Time Series with R

Summer School on Mathematical Methods in Finance and Economy

Thibault LAURENT

Toulouse School of Economics

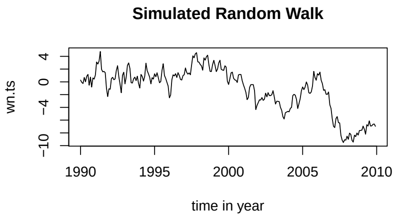
June 2010 (slides modified in August 2010)

What is a `ts` object ?

- we simulate a random walk with `cumsum` and `rnorm`:
```
> wn <- cumsum(rnorm(240, 0, 1))
```
- we create a `ts` object with the function `ts`, using a frequency of 12 (we observe a phenomenon monthly), starting in 1990:
```
> wn.ts <- ts(wn, start = 1990, frequency = 12)
```
- Time can be considered like that: one unit corresponds to one year. A year is divided into 12 (months). Thus, it is easy to visualize on the x-axis the beginning of each year. The values of time are given by the `time` function:
```
> time(wn.ts)
```
- We can print only a part of the series by using the `window` function, here for year 2001:
```
> window(wn.ts, 2001, 2001.95)
```

Plot a `ts` object

```
> plot(wn.ts, main = "Simulated Random Walk", xlab = "time in year")
```

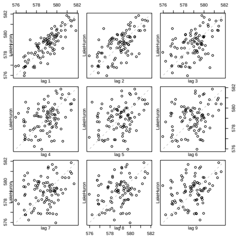


User may then use functions `lines`, `points`, `abline`, etc. to complete the graphic.

The lag plot

We can obtain a lag plot of the observations by using the function `lag.plot`, applied here to the `LakeHuron` data included in R (see `help(LakeHuron)` for more details):

```
> str(LakeHuron)
> lag.plot(LakeHuron, 9, do.lines = FALSE)
```

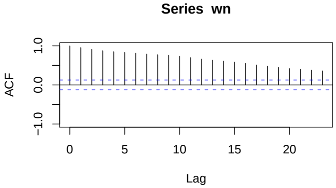


Obviously, the time series is strongly auto-correlated...

ACF/PACF graphic

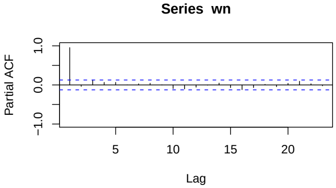
ACF graphic

```
> acf(wn, ylim = c(-1, 1))
```



PACF

```
> pacf(wn, ylim = c(-1, 1))
```

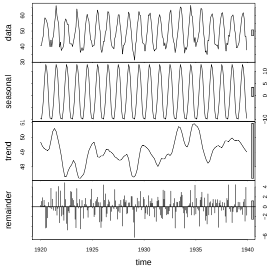


In this case, the series is not stationary (ACF decreasing non exponentially)

Structural decomposition

You can draw a decomposition of the series in trend + season + error in the case of a series with seasonality (defined with `frequency` option in function `ts`) by using the function `stl`. For example, with the `nottem` data:

```
> plot(stl(nottem, "per"))
```

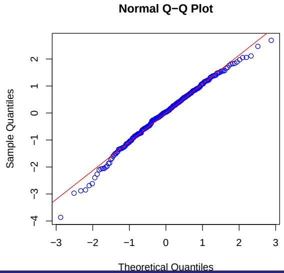


A qq-plot graphic

In the following slides, we present some tools which are useful for detecting a white noise

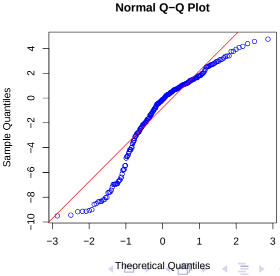
A gaussian sample:

```
> s.norm <- rnorm(250, 0, 1)
> qqnorm(s.norm, col = "blue")
> qqline(s.norm, col = "red")
```



The random walk series:

```
> qqnorm(wn, col = "blue")
> qqline(wn, col = "red")
```



Tests based on Skewness/Kurtosis values

Skewness/Kurtosis may be close to 0 if the series is white noise.

```
> require(fUtilities)
> skewness(wn)

[1] -0.8164437
attr(,"method")
[1] "moment"

> kurtosis(wn)

[1] -0.1958579
attr(,"method")
[1] "excess"
```

Tests of normality

Jarque-Bera test:

```
> require(tseries)
> jarque.bera.test(wn)

        Jarque Bera Test

data:  wn
X-squared = 27.2963, df = 2, p-value = 1.182e-06
```

Shapiro-Wilk normality test:

```
> shapiro.test(wn)

        Shapiro-Wilk normality test

data:  wn
W = 0.9218, p-value = 6.243e-10
```

For the random walk, the hypothesis of normality is not accepted in the two tests...

