Chapter 2
Time Series

Time series provide another class of models for sequences of data points indexed by discrete time. This chapter reviews the main time series models - moving average, autoregressive, and integrated - and their properties, such as stationarity, which is considered using autocovariance and unit root tests. Several examples illustrate the fitting of time series models to financial data in R and Python. We conclude with an application to a pair trading algorithm on a financial market, using the Dickey–Fuller stationarity test.

2.1 Autoregressive Moving Average

We use a step-by-step approach to the construction of time series, starting with the most basic setting of independent sequences, and building progressively towards more interaction. In what follows, we let

$$\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$

denote the set of all (signed) integers.

White noise

A white noise sequence is a sequence $(Z_n)_{n\in\mathbb{Z}}$ of independent, centered and identically distributed random variables with unit variance, with

$$\mathbb{E}[Z_n] = 0, \quad \text{and} \quad \mathrm{Cov}(Z_n, Z_m) = \mathbb{1}_{\{n=m\}}, \qquad n, m \in \mathbb{Z}.$$

```r
Zn<-rnorm(100,0,1); Zn
```

Listing 2.1: R code - White noise generation.

```python
import numpy as np
Zn = np.random.normal(loc=0, scale=1, size=100)
```

Listing 2.2: Python code - White noise generation.

Moving Average (MA) model

Definition 2.1. *Let $\beta_1, \ldots, \beta_q$ be a sequence of real deterministic coefficients such that $\beta_q \neq 0$, $q \geqslant 1$. In the MA(q) model of order q, the current state X_n of the system is expressed as the linear combination*

$$\begin{aligned} X_n &:= Z_n + \beta_1 Z_{n-1} + \cdots + \beta_q Z_{n-q} \\ &= Z_n + \sum_{k=1}^{q} \beta_k Z_{n-k}, \qquad n \geqslant 0, \end{aligned} \tag{2.1}$$

of the q previous values $Z_{n-1}, \ldots, Z_{n-q}$.

We will use the "lag operator" or "backward time shift operator" L defined as

$$LZ_n := Z_{n-1}, \qquad n \geqslant 1. \tag{2.2}$$

```r
library(zoo)
Zn<-zoo(rnorm(5,0,1)); lag(Zn,-1, na.pad= TRUE)
```

Listing 2.3: R code - Lag operator.

```python
import pandas as pd; import numpy as np
Zn= pd.Series(np.random.normal(0, 1, 5)); lagged_Zn= Zn.shift(1)
```

Listing 2.4: Python code - Lag operator.

The lag operator L can be iterated as

$$L^k Z_n = Z_{n-k}, \qquad n \in \mathbb{Z},\ k \geqslant 0,$$

and can be used to rewrite (2.1) as

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

$$\begin{aligned} X_n &= Z_n + \beta_1 L Z_n + \cdots + \beta_q L^q Z_n \\ &= Z_n + \sum_{k=1}^{q} \beta_k L^k Z_n \\ &= Z_n + \psi(L) Z_n, \qquad n \geqslant q, \end{aligned}$$

where

$$\psi(L) := \beta_1 L + \cdots + \beta_q L^q = \sum_{k=1}^{q} \beta_k L^k$$

is the *moving average operator* given by the function

$$\psi(x) := \beta_1 x + \cdots + \beta_q x^q = \sum_{k=1}^{q} \beta_k x^k.$$

Example: generating MA(2) samples

The next codes generate samples of the MA(2) times series

$$X_n := Z_n - 0.7 \times Z_{n-1} + 0.1 \times Z_{n-2},$$

with $\beta_1 = -0.7$ and $\beta_2 = 0.1$.

```
n=41;ma<-arima.sim(model=list(ma=c(-.7,.1)),n.st=100,n)
x=seq(100,100+n-1);dev.new(width=12,height=6)
plot(x,ma,pch=19,ylab='',xlab='n',main='',col='blue',cex.axis=1.8,cex.lab=1.5,las=1)
lines(x,ma,col='blue');grid()
```

Listing 2.5: R code - MA(2) time series generation.

```
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess;n=41;n_s=100
ma_c=np.array([1,.7,-.1]);ar_c=np.array([1.0]);arma_p=ArmaProcess(ar_c, ma_c)
ma_sim=arma_p.generate_sample(nsample=n+n_s)[n_s:];x=np.arange(100,100+n)
plt.figure(figsize=(12,6));plt.plot(x,ma_sim,'ob-',c='b',label='')
plt.xlabel('n',fontsize=14);plt.title('');plt.grid(True)
plt.tick_params(axis='both', which='major', labelsize=12); plt.show()
```

Listing 2.6: Python code - MA(2) time series generation.

The ARIMA command uses a parameter "n.start", here taken equal to 100, which creates a "burn-in" initial time interval which ensures sufficient randomness in the behavior of X_n.

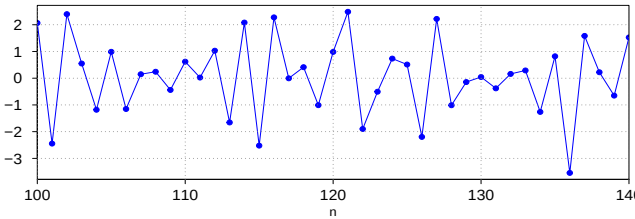


Fig. 2.1: MA(2) Samples.

Autoregressive (AR) model

In the simplest AR(1) model, the current state X_n of the system is expressed in feedback form as

$$X_n := Z_n + \alpha_1 X_{n-1}, \qquad n \geqslant 1, \tag{2.3}$$

Definition 2.2. *Let $\alpha_1, \ldots, \alpha_p$ be a sequence of real deterministic coefficients such that $\alpha_p \neq 0$, $p \geqslant 1$. In the AR(p) model of order p, the state X_n of the system is expressed as the linear feedback combination*

$$\begin{aligned} X_n &:= Z_n + \alpha_1 X_{n-1} + \cdots + \alpha_p X_{n-p} \\ &= Z_n + \sum_{k=1}^{p} \alpha_k X_{n-k}, \qquad n \geqslant p, \end{aligned} \tag{2.4}$$

of the p previous values $X_{n-1}, \ldots, X_{n-p}$ of the time series.

Using again the lag operator L defined in (2.2), we can rewrite (2.4) as

$$\begin{aligned} X_n &= Z_n + \alpha_1 L X_n + \cdots + \alpha_p L^p X_n \\ &= Z_n + \sum_{k=1}^{p} \alpha_k L^k X_n \\ &= Z_n + \phi(L) X_n, \qquad n \geqslant p, \end{aligned}$$

where

$$\phi(L) := \alpha_1 L + \cdots + \alpha_p L^p = \sum_{k=1}^{p} \alpha_k L^k$$

is the operator given by the function

$$\phi(x) := \alpha_1 x + \cdots + \alpha_p x^p = \sum_{k=1}^{p} \alpha_k x^k.$$

Proposition 2.3. *The equation*

$$X_n := Z_n + \alpha_1 X_{n-1}, \qquad n \in \mathbb{Z}, \tag{2.5}$$

defines an AR(1) time series $(X_n)_{n\in\mathbb{Z}}$*, and can be solved recursively in the following cases:*

a) When $|\alpha_1| < 1$*,* (2.5) *admits the converging* causal *moving average solution*

$$X_n = \sum_{k\geqslant 0} \alpha_1^k Z_{n-k}, \qquad n \in \mathbb{Z}, \tag{2.6}$$

with

$$\mathrm{Var}[X_n] = \sum_{k\geqslant 0} |\alpha_1|^{2k} = \frac{1}{1-|\alpha_1|^2} \geqslant 1, \qquad n \in \mathbb{Z}. \tag{2.7}$$

b) When $|\alpha_1| > 1$*,* (2.5) *admits the converging* non-causal *moving average solution*

$$X_n = -\sum_{k\geqslant 1} \frac{1}{\alpha_1^k} Z_{n+k}, \qquad n \in \mathbb{Z}, \tag{2.8}$$

with

$$\mathrm{Var}[X_n] = \sum_{k\geqslant 1} |\alpha_1|^{-2k} = \frac{1}{|\alpha_1|^2 - 1}, \qquad n \in \mathbb{Z}. \tag{2.9}$$

No such converging solutions exist when $|\alpha_1| = 1$*.*

Proof. *a)* When $|\alpha_1| < 1$ we may write, using backward induction,

$$\begin{aligned}
X_n &= Z_n + \alpha_1 X_{n-1} \\
&= Z_n + \alpha_1 (Z_{n-1} + \alpha_1 X_{n-2}) \\
&= Z_n + \alpha_1 (Z_{n-1} + \alpha_1 (Z_{n-2} + \alpha_1 X_{n-3})) \\
&= Z_n + \alpha_1 (Z_{n-1} + \alpha_1 (Z_{n-2} + \alpha_1 (Z_{n-3} + \alpha_1 X_{n-4}))) \\
&= Z_n + \alpha_1 Z_{n-1} + \alpha_1^2 Z_{n-2} + \alpha_1^3 Z_{n-3} + \alpha_1^4 X_{n-4} \\
&= \cdots \\
&= \sum_{k\geqslant 0} \alpha_1^k Z_{n-k}, \quad n \in \mathbb{Z},
\end{aligned}$$

which converges when the solution $z = 1/\alpha_1$ of the equation $\phi(z) = \alpha_1 z = 1$ satisfies $|\alpha_1| < 1$, *i.e.* $|z| > 1$.

b) When $|\alpha_1| > 1$, we write

$$X_n = -\alpha_1^{-1} Z_{n+1} + \alpha_1^{-1} X_{n+1}, \qquad n \geqslant 0,$$

which can be solved by forward induction as

$$\begin{aligned} X_n &= -\alpha_1^{-1} Z_{n+1} + \alpha_1^{-1} X_{n+1} \\ &= -\alpha_1^{-1} Z_{n+1} + \alpha_1^{-1}(-\alpha_1^{-1} Z_{n+2} + \alpha_1^{-1} X_{n+2}) \\ &= -\alpha_1^{-1} Z_{n+1} + \alpha_1^{-1}(-\alpha_1^{-1} Z_{n+2} + \alpha_1^{-1}(-\alpha_1^{-1} Z_{n+3} + \alpha_1^{-1} X_{n+3})) \\ &= -\alpha_1^{-1} Z_{n+1} - \alpha_1^{-2} Z_{n+2} - \alpha_1^{-3} Z_{n+3} + \alpha_1^{-4} X_{n+3} \\ &= \cdots \\ &= -\sum_{k \geqslant 1} \frac{1}{\alpha_1^k} Z_{n+k}, \quad n \in \mathbb{Z}, \end{aligned}$$

when the solution $z = 1/\alpha_1$ of the equation $\phi(z) = \alpha_1 z = 1$ satisfies $|z| < 1$. □

