361 7.53
## 1980 Q2 929.8040 404.6398 388.1358 7.70
## 1980 Q3 930.3184 403.8149 390.5401 7.47
## 1980 Q4 931.4277 404.2158 393.9638 7.27
## 1981 Q1 932.6620 405.0467 396.7647 7.37
## 1981 Q2 933.5509 404.4167 400.0217 7.13
## 1981 Q3 933.5315 402.8191 400.7515 7.40
## 1981 Q4 933.0769 401.9773 405.7335 8.33
```

Série multivariée - Graphique

```
plot(Canada, nc = 2, xlab = "")
```



Série multivariée - Analyse préliminaire

- Le modèle VAR ne permet de modéliser que des séries *stationnaires*
- Un test de stationarité ADF (Augmented Dickey-Fuller) est réalisé sur les 4 séries
- Toutes les séries sont rendues stationnaires avec une intégration d'ordre 1

Test de stationarité sur la série prod

- La série prod présente une tendance, on utilise type="trend"
- La valeur tau3 est supérieure à la valeur critique à 10%, on rejette H0, la série n'est pas stationnaire

```
summary(ur.df(Canada[, "prod"], type = "trend", lags = 2))
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression trend
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.19924 -0.38994  0.04294  0.41914  1.71660
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.415228  15.309403   1.987   0.0506 .
## z.lag.1     -0.075791   0.038134  -1.988   0.0505 .
## tt           0.013896   0.006422   2.164   0.0336 *
## z.diff.lag1  0.284866   0.114359   2.491   0.0149 *
## z.diff.lag2  0.080019   0.116090   0.689   0.4927
## ---
```

Estimation du modèle

- On commence par un modèle d'ordre $p = 1$
- `Type = "both"` indique qu'on inclut la constante et la tendance dans le modèle

```
p1ct <- VAR(Canada, p = 1, type = "both")
```

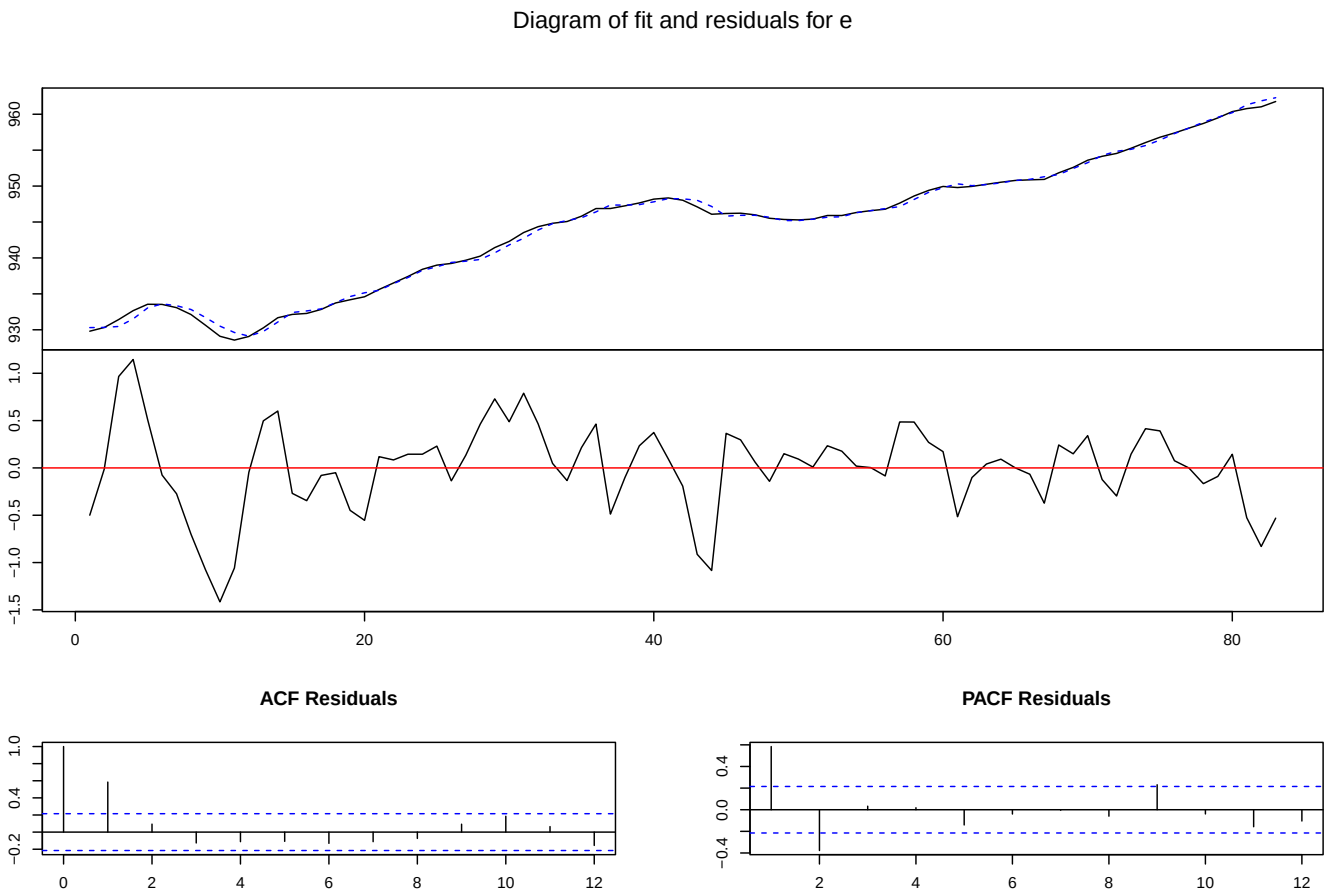
Equation pour la série e