Ljung-Box statistic

Use of the function `Box.test` for examining the null hypothesis of independence in a given time series.

```
> Box.test(wn, lag = 1, type = "Ljung-Box")
> Box.test(wn, lag = 2, type = "Ljung-Box")
> Box.test(wn, lag = 3, type = "Ljung-Box")
```

Shortcoming of this function: it can be applied only lag by lag. To appear soon: package **outilST** developped by Aragon (2010).

Other R tools

- The function `Lag` included in **Hmisc** computes a lag vector:

```
> require(Hmisc)
> Lag(1:10, 2)
```

- The function `diff` and `diffinv` returns respectively the vectors $\Delta y_t = y_{t+1} - y_t$ and $\Delta^{-1} y_t$:

```
> diff(cumsum(1:10))
> diffinv(cumsum(1:10))
```

- The function `lowess` may be used to smooth the time series:

```
> plot(wn.ts, main = "Simulated Random Walk", xlab = "time in year'
> lines(lowess(wn.ts), col = "blue", lty = "dashed")
```

- See also `http://cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf`

Case of an AR(1)

Consider the following AR(1) model :

$$y_t = -18 - 0.8 y_{t-1} + z_t, \; t = 1, ..., 200$$

with $z_t \sim N(0, 1.5)$. Notice that $\mathbb{E}(Y) = -10$.
How can we simulate this series and analyze it ?

Simulation

1. Simulate the white noise z_t:

```
> set.seed(951)
> n2 = 250
> noise = rnorm(n2, 0, sqrt(1.5))
```

2. Apply the recurrence relation $y_t = -0.8 \times y_{t-1} + (z_t - 18)$ by using the function `filter` with a initial value $y_0 = \mathbb{E}(Y) = -10$ (`init=-10`):

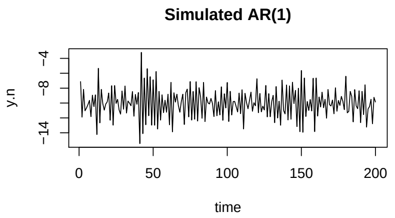
```
> noise18 = noise - 18
> y.n = filter(noise18, c(-0.8), side = 1, method = "recursive",
+       init = -10)
```

3. Delete the beginning of the series:

```
> y.n = y.n[-c(1:50)]
```

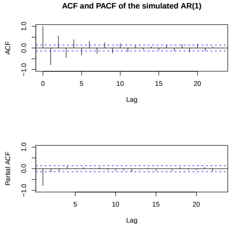
Representation of the series

```
> plot(y.n, type = "l", xlab = "time", main = "Simulated AR(1)")
```



Analysis of the ACF/PACF graphic

```
> op <- par(mfrow = c(2, 1))
> acf(y.n, main = "ACF and PACF of the simulated AR(1)",
+     ylim = c(-1, 1))
> pacf(y.n, main = "", ylim = c(-1, 1))
> par(op)
```



The analysis of the ACF (decreasing exponentially) and the PACF (close to 0 for $p > 1$) strongly suggest an AR(1).

Fitting an AR(1)

We may use the function `arima` to fit an ARIMA(p,d,q).

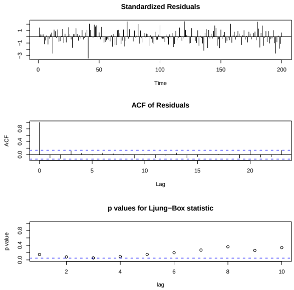
```
> (y.fit = arima(y.n, order = c(1, 0, 0)))

Call:
arima(x = y.n, order = c(1, 0, 0))

Coefficients:
          ar1  intercept
      -0.7899   -10.0269
s.e.   0.0432     0.0497

sigma^2 estimated as 1.574:  log likelihood = -329.65,  aic = 665.29
```

Notice that the value associated to `intercept` is not the intercept, but the mean (see `http://www.stat.pitt.edu/stoffer/tsa2/Rissues.htm#Issue1`)

Diagnostic plots

The object created by `arima` contains several informations and may be used by the function `tsdiag` which plots different graphics for checking that the residuals are white noise.