Example: generating AR(2) samples

The following R and Python codes generate samples of the AR(2) times series

$$X_n := Z_n + 0.9 \times X_{n-1} - 0.2 \times X_{n-2},$$

with $\alpha_1 = 0.9$ and $\alpha_2 = -0.1$.

```
n=41;ar.sim<-arima.sim(model=list(ar=c(.9,-.2)),n.st=100,n)
x=seq(100,100+n-1);dev.new(width=12,height=6)
plot(x,ar.sim,pch=19,ylab='',xlab='n',main='',col='blue',cex.axis=1.8,cex.lab=1.5,las=1)
lines(x,ar.sim,col='blue');grid()
```

Listing 2.7: R code - AR(2) time series generation.

```
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess; n= 41; n_s= 100
ar_c= np.array([1,-.9,.2]); ma_c= np.array([1]); ar_p= ArmaProcess(ar_c, ma_c)
ar_sim= ar_p.generate_sample(nsample=n + n_s)[n_s:]; x= np.arange(100, 100 + n)
plt.figure(figsize=(12,6));plt.plot(x,ar_sim,'o',c='b',label='')
plt.plot(x,ar_sim,'-',c='b');plt.xlabel('n',fontsize=14);plt.title('')
plt.grid(True);plt.tick_params(axis='both', which='major', labelsize=12); plt.show()
```

Listing 2.8: Python code - AR(2) time series generation.

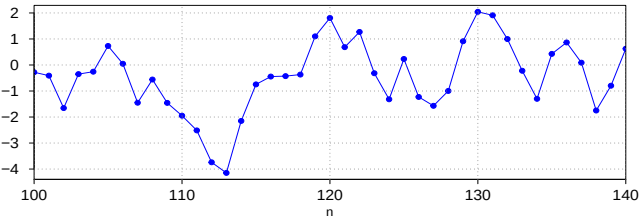


Fig. 2.2: AR(2) Samples.

Autoregressive Moving Average (ARMA) model

Definition 2.4. *Let $\alpha_1, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ be sequences of real deterministic coefficients such that $\alpha_p \neq 0$ and $\beta_q \neq 0$, $p, q \geqslant 1$. In the ARMA(p, q) model of orders p, q, the current state X_n of the system is expressed as the linear combination*

$$\begin{aligned} X_n &:= Z_n + \alpha_1 X_{n-1} + \cdots + \alpha_p X_{n-p} + \beta_1 Z_{n-1} + \cdots + \beta_p Z_{n-q} \\ &= Z_n + \sum_{k=1}^{p} \alpha_k X_{n-k} + \sum_{k=1}^{q} \beta_k Z_{n-k}, \end{aligned} \tag{2.10}$$

of the p previous values $X_{n-1}, \ldots, X_{n-p}$ and $Z_{n-1}, \ldots, Z_{n-p}$, $n \geqslant \text{Max}(p, q)$.

Using again the lag operator L defined in (2.2), we can rewrite (2.10) as

$$\begin{aligned} X_n &= Z_n + \sum_{k=1}^{p} \alpha_k L^k X_n + \sum_{k=1}^{q} \beta_k L^k Z_n \\ &= Z_n + \phi(L) X_n + \psi(L) Z_n, \qquad n \geqslant 1. \end{aligned}$$

Example: generating ARMA$(2, 2)$ samples

The next R and Python codes generate samples of the ARMA$(2, 2)$ times series

$$X_n := Z_n + 0.9 \times X_{n-1} - 0.2 \times X_{n-2} - 0.7 \times Z_{n-1} + 0.1 \times Z_{n-2},$$

with $\alpha_1 = 0.9$, $\alpha_2 = -0.1$, and $\beta_1 = -0.7$, $\beta_2 = 0.1$.

```r
n=41; arma.sim<-arima.sim(model=list(ar=c(.9,-.2),ma=c(-.7,.1)),n.st=100,n)
x=seq(100,100+n-1); dev.new(width=12,height=6)
plot(x,arma.sim,pch=19,ylab='',xlab='n',main='',col='blue',cex.axis=1.8,cex.lab=1.5,las=1)
lines(x,arma.sim,col='blue');grid()
```

Listing 2.9: R code - ARMA(2,2) time series generation.

```python
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess;n=41;n_s=100;
ar_c=np.array([1,-.9,.2]);ma_c=np.array([1,.7,-.1]);arma_p=ArmaProcess(ar=ar_c,ma=ma_c)
arma_s= arma_p.generate_sample(nsample=n + n_s)[n_s:];x= np.arange(100, 100 + n);
plt.figure(figsize=(12,6));plt.plot(x,arma_s,'ob-',label='');plt.xlabel('n');plt.title('')
plt.grid(True);plt.tick_params(axis='both', which='major', labelsize=12);plt.show()
```

Listing 2.10: Python code - ARMA(2,2) time series generation.

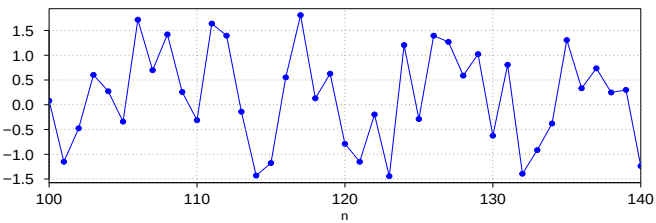


Fig. 2.3: ARMA(2, 2) Samples.

Autoregressive Integrated Moving Average (ARIMA) model

Consider the difference operator ∇ defined as

$$\nabla := \mathrm{I} - L$$

where I is the identity operator which satisfies $\mathrm{I}X_n = X_n$, and L is the lag operator defined in (2.2). We have

$$\nabla X_n := X_n - X_{n-1}, \qquad n \geqslant 1,$$

and the operation $\nabla X_n = X_n - X_{n-1}$ can be iterated as follows:

$$\begin{aligned} \nabla^2 X_n &= \nabla\nabla X_n \\ &= \nabla(X_n - X_{n-1}) \\ &= \nabla X_n - \nabla X_{n-1} \\ &= X_n - X_{n-1} - (X_{n-1} - X_{n-2}) \\ &= X_n - 2X_{n-1} + X_{n-2}, \end{aligned} \tag{2.11}$$

and

$$\begin{aligned}\nabla^3 X_n &= \nabla\nabla^2 X_n \\ &= \nabla X_n - 2\nabla X_{n-1} + \nabla X_{n-2} \\ &= X_n - X_{n-1} - 2(X_{n-1} - X_{n-2}) + X_{n-2} - X_{n-3} \\ &= X_n - 3X_{n-1} + 3X_{n-2} - X_{n-3}. \end{aligned} \tag{2.12}$$

The time series $(X_n)_{k\geqslant 1}$ can be recovered by integrating $(\nabla X_k)_{k\geqslant 1}$ using the telescoping identity

$$X_n = X_0 + \sum_{k=1}^{n}(X_k - X_{k-1}) = X_0 + \sum_{k=1}^{n}\nabla X_k, \quad n \geqslant 1. \tag{2.13}$$

More generally, the time series $(X_n)_{n\geqslant 0}$ can be recovered by successive applications of the discrete integration formula (2.13) as in Proposition 2.5 which extends (2.11)-(2.12) and (2.13), using the convention $\nabla^0 = \mathrm{I}$.

Proposition 2.5. *a) The iterated operator ∇^d satisfies*

$$\nabla^d X_n = \sum_{k=0}^{d}\binom{d}{k}(-1)^k X_{n-k}, \quad n \geqslant d \geqslant 0.$$

b) The time series $(X_n)_{n\geqslant d}$ can be recovered from $\nabla X_n, \nabla^2 X_n, \ldots, \nabla^d X_n$ as

$$X_n = X_{n-d} + \sum_{k=1}^{d}\binom{d}{k}\nabla^k X_{n+k-d}, \quad n \geqslant d \geqslant 0.$$

Proof. (a) This is a consequence of the binomial operator identity

$$\begin{aligned}\nabla^d &= (\mathrm{I} - L)^d \\ &= \sum_{k=0}^{d}\binom{d}{k}(\mathrm{I})^{n-k}(-L)^k \\ &= \sum_{k=0}^{d}\binom{d}{k}(-1)^k L^k. \end{aligned}$$

(b) Apply the binomial operator identity

$$\mathrm{I} = (\mathrm{I} - L + L)^d = (L + \nabla)^d = \sum_{k=0}^{d}\binom{d}{k}L^{d-k}\nabla^k,$$

and note that the operators L and ∇ commute. □

Definition 2.6. *Let $\alpha_1, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ be sequences of nonnegative deterministic coefficients such that $\alpha_p \neq 0$ and $\beta_q \neq 0$, $p, q \geqslant 1$. In the ARIMA(p, d, q) model of orders p, d, q, the iterated difference process $(\nabla^d X_n)_{n \geqslant 0}$ is modeled as the ARMA(p, q) time series*

$$\begin{aligned}\nabla^d X_n := {} & Z_n + \alpha_1 \nabla^d X_{n-1} + \cdots + \alpha_p \nabla^d X_{n-p} \\ & + \beta_1 Z_{n-1} + \cdots + \beta_q Z_{n-q} \\ = {} & Z_n + \sum_{k=1}^{p} \alpha_k \nabla^d X_{n-k} + \sum_{k=1}^{q} \beta_k Z_{n-k}, \end{aligned} \tag{2.14}$$

$n \geqslant \mathrm{Max}(p + d, q + d)$, based on the p previous values $\nabla^d X_{n-1}, \ldots, \nabla^d X_{n-p}$ and $Z_{n-1}, \ldots, Z_{n-q}$.