```
summary(p1ct, equation = "e")
```

```
##
## VAR Estimation Results:
## ========================
## Endogenous variables: e, prod, rw, U
## Deterministic variables: both
## Sample size: 83
## Log Likelihood: -207.525
## Roots of the characteristic polynomial:
## 0.9504 0.9504 0.9045 0.7513
## Call:
## VAR(y = Canada, p = 1, type = "both")
##
##
## Estimation results for equation e:
## ==================================
## e = e.l1 + prod.l1 + rw.l1 + U.l1 + const + trend
##
##           Estimate Std. Error t value Pr(>|t|)
```

Graphique valeurs prédites et résidus

```
plot(p1ct, names = "e")
```



Modèles ECM

- Nous allons utliser les données `USIncExp` et suivre l'exemple de l' article présentant la librairie `strucchange`

```
USIncExp2 <- window(USIncExp, start = c(1985,12))
```

- La fonction de consommation (consumption function) est modélisée par un modèle ECM (cf. Hansen , 1992a):

$$\Delta c_t = \beta_1 + \beta_2 e_{t-1} + \beta_3 \Delta i_t + \mu_t \quad (1)$$

$$e_t = c_t - \alpha_1 - \alpha_2 i_t \quad (2)$$

où c_t représente les dépenses de consommation et i_t le revenu

Modèle ECM - Equation de cointégration

- L'équation de cointégration (2) est estimée à l'aide d'un modèle OLS

$$c_t = \alpha_1 + \alpha_2 i_t + e_t$$

- Les résidus $\hat{e}_t$ seront utilisés comme variable indépendante dans la modélisation de la fonction de consommation (1)

```
coint.res <- residuals(lm(expenditure ~ income, data = USIncExp2))
```

Modèle ECM - Préparation des données

- On utilise la fonction `lag` car les résidus dans l'équation (1) sont à $t-1$
- On utilise la fonction `diff` pour intégrer les séries (obtenir Δc_t et Δi_t)

```
coint.res <- stats::lag(ts(coint.res, start = c(1985,12), freq = 12), k = -1)
USIncExp2 <- cbind(USIncExp2, diff(USIncExp2), coint.res)
USIncExp2 <- window(USIncExp2, start = c(1986,1), end = c(2001,2))
colnames(USIncExp2) <- c("income", "expenditure", "diff.income", "diff.expenditure", "coint.res")
window(USIncExp2, start=c(1986,1), end=c(1986,12))
```

```
##          income expenditure diff.income diff.expenditure coint.res
## Jan 1986 3630.1      2829.3         6.5              11.0 27.510616
## Feb 1986 3647.7      2821.4        17.6              -7.9 33.081616
## Mar 1986 3674.8      2824.6        27.1               3.2 10.481554
## Apr 1986 3674.3      2838.6        -0.5              14.0 -8.953200
## May 1986 3686.6      2863.1        12.3              24.5  5.464415
## Jun 1986 3703.7      2869.3        17.1               6.2 19.691076
## Jul 1986 3721.3      2891.0        17.6              21.7 11.608630
## Aug 1986 3734.6      2909.9        13.3              18.9 18.608568
## Sep 1986 3752.0      2984.8        17.4              74.9 26.399998
## Oct 1986 3756.6      2947.5         4.6             -37.3 86.766982
## Nov 1986 3770.6      2945.6        14.0              -1.9 45.624921
## Dec 1986 3797.0      3016.9        26.4              71.3 32.031690
```

Modèle ECM - Estimation

- La variable dépendante est l'augmentation des dépenses and les variables indépendantes sont les résidus de cointegration et l'augmenation des revenus (plus une constante)

```
ecm.model <- diff.expenditure ~ coint.res + diff.income
summary(lm(ecm.model, data=USIncExp2))
```

```
##
## Call:
## lm(formula = ecm.model, data = USIncExp2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -70.555 -11.947  -0.528  10.452  55.576
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.82428    1.88918   9.435  < 2e-16 ***
## coint.res   -0.06843    0.03318  -2.063   0.0406 *
## diff.income  0.19001    0.04282   4.438 1.59e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.09 on 179 degrees of freedom
## Multiple R-squared:  0.1021, Adjusted R-squared:  0.09203
## F-statistic: 10.17 on 2 and 179 DF,  p-value: 6.542e-05
```