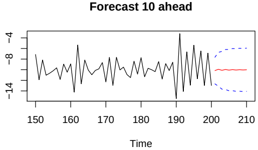
```
> tsdiag(y.fit)
```



Forecasting

Forecast 10 ahead:

```
> y.fore = predict(y.fit, n.ahead = 10)
> U = y.fore$pred + 2 * y.fore$se
> L = y.fore$pred - 2 * y.fore$se
> miny = min(y.n, L)
> maxy = max(y.n, U)
> ts.plot(window(ts(y.n, 150, 200)), y.fore$pred, col = 1:2,
+     ylim = c(miny, maxy), main = "Forecast 10 ahead")
> lines(U, col = "blue", lty = "dashed")
> lines(L, col = "blue", lty = "dashed")
```



Case of an ARIMA(1,1,2)

Consider the following ARMA(1,2) model :

$$y_t = -0.9 - 0.8y_{t-1} + z_t - 0.3z_{t-1} + 0.6z_{t-2},\ t = 1, ..., 200$$

with $z_t \sim N(0,4)$. We can re-write it as:

$$y_t = -0.5 + \frac{1-0.3B+0.6B^2}{1+0.8B} z_t,\ t = 1, ..., 200$$

The following model is an ARIMA(1,1,2):

$$\Delta y_t = -0.5 + \frac{1-0.3B+0.6B^2}{1+0.8B} z_t,\ t = 1, ..., 200$$

How can we simulate this series and analyze it ?

Simulation

1. simulate an ARMA(1,2) with the function `arima.sim`:

```
> set.seed(121181)
> yd.n = -0.5 + arima.sim(n = 200, list(ar = -0.8, ma = c(-0.3,
+     0.6)), sd = 2, n.start = 50)
```

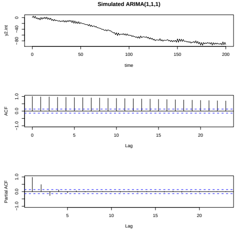
2. apply the function `diffinv` to the previous ARMA(1,2):

```
> y2.int <- diffinv(yd.n)
```

ACF and PACF of the initial series

The analysis of the ACF (decreasing non exponentially) confirms the non stationarity of the series.

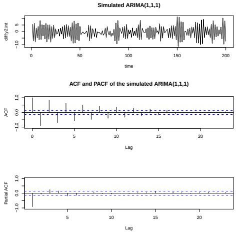
```
> op <- par(mfrow = c(3, 1))
> plot(y2.int, type = "l", xlab = "time", main = "Simulated ARIMA(1,1,1
> acf(y2.int, main = "", ylim = c(-1, 1))
> pacf(y2.int, main = "", ylim = c(-1, 1))
> par(op)
```



ACF and PACF of the differenciated series

The analysis of the differenciated series suggests an ARMA (ACF and PACF decrease exponentially).

```
> diff.y2.int <- diff(y2.int)
> op <- par(mfrow = c(3, 1))
> plot(diff.y2.int, type = "l", xlab = "time", main = "Differenciated s
> acf(diff.y2.int, main = "", ylim = c(-1, 1))
> pacf(diff.y2.int, main = "", ylim = c(-1, 1))
> par(op)
```



Identification of an ARMA(p,q)

- apply the methodology for selecting parameters in an OLS model such as "backward" or "forward", by using the function `arima`. The function `t_stat` in package **outilST** will give the p-values of each parameter.
- The MINIC (Minimum Information Criterion) method may be used to identify the parameters p and q (to appear soon: function `armaselect` in package **outilST**) which compares a specific criteria in several models.

MINIC method

`armaselect` of package `outilST` returns Schwartz' Bayesian Criterion (SBC) value for different models:

```
> armaselect(diff.y2.int, max.p = 15, max.q = 15)
```

```
     p q      sbc
[1,] 1 2 268.9689
[2,] 2 2 270.8024
[3,] 1 3 271.5127
[4,] 1 4 274.8745
[5,] 5 0 274.8792
```

It gives the ARMA(1,2) as the best model...

Fitting an ARIMA(1,1,2)

We may use the function `Arima` included in package **forecast** to fit an ARIMA(1,1,2) to the initial series.

```
> require(forecast)
This is forecast 2.04
> (y2.fit = Arima(y2.int, order = c(1, 1, 2), include.drift = TRUE))
Series: y2.int
ARIMA(1,1,2) with drift

Call: Arima(x = y2.int, order = c(1, 1, 2), include.drift = TRUE)

Coefficients:
          ar1      ma1     ma2    drift
      -0.8175  -0.2345  0.4635  -0.4593
s.e.   0.0475   0.0719  0.0743   0.0891

sigma^2 estimated as 3.473:  log likelihood = -409.5
```

Unit root tests

The package **urca** contains two functions useful to detect a possible non stationarity in the series:

- The function `ur.df` computes the Augmented Dickey-Fuller test. The choice of test (option `test='trend'` or `test='drift'`) may be suggested by the series itself...
- The function `ur.kpss` computes the Kwiatkowski test with the different options `type='tau'` or `type='mu'`.