Using again the lag operator L defined in (2.2), we can rewrite (2.14) as

$$\nabla^d X_n = Z_n + \phi(L) \nabla^d X_n + \psi(L) Z_n, \tag{2.15}$$

$n \geqslant \mathrm{Max}(p + d, q + d)$, where the functions $\phi(z)$ and $\psi(z)$ are given by

$$\phi(z) = \sum_{k=1}^{p} \alpha_k z^k \quad \text{and} \quad \psi(z) = \sum_{k=1}^{p} \beta_k z^k.$$

In other words, (2.15) rewrites as

$$(\mathrm{I} - \phi(L)) \nabla^d X_n = Z_n + \psi(L) Z_n,$$

$n \geqslant \mathrm{Max}(p + d, q + d)$. The time series $(X_n)_{n \geqslant 0}$ can then be recovered by successive applications of the discrete integration formula (2.13), or using Proposition 2.5-(b). Alternatively, we can start by recovering $\nabla^{d-1} X_n$ from $\nabla^d X_n$ as

$$\begin{aligned}\nabla^{d-1} X_n &= \nabla^{d-1} X_0 + \sum_{k=1}^{n} \left(\nabla^{d-1} X_k - \nabla^{d-1} X_{k-1} \right) \\ &= \nabla^{d-1} X_0 + \sum_{k=1}^{n} \nabla^d X_k, \end{aligned}$$

followed by

$$\nabla^{d-2} X_n, \nabla^{d-3} X_n, \ldots, \nabla^2 X_n, \nabla X_n, X_n,$$

by induction on $n \geqslant d$.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

Example: generating ARIMA$(1,2,3)$ samples

The next R and Python codes generate samples of the ARIMA$(1,2,3)$ times series $(X_n)_{n\geqslant 5}$ defined from (2.13) and

$$\nabla X_n := Z_n + 0.5 \times X_{n-1} + 0.5 \times Z_{n-1} + 0.5 \times Z_{n-2} - 0.5 \times Z_{n-3},$$

with $\alpha_1 = 0.5$ and $\beta_1 = 0.5$, $\beta_2 = 0.5$, $\beta_3 = -0.5$.

```
n=41;d=2;arima.s<-arima.sim(model=list(ar=c(.5),ma=c(.5,.5,-.5), order=c(1,d,3)), n.st=100,n)
x=seq(100,100+n+d-1); dev.new(width=12,height=6)
plot(x,arima.s,pch=19, ylab='', xlab='n', main= paste("", sep=' '),cex.axis=1.8,
  cex.lab=1.5,las=1); lines(x,arima.s,col='blue');grid()
```

Listing 2.11: R code - ARIMA(1,2,3) time series generation.

```
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess;
n=41;d=2;n_s=100;n_total=n+n_s+d;ar=np.array([1,.5);ma=np.array([1,.5,.5,-.5]);
arma_p=ArmaProcess(ar, ma);siml= arma_p.generate_sample(nsample=n_total);
siml= np.cumsum(np.cumsum(siml));sim_dt=siml[-(n+d):];x=np.arange(100,100+n+d);
plt.figure(figsize=(12,6));plt.plot(x,sim_dt,'ob-');plt.xlabel("n");plt.grid(True)
plt.title("");plt.tick_params(axis='both',which='major',labelsize=12);plt.show()
```

Listing 2.12: Python code - ARIMA(1,2,3) time series generation.

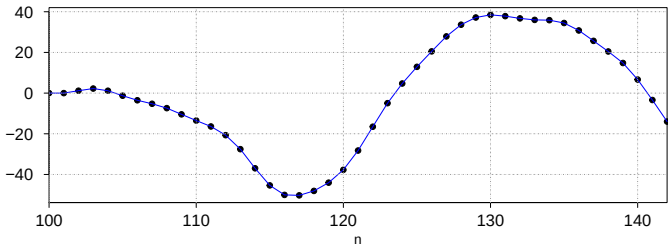


Fig. 2.4: ARIMA$(1,2,3)$ Samples.

Note that the ARIMA graph of Figure 2.4 has more potential for prediction than the ARMA graph of Figure 2.3 due to increased dependence on past samples in the considered model, or longer memory.

2.2 Autoregressive Heteroskedasticity

As above, we consider a white noise sequence $(Z_n)_{n\in\mathbb{Z}}$ of independent, centered and identically distributed with unit variance.

Autoregressive Conditional Heteroskedasticity (ARCH) model

Heteroskedasticity refers to time-dependent variance in a time series.

Definition 2.7. *Let $\alpha_0, \ldots, \alpha_p \geqslant 0$ be a sequence of nonnegative deterministic coefficients such that $\alpha_p \neq 0$, $p \geqslant 1$. In the ARCH(p) model of order p, the current state X_n of the system is expressed by the equation*

$$X_n := \sigma_n Z_n, \qquad n \geqslant 0, \tag{2.16}$$

where $\sigma_n > 0$ is given by

$$\begin{aligned} \sigma_n^2 &= \alpha_0 + \sum_{k=1}^{p} \alpha_k X_{n-k}^2 \\ &= \alpha_0 + \sum_{k=1}^{p} \alpha_k \sigma_{n-k}^2 Z_{n-k}^2, \qquad n \geqslant p. \end{aligned} \tag{2.17}$$

Using the lag operator L defined in (2.2), we can rewrite (2.17) as

$$\begin{aligned} \sigma_n^2 &= \alpha_0 + \sum_{k=1}^{p} \alpha_k L^k X_n^2 \\ &= \alpha_0 + \psi(L) X_n^2, \qquad n \geqslant p, \end{aligned}$$

where

$$\psi(L) := \alpha_1 L + \cdots + \alpha_p L^p = \sum_{k=1}^{p} \alpha_k L^k.$$

In particular, noting that σ_n is independent of Z_n since σ_n depends only on $Z_{n-1}, Z_{n-2}, \ldots$, we have $\mathbb{E}[X_n] = 0$ and

$$\mathbb{E}[X_n^2] = \mathbb{E}[\sigma_n^2 Z_n^2] = \mathbb{E}[Z_n^2]\mathbb{E}[\sigma_n^2] = \mathrm{Var}[Z_n]\mathbb{E}[\sigma_n^2] = \mathbb{E}[\sigma_n^2],$$

hence the recursion

$$\mathbb{E}[\sigma_n^2] = \alpha_0 + \sum_{k=1}^{p} \alpha_k \mathbb{E}[X_{n-k}^2] = \alpha_0 + \sum_{k=1}^{p} \alpha_k \mathbb{E}[\sigma_{n-k}^2].$$

Example: generating ARCH(2) samples

The next R and Python codes generate samples of the ARCH(2) times series $(X_n)_{n \geqslant 5}$ with variance process given by

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

$$\sigma_n^2 = \alpha_0 + 0.2 \times X_{n-1}^2 + 0.4 \times X_{n-2}^2, \qquad n \geqslant 2, \tag{2.18}$$

with $\alpha_1 = 0.2$ and $\alpha_2 = 0.4$.

```
library(fGarch); arch.spec<-garchSpec(model=list(alpha0= 1e-5,alpha=c(.2,.4),beta= 0))
n=100; arch.sim<-garchSim(arch.spec,n); x=seq(1,n); dev.new(width=12,height=6)
plot(x,arch.sim,pch=19,ylab='',xlab='n',main='',col='blue',cex.axis=1.5,cex.lab=1.7,las=1)
lines(x,arch.sim,col='blue');grid()
```

Listing 2.13: R code - ARCH(2) time series generation.

```
import numpy as np, matplotlib.pyplot as plt
from arch import arch_model;np.random.seed(42);n=100
arch_s=arch_model(None,mean='Zero',vol='ARCH',p=2).simulate(np.array([1e-5,.2,.4]),n).data
plt.figure(figsize=(12, 6)); x= np.arange(1, n+1); plt.plot(x, arch_s, 'bo-', ms=6, alpha=.7)
plt.title('');plt.xlabel('n');plt.grid(True,ls='--',alpha=.5);plt.tight_layout();plt.show()
```

Listing 2.14: Python code - ARCH(2) time series generation.

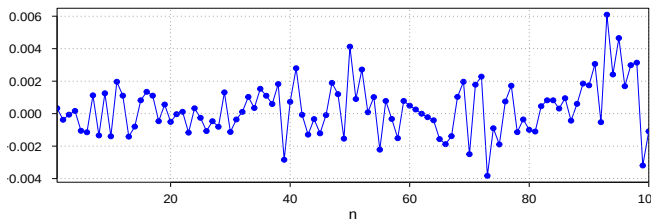


Fig. 2.5: ARCH(2) Samples.

Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model

Definition 2.8. *Let $\alpha_0, \ldots, \alpha_p \geqslant 0$ and $\beta_1, \ldots, \beta_q \geqslant 0$ be sequences of non-negative deterministic coefficients such that $\alpha_p \neq 0$ and $\beta_q \neq 0$, $p, q \geqslant 1$. In the GARCH(p, q) model with orders p, q, the current state X_n of the system is expressed by the equation (2.16), where $\sigma_n > 0$ is given by*

$$\begin{aligned} \sigma_n^2 &= \alpha_0 + \sum_{k=1}^{p} \alpha_k X_{n-k}^2 + \sum_{k=1}^{q} \beta_k \sigma_{n-k}^2 \\ &= \alpha_0 + \sum_{k=1}^{p} \alpha_k \sigma_{n-k}^2 Z_{n-k}^2 + \sum_{k=1}^{q} \beta_k \sigma_{n-k}^2, \quad n \geqslant \mathrm{Max}(p, q). \end{aligned} \tag{2.19}$$

Using the lag operator L, we can rewrite (2.19) as

$$\begin{aligned}\sigma_n^2 &= \alpha_0 + \sum_{k=1}^{p} \alpha_k L^k X_n^2 + \sum_{k=1}^{q} \beta_k L^k \sigma_n^2 \\ &= \alpha_0 + \phi(L) X_n^2 + \psi(L)\sigma_n^2, \qquad n \geqslant \mathrm{Max}(p,q),\end{aligned}$$

where

$$\phi(L) := \alpha_1 L + \cdots + \alpha_q L^q = \sum_{k=1}^{q} \alpha_k L^k$$

and

$$\psi(L) := \beta_1 L + \cdots + \beta_q L^q = \sum_{k=1}^{q} \beta_k L^k.$$

As above, we have

$$\mathbb{E}[X_n^2] = \mathbb{E}[\sigma_n^2 Z_n^2] = \mathbb{E}[Z_n^2]\mathbb{E}[\sigma_n^2] = \mathbb{E}[\sigma_n^2],$$

hence

$$\begin{aligned}\mathbb{E}[X_n^2] &= \alpha_0 + \sum_{k=1}^{p} \alpha_k \mathbb{E}[X_{n-k}^2] + \sum_{k=1}^{q} \beta_k \mathbb{E}[\sigma_{n-k}^2] \\ &= \alpha_0 + \sum_{k=1}^{p} \alpha_k \mathbb{E}[\sigma_{n-k}^2] + \sum_{k=1}^{q} \beta_k \mathbb{E}[\sigma_{n-k}^2], \quad n \geqslant \mathrm{Max}(p,q).\end{aligned}$$

Example: generating GARCH(2, 1) samples

The next R and Pythoncodes generate samples of the GARCH$(2,1)$ times series times series $(X_n)_{n\geqslant 5}$ with variance process given by

$$\sigma_n^2 = \alpha_0 + 0.2 \times X_{n-1}^2 + 0.4 \times X_{n-2}^2 + 0.3 \times \sigma_{n-1}^2, \quad n \geqslant 2, \tag{2.20}$$

where $\alpha_1 = 0.2$, $\alpha_2 = 0.4$, and $\beta_1 = 0.3$.

```
library(fGarch); garch.sp<-garchSpec(model=list(alpha0= 1e-5,alpha=c(.2,.4),beta=c(.3)))
n=100; garch.sim<-garchSim(garch.sp,n); x=seq(1,n); dev.new(width=12,height=6)
plot(x,garch.sim,pch=19,ylab='',xlab='n',main='',col='blue',cex.axis=1.5,cex.lab=1.7,las=1);
lines(x,garch.sim,col='blue');grid()
```

Listing 2.15: R code - GARCH(2,1) time series generation.

The package arch can be similarly used in Python.

```
import numpy as np, matplotlib.pyplot as plt; from arch import arch_model
np.random.seed(42); plt.figure(figsize=(12,6));plt.xlabel('n')
plt.plot(np.arange(1,101), arch_model(None, mean='Zero', vol='GARCH', p=2,
  q=1).simulate([1e-5,.2,.4,.3], 100).data, 'bo-', ms=6, alpha=.7)
plt.title('');plt.grid(True,ls='--',alpha=.5);plt.tight_layout();plt.show()
```

Listing 2.16: Python code - GARCH(2,1) time series generation.

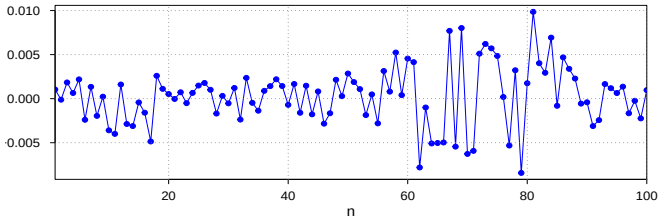


Fig. 2.6: GARCH$(2,1)$ Samples.

Similarly to Proposition 2.3, we have the following result.

Proposition 2.9. GARCH$(1,1)$ *model. The equations* $X_n = \sigma_n Z_n$ *and*

$$\sigma_n^2 := \alpha_0 + \alpha_1 X_{n-1}^2 + \beta_1 \sigma_{n-1}^2, \quad n \in \mathbb{Z},$$

with $\alpha_1, \beta_1 \geqslant 0$ *and* $\alpha_1 + \beta_1 < 1$*, define a* GARCH$(1,1)$ *time series* $(X_n)_{n\in\mathbb{Z}}$ *which admits the causal solution*

$$\sigma_n^2 = \alpha_0 \sum_{k\leqslant n} \prod_{l=k}^{n-1} \big(\alpha_1 Z_l^2 + \beta_1\big), \qquad n \in \mathbb{Z}, \tag{2.21}$$

with

$$\mathbb{E}[\sigma_n^2] = \alpha_0 \sum_{k\geqslant 0} (\alpha_1 + \beta_1)^k = \frac{\alpha_0}{1-\alpha_1-\beta_1}, \qquad n \in \mathbb{Z}. \tag{2.22}$$

No such converging solution exists when $\alpha_1 + \beta_1 \geqslant 1$.

2.3 Time Series Stationarity

Strict stationarity

Definition 2.10. *A time series* $(X_n)_{n\in\mathbb{Z}}$ *is* strictly stationary *with order* $p \geqslant 1$ *if the equality in distribution*

$$(X_n, X_{n-1}, \ldots, X_{n-p}) \overset{d}{=} (X_{n+m}, X_{n+m-1}, \ldots, X_{n+m-p}),$$

holds for all $n \in \mathbb{Z}$ *and* $m, p \geqslant 0$.

In other words, Definition 2.10 states that the random vectors

$$(X_n, X_{n-1}, \ldots, X_{n-p}) \quad \text{and} \quad (X_{n+m}, X_{n+m-1}, \ldots, X_{n+m-p})$$

have same distribution for all $m \in \mathbb{Z}$ and $p \geqslant 1$.

Example. The MA(q) time series

$$\begin{aligned} X_n &= Z_n + \beta_1 Z_{n-1} + \cdots + \beta_q Z_{n-q} \\ &= Z_n + \sum_{k=1}^{q} \beta_k Z_{n-k}, \qquad n \geqslant q, \end{aligned}$$

is *strictly stationary* of any order $p \geqslant 1$, due to the equality in distribution

$$\begin{aligned} &\left(Z_n + \sum_{k=1}^{q} \beta_k Z_{n-k}, \ldots, Z_{n-p} + \sum_{k=1}^{q} \beta_k Z_{n-p-k} \right) \\ &\quad \overset{d}{=} \left(Z_{n+m} + \sum_{k=1}^{q} \beta_k Z_{n+m-k}, \ldots, Z_{n+m-p} + \sum_{k=1}^{q} \beta_k Z_{n+m-p-k} \right), \end{aligned}$$

as $(Z_n)_{n \geqslant 0}$ is an *i.i.d.* sequence.

Weak stationarity

Definition 2.11. *A time series* $(X_n)_{n \geqslant 0}$ *is* weakly stationary *if*

i) $\mathbb{E}[X_n] = \mathbb{E}[X_0]$, $n \geqslant 0$, *and*

ii) the autocovariance *

$$(n, m) \longmapsto \mathrm{Cov}(X_n, X_m)$$

depends only on the absolute difference $|n - m|$, $n, m \geqslant 0$.

The autocovariances $\mathrm{Cov}(X_n, X_{n+l})$ and cross-covariances $\mathrm{Cov}(Y_n, X_{n+l})$ of time series with lag parameter $l \in \mathbb{Z}$ can be respectively estimated as follows:

```
n=1000; ar.sim<-arima.sim(model=list(ar=c(.9,-.2)),n.st=100,n)
ar.acf<-acf(ar.sim,type="covariance",plot=T,col='blue',lwd=4); dev.new(width=12,height=6)
ar.ccf<-ccf(ar.sim,ar.sim,type="covariance",plot=T,lwd=4,col='blue',main='',cex.axis=1.8,
  cex.lab=1.5,las=1);grid()
```

Listing 2.17: R code - Autocovariance function.

* The covariance $\mathrm{Cov}(X, Y)$ is defined as $\mathrm{Cov}(X, Y) := \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$.

```
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess; import statsmodels.api as sm
ar_sim= ArmaProcess(ar=[.9,-.2], ma=[1]).generate_sample(1000, burnin=100)
acf_vals= sm.tsa.acf(ar_sim, fft=False, nlags=30);lags= np.arange(-30, 31);
two_sided_acf= np.concatenate([acf_vals[:0:-1], acf_vals]);plt.figure(figsize=(12,6));
plt.stem(lags,two_sided_acf,linefmt='b',markerfmt='bo',basefmt=" ");plt.grid(True,alpha=.3)
plt.title('Two-sided Autocorrelation Function',fontsize=16);plt.tight_layout()
plt.xlabel('Lag',fontsize=14);plt.ylabel('Autocorrelation',fontsize=14);plt.show()
```

Listing 2.18: Python code - Autocovariance function.

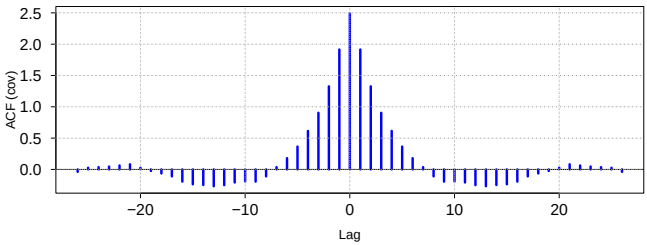


Fig. 2.7: Autocovariances of AR(2) Samples.

By representing an AR(q) series as a vector-valued AR(1) series we can obtain the following result, see *e.g.* Theorem 3.1.1 page 89 of Brockwell and Davis (1991) and Theorem 4.4 page 119 of Pourahmadi (2001).

Theorem 2.12. *Consider the AR(q) time series $(X_n)_{n\geqslant q}$ solution of*

$$X_n := Z_n + \phi(L)X_n = Z_n + \alpha_1 X_{n-1} + \cdots + \alpha_q X_{n-q},$$

with

$$\phi(z) = \alpha_1 z + \cdots + \alpha_q z^q, \qquad z \in \mathbb{C}.$$

1) Unit root test. *The AR(q) time series $(X_n)_{n\geqslant 0}$ is* weakly stationary *with lag order q if and only if* no *(complex) solution of the equation $\phi(z) = 1$ lies* on *the complex unit circle $\{z \in \mathbb{C} \ : \ |z| = 1\}$ in the complex plane $\mathbb{C}$.*[*]

2) Causality. *The AR(q) time series $(X_n)_{n\geqslant q}$ admits a* causal *expression if and only if the equation $\phi(z) = 1$ has no solution inside the complex unit disk $\{z \in \mathbb{C} \ : \ |z| \leqslant 1\}$.*

Examples

i) In the AR(1) example

$$X_n := Z_n + \alpha_1 X_{n-1} = Z_n + \phi(L)X_n, \qquad n \geqslant 1,$$

[*] See (MOE and UCLES 2022, page 15) or (MOE and UCLES 2022, page 20).

with $\phi(z) = \alpha_1 z$, the unique solution of the equation $\phi(z) = \alpha_1 z = 1$ is $z = 1/\alpha_1$, and it lies outside the complex unit circle if and only if $\alpha_1 \neq \pm 1$, *i.e.* $|\alpha_1| \neq 1$. Hence, by Theorem 2.12-1) the time series $(X_n)_{n\geqslant 2}$ is (weakly) stationarity if and only if $|\alpha_1| \neq 1$.