Fit an ARMAX or SARIMA

- The option `xreg` in `Arima` may be used to fit an ARMAX. For example, to adjust the model $y_t = \beta_0 + \beta_1 x_t + u_t$, $u_t = \phi u_{t-1} + z_t$, $t = 1 \dots T$:

```
> temps = time(LakeHuron)
> mod1.lac = Arima(LakeHuron, order = c(1, 0, 0), xreg = temps,
+     method = "ML")
```

- the option `seasonal=list(order=c(P,D,Q),period=per)` may be used to fit a SARIMA. For example:

```
> fitm = Arima(nott1, order = c(1, 0, 0), list(order = c(2,
+     1, 0), period = 12))
> summary(fitm)
```

Exploratory Data Analysis

Identification of ARIMA

Case study

A case study

1. Choose a series on `http://fr.finance.yahoo.com/` and find the *Code*. Here, we choose the Danone stock price (`Code=BP.NA`)
2. Import the series by using function `priceIts` of package **its**

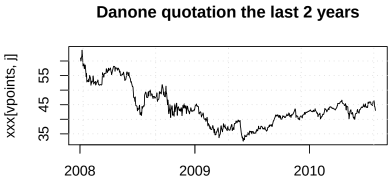
```
> require(its)
> danone = priceIts(instrument = "BN.PA", start = "2008-01-03",
+     end = "2010-07-31", quote = "Close")
> str(danone)
```

3. missing values ?

```
> manq = complete.cases(danone) == FALSE
```

Representation of the series

```
> plot(danone, main = "Danone quotation the last 2 years",
+      ylab = "in euros")
```



In general, with a financial series, we are interested by the return $\frac{y_t - y_{t-1}}{y_{t-1}}$.

Analysis of the returns

The function `returns` in package **fSeries** computes the returns and the function `its` creates an irregular time series:

```
> library(fSeries)
> y.ret <- its(returns(danone, percentage = TRUE), danone@dates)
```

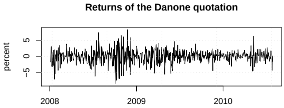
Kurtosis and Skewness tests indicate a strong heteroscedasticity (high value of kurtosis) with more negative returns than positive returns (negative value of skewness).

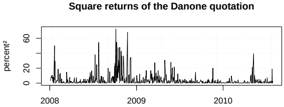
```
> require(fBasics)
> dagoTest(y.ret)
```

Representation of the returns and their square

We notice at the end of 2008 a strong variation...

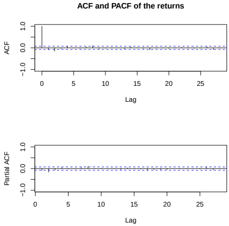
```
> op <- par(mfrow = c(2, 1))
> plot(y.ret, main = "Returns of the Danone quotation",
+      ylab = "percent")
> plot(y.ret^2, main = "Square returns of the Danone quotation",
+      ylab = "percent^2")
> par(op)
```





Analysis of the ACF/PACF graph

```
> op <- par(mfrow = c(2, 1))
> acf(na.omit(y.ret@.Data), ylim = c(-1, 1), main = "ACF and PACF of the
> pacf(na.omit(y.ret@.Data), ylim = c(-1, 1), main = "")
> par(op)
```



We thus try to adjust an AR(2)

Fit a first model

```
> (ret.fit = Arima(na.omit(y.ret@.Data), order = c(2, 0,
+     0), include.mean = FALSE))
> t_stat(ret.fit)
> tsdiag(ret.fit)
```

This model seems acceptable but we must verify whether there is heteroscedasticity in the residuals.

Conditional heteroscedasticty test

The function `ArchTest` included in package **FinTS** computes a conditional heteroscedasticty test:

```
> require(FinTS)
> rr <- ret.fit$residuals
> ArchTest(rr, lag = 12)
```

As we observe heteroscedasticty, we try to fit a GARCH(1,1), with the function `garchFit` included in package **fGarch**:

```
> require(fGarch)
> res.garch <- garchFit(~garch(1, 1), data = rr, trace = FALSE,
+     na.action = na.pass)
> summary(res.garch)
```

GARCH

To combine the AR(2) with the GARCH(1,1) applied to the residuals, we compute the following model:

```
> res2.garch <- garchFit(~arma(2, 0) + garch(1, 1), data = na.omit(y.re
+      include.mean = FALSE, trace = FALSE, na.action = na.pass)
> summary(res2.garch)
```

The model can finally be written as :

$y_t = -0.83 y_{t-1} - 0.88 y_{t-2} + \epsilon_t$ with $\epsilon_t = \sigma_t z_t$ with
$\sigma_t^2 = 0.07 + 0.1 \epsilon_{t-1} + 0.88 \sigma_{t-1}^2$

Prediction of a GARCH

```
> pred.zcond = predict(res2.garch, n.ahead = 30, trace = FALSE,
+      mse = "cond", plot = TRUE)
```