In this case, by Proposition 2.3 we have the series representations

$$\begin{cases} X_n = \displaystyle\sum_{k\geqslant 0} \alpha_1^k Z_{n-k}, & |\alpha_1| < 1, \qquad (2.23a) \\ X_n = -\displaystyle\sum_{k\geqslant 1} \frac{1}{\alpha_1^k} Z_{n+k}, & |\alpha_1| > 1, \qquad (2.23b) \end{cases}$$

which converge when $|\alpha_1| \neq 1$, with

$$\mathrm{Var}[X_n] = \mathbb{E}\big[X_n^2\big] = \frac{1}{|1-|\alpha_1|^2|}, \qquad n \geqslant 1,$$

see (2.7) and (2.9).

a) In the case of (2.23a) with $|\alpha_1| < 1$, by Theorem 2.12-2) we have the causal solution

$$X_n = \sum_{k\geqslant 0} \alpha_1^k Z_{n-k}, \qquad n \geqslant 0,$$

satisfies

$$\begin{aligned} \mathrm{Cov}(X_n, X_m) &= \mathbb{E}\left[\sum_{k\geqslant 0} \alpha_1^k Z_{n-k} \sum_{l\geqslant 0} \overline{\alpha}_1^l Z_{m-l}\right] \\ &= \alpha_1^{n-m} \sum_{k\geqslant 0} |\alpha_1|^{2k} \\ &= \frac{\alpha_1^{n-m}}{1-|\alpha_1|^2}, \qquad n \geqslant m \geqslant 0. \end{aligned}$$

b) In the case of (2.23b) with $|\alpha_1| > 1$, the non-causal solution

$$X_n = -\sum_{k\geqslant 1} \frac{1}{\alpha_1^k} Z_{n+k}, \qquad n \geqslant 0,$$

satisfies

$$\mathrm{Cov}(X_n, X_m) = \mathbb{E}\left[\sum_{k\geqslant 1} \frac{1}{\alpha_1^k} Z_{n+k} \sum_{l\geqslant 1} \frac{1}{\overline{\alpha}_1^l} Z_{m+l}\right]$$

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

$$= \frac{\alpha_1^{m-n}}{|\alpha_1|^2 - 1}, \qquad n \geqslant m \geqslant 0.$$

We note that the expressions (2.6)-(2.7) in the case $|\alpha_1| < 1$ correspond to strictly stationary times series, while (2.8)-(2.9) in the case $|\alpha_1| > 1$ correspond to weakly, but not strictly, stationary times series.

ii) In the AR(2) example

$$X_n := Z_n + 0.9 \times X_{n-1} - 0.2 \times X_{n-2} \tag{2.24}$$

of Figure 2.2 with $\phi(z) = 0.9z - 0.2z^2$, the solutions

$$z_+ = \frac{0.9 + \sqrt{0.9^2 - 4 \times 0.2}}{2 \times 0.2} = \frac{5}{2}, \quad z_- = \frac{0.9 - \sqrt{0.9^2 - 4 \times 0.2}}{2 \times 0.2} = 2$$

of the equation $\phi(z) = 1$ do not lie on the complex unit circle, hence by Theorem 2.12-1) the time series $(X_n)_{n \geqslant 2}$ is (weakly) stationarity. In addition, since $z_\pm$ lie outside the unit disk, Theorem 2.12-2) shows that the time series $(X_n)_{n \geqslant 2}$ has a causal expression.

```
N=20000;t=0:(N-1);a1=.9;a2=-.2;X=c(1,N);X[1]=1;X[2]=2;dev.new(width=12,height=8)
ar.sim=arima.sim(model=list(ar=c(a1,a2)),n.st=100,N);Z=rnorm(N,mean=0,sd=1)
for (k in 3:N){X[k]=Z[k]+a1*X[k-1]+a2*X[k-2];}
plot(t,X,pch=19, ylab='', xlab='n', col= "blue",cex.axis=1.8, cex.lab=1.5,las=1)
lines(t,X, col= "blue"); grid() # Weakly stationary
```

Listing 2.19: R code - Stationary AR(2) time series generation.

```
import numpy as np,matplotlib.pyplot as plt;N=20000;t=np.arange(N);a1=.9;a2=-.2
from statsmodels.tsa.arima_process import ArmaProcess
ar_c=np.array([1,-a1,-a2]); ma_c=np.array([1]); ar_p=ArmaProcess(ar_c,ma_c)
ar2_sim=ar_p.generate_sample(nsample=N)
plt.figure(figsize=(12,8)); plt.scatter(t,ar2_sim,s=10,color='blue',marker='o',alpha=0.6)
plt.plot(t,ar2_sim,color='blue',linewidth=0.5); plt.xlabel('');plt.ylabel('')
plt.tick_params(axis='both',which='major',labelsize=12); plt.grid(True); plt.show()
```

Listing 2.20: Python code - Stationary AR(2) time series generation.

iii) Consider the AR(2) time series

$$X_n := Z_n + X_{n-1} - X_{n-2} = Z_n + \phi(L)X_n, \qquad n \geqslant 2,$$

with $\phi(z) = z - z^2$.

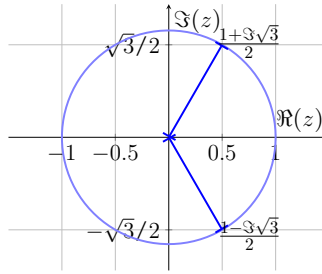


Fig. 2.8: Complex unit circle.

The solutions $z_\pm = (1 \pm i\sqrt{3})/2$ of the equation $\phi(z) = 1$ lie on the unit circle, hence by Theorem 2.12-1) the time series $(X_n)_{n\geqslant 2}$ is *not* (weakly) stationarity. In addition, since $z_\pm = (1 \pm i\sqrt{3})/2$ lie inside the unit disk, Theorem 2.12-2) shows that the time series $(X_n)_{n\geqslant 2}$ has no causal expression.

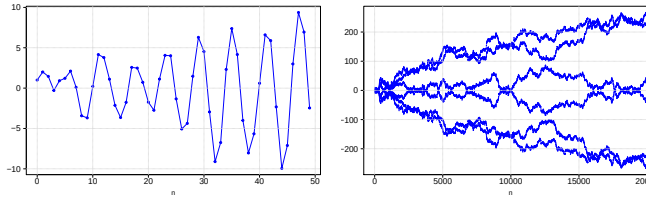
```r
library(tseries); N=1000;t=0:(N-1); a1=1.0;a2=-1.0; # Not weakly stationary
ar.sim<-arima.sim(model=list(ar=c(a1,a2)),n.st=100,N);Z<- rnorm(N,mean=0,sd=1)
X=c(1,N);X[1]=1;X[2]=2;for (k in
 3:N){X[k]=Z[k]+a1*X[k-1]+a2*X[k-2];};dev.new(width=12,height=8)
plot(t,X,pch=19, ylab='', xlab='n', col= "blue",cex.axis=1.8, cex.lab=1.5,las=1)
lines(t,X, col= "blue"); grid()
```

Listing 2.21: R code - Non-stationary AR(2) time series.

```python
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess; n= 41; n_s= 100
ar_c= np.array([1, 1, -1]); ma_c= np.array([1]); ar_p= ArmaProcess(ar_c, ma_c)
ar_sim= ar_p.generate_sample(nsample=n + n_s)[n_s:]; x= np.arange(100, 100 + n)
plt.figure(figsize=(12,6));plt.plot(x,ar_sim,'ob-',label='');plt.xlabel('n');plt.title('')
plt.grid(True);plt.tick_params(axis='both',which='major',labelsize=12);plt.show()
```

Listing 2.22: Python code - Non-stationary AR(2) time series.

Figure 2.9 presents a simulation of a non-stationary AR(2) time series with $\alpha_1 = 1$ and $\alpha_2 = -1$.



(a) 50 time steps. (b) $20,000$ time steps.

Fig. 2.9: Nonstationarity of an AR(2) time series with $\alpha_1 = 1$ and $\alpha_2 = -1$.

iv) Consider the AR(2) time series

$$X_n := Z_n + 2X_{n-1} + 2X_{n-2} = Z_n + \phi(L)X_n, \qquad n \geqslant 2,$$

with $\phi(z) = 2z + 2z^2$. The solutions $z_\pm = (-1 \pm \sqrt{3})/2$ of the equation $\phi(z) = 1$ lie outside the unit circle, hence by Theorem 2.12-1) the time series $(X_n)_{n\geqslant 2}$ is (weakly) stationarity. However, since $z_+ = (-1+\sqrt{3})/2$ lies inside the unit disk, Theorem 2.12-2) shows that the time series $(X_n)_{n\geqslant 2}$ has no causal expression.

```r
N=20000;t<-0:(N-1);a1=2.;a2=2.;Z<-rnorm(N,mean=0,sd=1) # No causal solution
X<-c(1,N);X[1]=1;X[2]=2;for (k in 3:N){X[k]=Z[k]+a1*X[k-1]+a2*X[k-2];};
dev.new(width=12,height=8);plot(t,X,pch=19,ylab='',xlab='n',col="blue",cex.axis=1.8,
  cex.lab=1.5,las=1); lines(t,X, col= "blue"); grid()
```

Listing 2.23: R code - Non-causal AR(2) time series generation.

```python
import numpy as np, matplotlib.pyplot as plt;N=20000;t=np.arange(0,N);a1=2.;a2=2.;
Z= np.random.normal(0, 1, N); X= np.zeros(N); X[0]= 1; X[1]= 2;
[exec(f"X[{k}]= Z[{k}] + a1 * X[{k-1}] + a2 * X[{k-2}]") for k in range(2, N)];
plt.figure(figsize=(12, 8)); plt.plot(t, X, 'o', ms=1, c='b'); plt.plot(t, X, '-', c='b');
plt.xlabel("n", fontsize=15); plt.ylabel('', fontsize=15);
plt.tick_params(axis='both', which='major', labelsize=18); plt.grid(True); plt.show()
```

Listing 2.24: Python code - Non-causal AR(2) time series generation.

v) Consider the AR(2) time series

$$X_n := Z_n + 2X_{n-1} - 2X_{n-2} = Z_n + \phi(L)X_n, \qquad n \geqslant 2,$$

with $\phi(z) = 2z - 2z^2$. The solutions $z_\pm = (1 \pm i)/2$ of the equation $\phi(z) = 1$ lie outside the unit circle, hence by Theorem 2.12-1) the time series $(X_n)_{n\geqslant 2}$ is (weakly) stationarity. However, since $z_\pm = (1 \pm i)/2$ lie inside the unit disk, Theorem 2.12-2) shows that the time series $(X_n)_{n\geqslant 2}$ has no causal expression.

```r
N=20000; t<- 0:(N-1); a1=2.0;a2=-2; # Weakly stationary, no causal solution
Z<- rnorm(N,mean=0,sd=1);X<- c(1,N); X[1]=1;X[2]=2;
for (k in 3:N){X[k]=Z[k]+a1*X[k-1]+a2*X[k-2];}; dev.new(width=12,height=8)
plot(t,X,pch=19, ylab='', xlab='n', col= "blue",cex.axis=1.8, cex.lab=1.5,las=1)
lines(t,X, col= "blue"); grid()
```

Listing 2.25: R code - Non-causal AR(2) time series generation.

```python
import numpy as np, matplotlib.pyplot as plt; N=2000;t=np.arange(0,N);
a1=2.;a2=-2;Z=np.random.normal(0,1,N);X=np.zeros(N);X[0]=1;X[1]=2;
[exec(f"X[{k}]= Z[{k}] + a1 * X[{k-1}] + a2 * X[{k-2}]") for k in range(2, N)];
plt.figure(figsize=(12, 8)); plt.plot(t, X, 'o', ms=1, c='b'); plt.plot(t, X, '-', c='b');
plt.xlabel("n", fontsize=15); plt.ylabel('', fontsize=15);
plt.tick_params(axis='both', which='major', labelsize=18); plt.grid(True); plt.show()
```

Listing 2.26: Python code - Non-causal AR(2) time series generation.

Stationarity test

The Dickey–Fuller test allows us to test the null (nonstationarity) hypothesis H_0, *i.e.* "$|\alpha_1| = 1$", against the alternative (stationarity) hypothesis "$|\alpha_1| \neq 1$".

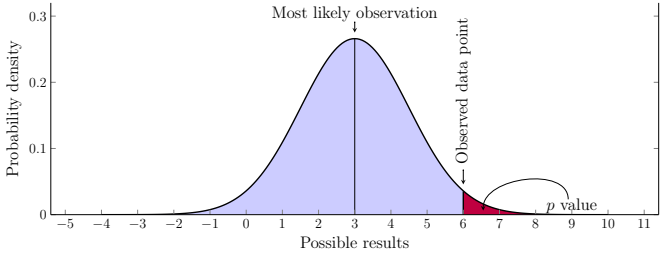


Fig. 2.10: Hypothesis testing.

a) Under the alternative (stationarity) hypothesis $|\alpha_1| \neq 1$, consider the residual

$$\sum_{t=1}^{n} (X_t - \alpha_1 X_{t-1})^2$$

representing the quadratic distance between $(X_t)_{1 \leqslant t \leqslant n}$ and $(\alpha_1 X_{t-1})_{1 \leqslant t \leqslant n}$. By Ordinary Linear Regression (OLS), the value of α_1 that minimizes this distance is given by

$$\widehat{\alpha}_1^{(n)} := \frac{\sum_{t=1}^{n} X_{t-1}X_t}{\sum_{t=1}^{n} X_{t-1}^2}, \qquad n \geqslant 1,$$

which can be rewritten as

$$\widehat{\alpha}_1^{(n)} = \frac{\sum_{t=1}^{n} X_{t-1}(Z_t + \alpha_1 X_{t-1})}{\sum_{t=1}^{n} X_{t-1}^2} = \alpha_1 + \frac{\sum_{t=1}^{n} X_{t-1}Z_t}{\sum_{t=1}^{n} X_{t-1}^2}.$$

When $|\alpha_1| \neq 1$, by (2.7), (2.9) and the Central Limit Theorem the renormalized test statistics

$$\sqrt{n}\big(\widehat{\alpha}_1^{(n)} - \alpha_1\big)$$

converges in distribution to $\mathcal{N}(0, 1 - |\alpha_1|^2)$ as n tends to infinity, see Chapters 8 and 17 of Hamilton (1994). Informally, we have

$$\begin{aligned}
\sqrt{n}\big(\widehat{\alpha}_1^{(n)} - \alpha_1\big) &\simeq \sqrt{n}\frac{\sum_{t=1}^{n} X_{t-1}Z_t}{\sum_{t=1}^{n} X_{t-1}^2} \\
&\simeq \frac{|1-|\alpha_1|^2|}{\sqrt{n}} \sum_{t=1}^{n} X_{t-1}Z_t \\
&\simeq \sqrt{\frac{|1-|\alpha_1|^2|}{n}} \sum_{t=1}^{n} Z_t \\
&\simeq \sqrt{\frac{|1-|\alpha_1|^2|}{n}} \mathcal{N}(0, n) \\
&\simeq \mathcal{N}(0, |1-|\alpha_1|^2|).
\end{aligned}$$

b) Under the null (nonstationarity) hypothesis H_0, *i.e.* $|\alpha_1| = 1$, the test statistics

$$\widehat{\alpha}_1^{(n)} := \frac{\sum_{t=1}^{n} X_{t-1}X_t}{\sum_{t=1}^{n} X_{t-1}^2},$$

and

$$\hat{t}^{(n)} := \frac{\sum_{t=1}^{n} X_{t-1} Z_t}{\sqrt{\sum_{t=1}^{n} X_{t-1}^2 \sum_{k=1}^{n} \left(X_t - \widehat{\alpha}_1^{(n)} X_{k-1}\right)^2 / (n-1)}}$$

converge respectively in distribution as n tends to infinity to the random variables

$$\frac{\int_0^1 B_s dB_s}{\int_0^1 B_s^2 ds} \quad \text{and} \quad \frac{\int_0^1 B_s dB_s}{\sqrt{\int_0^1 B_s^2 ds}},$$

where $(B_s)_{s\in[0,1]}$ is a standard Brownian motion, see Equations (17.4.2)-(17.4.7) and (17.4.11)-(17.4.12) in § 17.4 of Hamilton (1989) and § 6.1 in Tanaka (2017).

Those asymptotic distributions can be used to test the null hypothesis H_0, *i.e.* "$|\alpha_1| = 1$", against the alternative stationarity hypothesis "$|\alpha_1| \neq 1$".

The (Augmented) Dickey–Fuller stationarity test uses an additional lag parameter. It is implemented in the 'tseries' R package, as follows:

```
install.packages('tseries'); library('tseries'); adf.test(ar.sim); adf.test(arima.s)
```

Listing 2.27: R code - ADF statistical tests.

```
from statsmodels.tsa.stattools import adfuller; import numpy as np;
result_ar=adfuller(ar2_sim);print(f"ADF test for AR process:");print(f"p-value: {result_ar[1]}")
```

Listing 2.28: Python code - ADF statistical tests.

The output of the adf.test command leads us to reject the nonstationarity (null) hypothesis H_0 at the level 1% for the AR(2) time series (2.24) of Figure 2.2:

```
        Augmented Dickey–Fuller Test
data: ar.sim
Dickey–Fuller = -13.765, Lag order = 16, p-value = 0.01
alternative hypothesis: stationary
Warning message:
In adf.test(ar.sim) : p-value smaller than printed p-value
```

Applying the Augmented Dickey–Fuller Test to the ARIMA time series of Figure 2.4 would not allow us to reject the nonstationarity (null) hypothesis H_0. Other stationarity tests for time series include the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, which relies on linear regression.

2.4 Fitting Time Series to Financial Data

Market data can be fitted to an ARIMA(p, d, q) model in R and Python.

Fitting market returns

The following commands can be used to fit an ARIMA model to data.

```r
arima(data,order=c(p,d,q))
```

Listing 2.29: R code - Fitting an ARIMA model to data.

```python
import numpy as np; from statsmodels.tsa.arima.model import ARIMA
model= ARIMA(data, order=(1, 1, 1))
```

Listing 2.30: Python code - Fitting an ARIMA model to data.

For an example based on market returns, we can use the following data set.

```r
library(quantmod); getSymbols("1800.HK",from="2013-01-01",to="2014-11-30",src="yahoo")
stock=Ad(`1800.HK`); stock=stock[!is.na(stock)];
stock.rtn=(stock-lag(stock))/lag(stock)[-1]; dev.new(width=12,height=6)
stock.rtn=stock.rtn[!is.na(stock.rtn)];chartSeries(stock.rtn,up.col="blue",theme="white")
```

Listing 2.31: R code - Fetching market returns.

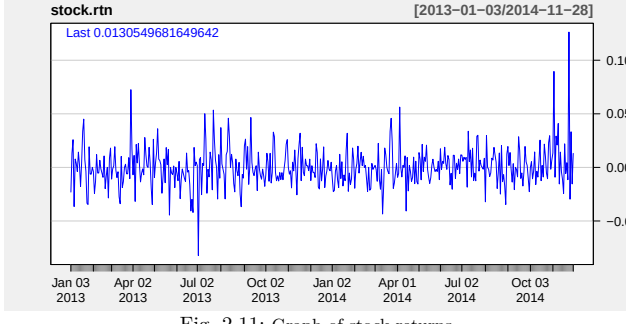


Fig. 2.11: Graph of stock returns.

```python
import yfinance as yf, matplotlib.pyplot as plt;
import pandas as pd, mplfinance as mpf, pandas_datareader.data as web
stk_d = yf.download("1800.HK")
stock_rtn= stk_d['Close'].pct_change().dropna(); plt.figure(figsize=(12, 6))
plt.plot(stock_rtn.index, stock_rtn.values, 'b'); plt.title('1800.HK Returns')
plt.xlabel('Date'); plt.ylabel('Returns'); plt.grid(True); plt.show()
print(f"Number of observations: {len(stock_rtn)}")
```

Listing 2.32: Python code - Fetching market returns.

```
library(forecast)
auto.arima(stock.rtn);acf(stock.rtn,type="covariance",plot=T,col='blue',lwd=4);
```

Listing 2.33: R code - Fitting an ARIMA model to market returns.

The output of the auto.arima command identifies these data to a white noise, as ARIMA(0,0,0).

```
Series: stock.rtn
ARIMA(0,0,0) with zero mean
sigma^2 estimated as 0.0003266: log likelihood=1219.37
AIC=-2436.74 AICc=-2436.73 BIC=-2432.58
```

```
from statsmodels.tsa.stattools import acf; from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt; model= ARIMA(stock_rtn, order=(1,0,1)).fit();
print(model.summary()); autocovariance= acf(stock_rtn, nlags=20)
plt.stem(range(len(autocovariance)),autocovariance,linefmt='blue',markerfmt='bo',basefmt=' ')
plt.axhline(y=0, c='k', ls='-', alpha=.3); plt.show()
```

Listing 2.34: Python code - Fitting an ARIMA model to market returns.

We can also fit these data to an MA(3) time series using the following commands.

```
arima(stock.rtn,order=c(0,0,3))
```

Listing 2.35: R code - Fitting an MA(3) model to market returns.

```
from statsmodels.tsa.arima.model import ARIMA
model= ARIMA(stock_rtn, order=(0,0,3)).fit(); print(model.summary())
```

Listing 2.36: Python code - Fitting an MA(3) model to market returns.

This yields the output:

```
Coefficients:
      ma1     ma2     ma3
      0.0029 0.0470 -0.0416
s.e. 0.0452 0.0467 0.0465
```

and AIC=-2430.19. Sample data from this time series can be generated (up to rescaling) from the following codes.

```
n=length(stock.rtn); x=seq(100,100+n-1); myPars=chart_pars();myPars$cex=1.4;
arima.s=arima.sim(model=list(ma=c(.0029,.0470,-.0416),order=c(0,0,3)),n.st=100,n)
dates=index(stock.rtn);ma=xts(x=arima.s, order.by= dates)*sd(stock.rtn);
myTh=chart_theme();myTh$col$line.col="blue";myTh$rylab=F;dev.new(width=12,height=8)
par(mfrow=c(1,2));myPars$mar=c(3,2,0,1);chart_Series(stock.rtn,theme=myTh,pars=myPars)
graph<-chart_Series(ma,theme=myTh,pars=myPars);myylim<-graph$get_ylim()
myylim[[2]]<-structure(c(min(stock.rtn),max(stock.rtn)),fixed=T);graph$set_ylim(myylim);graph
```

Listing 2.37: R code - Calibrated time series generation.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

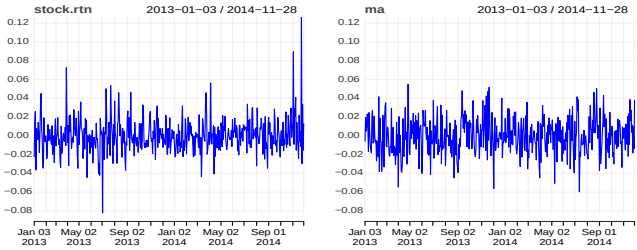


Fig. 2.12: Calibrated ARIMA$(0,0,3)$ samples vs. market returns.

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess;n=len(stock_rtn);ar=np.array([1])
ma= np.array([1,.0029,.0470,-.0416]);ma_p= ArmaProcess(ar, ma)
arima_s= ma_p.generate_sample(nsample=n, burnin=100)
std_val= float(stock_rtn.std()); ma_series= pd.Series(arima_s* std_val)
stock_rtn_n= pd.Series(stock_rtn.values.squeeze());fig, (ax1, ax2)=plt.subplots(1, 2,
 figsize=(16, 8))
ax1.plot(stock_rtn_n.index,stock_rtn_n.values,c='b',lw=0.4);ax1.set_title('Original Returns')
y_min,y_max=float(stock_rtn_n.min()),float(stock_rtn_n.max())
ax2.plot(ma_series.index, ma_series.values, c='b', lw=0.4); ax2.set_ylim(y_min, y_max);
ax2.set_title('Simulated MA(3) Process'); plt.tight_layout(); plt.show()
```

Listing 2.38: Python code - Calibrated time series generation.

Next, we fit these data to an ARMA(2,2) time series using the command

```
arima(stock.rtn,order=c(2,0,2))
```

Listing 2.39: R code - Fitting ARMA(2,2) to market returns.

with the following output:

Coefficients:

	ar1	ar2	ma1	ma2
	-1.0593	-0.9048	1.0509	0.9508
s.e.	0.0679	0.0444	0.0474	0.0273

and AIC=-2432.57.

```
from statsmodels.tsa.arima.model import ARIMA
model= ARIMA(stock_rtn, order=(2,0,2)).fit(); print(model.summary())
```

Listing 2.40: Python code - Fitting ARMA(2,2) to market returns.

Sample data from this time series can be generated (up to rescaling) from the following code.

```
n=length(stock.rtn); x=seq(100,100+n-1); dates=index(stock.rtn);myPars=chart_pars();
arima.s=arima.sim(model=list(ar=c(-1.0593,-.9048),ma=c(1.05,.95),order=c(2,0,2)), n.st=100,n)
ar=xts(x=arima.s, order.by= dates)*sd(stock.rtn);myPars$cex=1.4; myTh=chart_theme();
myTh$col$line.col="blue";myTh$rylab=F; dev.new(width=12,height=8); par(mfrow=c(1,2));
myPars$mar=c(3,2,0,1);chart_Series(stock.rtn,theme=myTh,pars=myPars)
graph=chart_Series(ar,theme=myTh,pars=myPars); myylim=graph$get_ylim()
myylim[[2]]=structure(c(min(stock.rtn),max(stock.rtn)),fixed=T);graph$set_ylim(myylim);graph
```

Listing 2.41: R code - Calibrated time series generation.

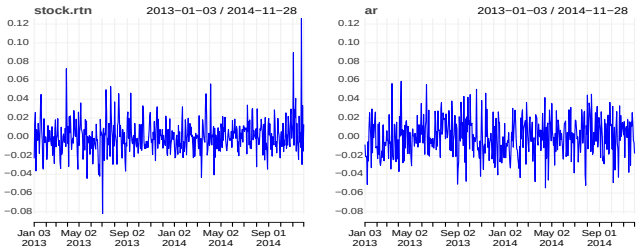


Fig. 2.13: Calibrated ARIMA$(2,0,2)$ samples vs. market returns.

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt; n= len(stock_rtn);
from statsmodels.tsa.arima_process import ArmaProcess;arima_s= np.random.normal(0,1,n)
std_val= stock_rtn.std().iloc[0] if hasattr(stock_rtn.std(), 'iloc') else float(stock_rtn.std())
ar_s= pd.Series(arima_s * std_val); stock_rtn_n= pd.Series(stock_rtn.values.flatten())
fig, (ax1, ax2)= plt.subplots(1, 2, figsize=(16, 8)); ax1.plot(stock_rtn_n.values, c='b', lw=0.4)
ax1.set_title('Original Stock Returns');ax2.plot(ar_s.values,c='b',lw=0.4);plt.tight_layout()
ax2.set_ylim(stock_rtn_n.min(),stock_rtn_n.max());ax2.set_title('ARMA Process');plt.show()
```

Listing 2.42: Python code - Calibrated time series generation.

These data can also be fit to a GARCH(1,1) time series using the following command.

```
library(fGarch); garchFit(~ garch(1,1), data= stock.rtn, trace= FALSE)
```

Listing 2.43: R code - Fitting GARCH(1,1) to market returns.

This gives the following output.

```
Coefficients:
mu          omega      alpha1     beta1
-1.593e-04  9.124e-06  4.522e-02  9.308e-01
```

The package arch can be similarly used in Python.

```
from arch import arch_model; model=arch_model(stock_rtn,vol='Garch',p=1,q=1)
result=model.fit(update_freq=0,disp='off');print(result.summary())
```

Listing 2.44: Python code - Fitting GARCH(1,1) to market returns.

Sample data from this time series can be generated (up to rescaling) from the following codes.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

```
n=length(stock.rtn); myPars<- chart_pars(); myPars$cex<-1.4;myTh<- chart_theme();
garch.sp<-garchSpec(model=list(alpha0= 9.124e-06, alpha=c(4.522e-02),beta= c(9.308e-01)))
garch.sim<-garchSim(garch.sp,n); x=seq(100,100+n-1); dates<- index(stock.rtn)
garch<-xts(x=garch.sim, order.by= dates)*sd(stock.rtn)/sd(garch.sim);
myTh$col$line.col<- "blue"; myTh$rylab<- FALSE; dev.new(width=12,height=8)
par(mfrow=c(1,2));myPars$mar=c(3,2,0,1);chart_Series(stock.rtn,theme=myTh,pars=myPars)
graph<-chart_Series(garch,theme=myTh,pars=myPars); myylim<- graph$get_ylim()
myylim[[2]]<-structure(c(min(stock.rtn),max(stock.rtn)),fixed=T);graph$set_ylim(myylim);graph
```

Listing 2.45: R code - Calibrated GARCH time series generation.

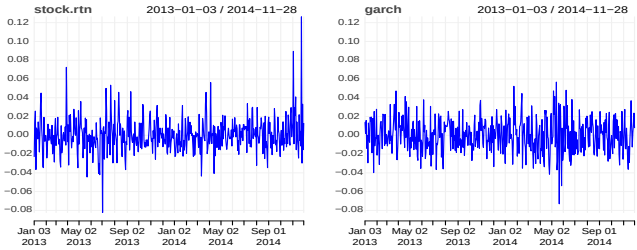


Fig. 2.14: Calibrated GARCH samples vs. market returns.

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
n=len(stock_rtn);garch_s=np.random.normal(0,1,n)
std_val= stock_rtn.std().iloc[0] if hasattr(stock_rtn.std(), 'iloc') else float(stock_rtn.std())
garch_series= pd.Series(garch_s* std_val); stock_rtn_n= pd.Series(stock_rtn.values.flatten())
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(16,8));ax1.plot(stock_rtn_n.values,c='b',lw=0.4)
plt.tight_layout(); ax1.set_title('Original Returns'); ax2.plot(garch_series.values, c='b', lw=0.4)
ax2.set_ylim(stock_rtn_n.min(),stock_rtn_n.max());ax2.set_title('');plt.show()
```

Listing 2.46: Python code - Calibrated GARCH time series generation.

Fitting market prices

Next, we fit market price data to an ARIMA time series.

```
library(quantmod); getSymbols("1800.HK",from="2007-01-03",to="2011-12-02",src="yahoo")
stock=Ad(`1800.HK`); chartSeries(stock,up.col="blue",theme="white")
n= length(stock); arima(stock,order=c(2,1,2))
```

Listing 2.47: R code - Fitting ARIMA(2,1,2) to market prices.

Fig. 2.15: Cumulative stock returns.

```
import yfinance as yf,matplotlib.pyplot as plt,pandas as pd,mplfinance as mpf
import pandas_datareader.data as web; import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA; stock= yf.download("1800.HK")
stk_d= stock['Close']; plt.figure(figsize=(12, 6)); plt.plot(stk_d, c='b')
plt.title('1800.HK Adjusted Close Price');plt.ylabel('Price'); plt.grid(True); plt.show()
model= ARIMA(stk_d, order=(2,1,2)).fit(); print(model.summary());n= len(stk_d)
```

Listing 2.48: Python code - Fitting ARIMA(2,1,2) to market prices.

The output of the auto.arima(stock) command identifies these data to an ARIMA(2,1,0) time series of integrated order one.

```
Series: stock
ARIMA(2,1,0)
Coefficients: ar1 ar2 0.0605 -0.0006 s.e. 0.0288 0.0288
sigma^2 = 0.05082: log likelihood = 84.47
AIC=-162.94 AICc=-162.92 BIC=-147.63
```

We may also fit these data to an ARIMA(2,1,2) time series using the command arima(stock,order=c(2,1,2)).

```
Coefficients:
      ar1      ar2     ma1    ma2
      -0.3073 -0.9626 0.3452 0.9783
s.e. 0.0137  0.0178  0.0092 0.0155
sigma^2 estimated as 0.04987: log likelihood = 94.49, aic = -178.98
```

```
n=length(stock)-1;myPars=chart_pars();myPars$cex=1.4;x=seq(100,100+n);dates=index(stock)
arima.s=arima.sim(model=list(ar=c(-.31,-.94),ma=c(.35,.96),order=c(2,1,2)), n.st=100,n)
ar=xts(x=arima.s,order.by=dates);myTh=chart_theme();myTh$col$line.col="blue"
myTh$rylab=F;dev.new(width=12,height=8);par(mfrow=c(1,2));myPars$mar=c(3,2,0,1)
chart_Series(stock,theme=myTh,pars=myPars)
chart_Series(as.vector(stock[1])+ar,theme=myTh,pars=myPars)
```

Listing 2.49: R code - ARIMA(2,1,2) time series generation.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

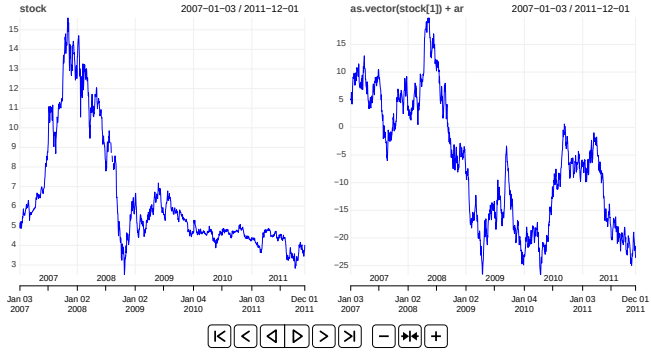


Fig. 2.16: Calibrated ARIMA$(2,1,2)$ samples vs. market prices.*

```
import numpy as np, yfinance as yf,matplotlib.pyplot as plt,pandas as pd;
import mplfinance as mpf,pandas_datareader.data as web
from statsmodels.tsa.arima.model import ARIMA; ar= np.array([1,.3133,.9464])
from statsmodels.tsa.arima_process import ArmaProcess; n= len(stock) - 1;
ma= np.array([1,.3535,.9637]); arma_p= ArmaProcess(ar, ma)
arma_s= arma_p.generate_sample(nsample=n + 100); diffs= arma_s[100:];
arima_s= np.cumsum(diffs); ar_s= pd.Series(arima_s, index=stk_d.index[1:])
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(16,8));ax1.plot(stk_d.index,stk_d.values,c='b',lw=1.);
plt.tight_layout();ax1.set_title('Original Prices');combined_series=stk_d.iloc[0].item()+ar_s
ax2.plot(combined_series.index,combined_series.values,c='b',lw=1.)
ax2.set_title('Simulated ARIMA(2,1,2) Process');plt.show()
```

Listing 2.50: Python code - ARIMA(2,1,2) time series generation.

2.5 Application: Pair Trading

Pair trading data

We consider two assets that can be traded in pairs.

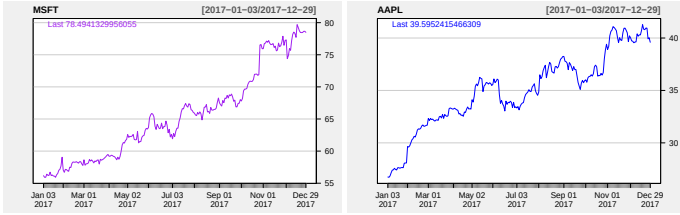
```
install.packages("quantmod"); library(quantmod)
symb= c("1800.HK","KO","PEP"); symb= c("1800.HK","MSFT","AAPL")
getSymbols(symb,from=Sys.Date()-365,to=Sys.Date())
getSymbols(symb,from="2017-01-01",to="2018-01-01")
ClP=lapply(symb, function(x) Ad(get(x)))
stock=do.call(merge, ClP);stock.price=stock[rowSums(is.na(stock[ , 1:3])) == 0, ];
chartSeries(stock.price[,2],up.col="purple",theme="white",name= symb[2])
chartSeries(stock.price[,3],up.col="blue",theme="white",name= symb[3])
```

Listing 2.51: R code - Fetching market data.

* The animation works in Acrobat Reader on the entire pdf file.

```
myPars=chart_pars();myPars$cex=1.4;myTh=chart_theme();myTh$col$line.col="blue"
dev.new(width=12,height=8);par(mfrow=c(1,3))
chart_Series(stock.price[,2],up.col="purple",theme=myTh,name= symb[2],pars=myPars)
chart_Series(stock.price[,3],up.col="blue",theme=myTh,name= symb[3],pars=myPars)
add_TA(stock.price[,2], col='purple', lw=2, on= 1)
```

Listing 2.52: R code - Plotting market data.



(a) MSFT graph. (b) AAPL graph.

Fig. 2.17: MSFT *vs.* AAPL graphs.

```
import yfinance as yf; import pandas as pd; import matplotlib.pyplot as plt
from datetime import datetime, timedelta; symb= ["1800.HK", "MSFT", "AAPL"]
data= yf.download(symb, start="2017-01-01", end="2018-01-01")
close_prices= data['Close']; stock_p= close_prices.dropna(); plt.figure(figsize=(12, 6))
plt.plot(stock_p.index, stock_p[symb[1]], c='m', lw=2, label=symb[1])
plt.title(f"{symb[1]} Price Chart");plt.grid(True);plt.legend();plt.show();plt.figure(figsize=(12,6))
plt.plot(stock_p.index, stock_p[symb[2]], c='b', lw=2, label=symb[2])
plt.title(f"{symb[2]} Price Chart"); plt.grid(True); plt.legend(); plt.show()
```

Listing 2.53: Python code - Fetching market data.

```
stock_p=data['Close'].dropna();fig,(ax1,ax2,ax3)=plt.subplots(1,3,figsize=(16,8))
ax1.plot(stock_p.index,stock_p[symb[1]],c='m',lw=2);ax1.set_title(f"{symb[1]} Price")
ax2.plot(stock_p.index,stock_p[symb[2]],c='b',lw=2);ax2.set_title(f"{symb[2]} Price")
ax3.plot(stock_p.index,stock_p[symb[1]],c='m',lw=2,label=symb[1]);ax1.grid(True)
ax3.plot(stock_p.index,stock_p[symb[2]],c='b',lw=2,label=symb[2]);ax2.grid(True)
ax3.set_title("Both Stocks Overlay");ax3.legend();ax3.grid(True);plt.tight_layout();plt.show()
```

Listing 2.54: Python code - Plotting market data.

Linear regression

As the two assets may evolve within different price ranges, we use a linear regression to put them both on the scale of the second asset.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

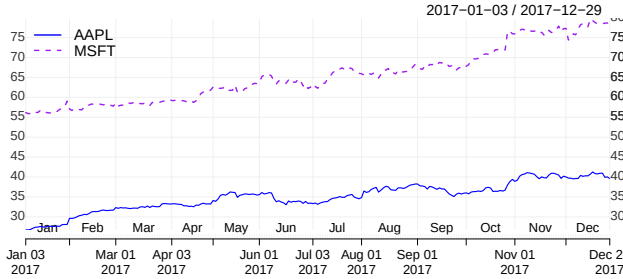


Fig. 2.18: Comparison graph before linear regression.

Letting

$$r_t^{(1)} := \log(S_t^{(1)}) \quad \text{and} \quad r_t^{(2)} := \log(S_t^{(2)}), \qquad t \geqslant 1,$$

by an Ordinary Least Square (OLS) regression we derive a linear relationship of the form

$$r_t^{(2)} = a + b r_t^{(1)} + X_t, \quad t \geqslant 1, \tag{2.25}$$

between $\left(r_k^{(1)}\right)_{k\geqslant 1}$ and $\left(r_k^{(2)}\right)_{k\geqslant 1}$, where X_k is a random remainder term, by minimization of the quadratic residual distance

$$\sum_{t=1}^{n} \left(r_t^{(2)} - a - b r_t^{(1)}\right)^2 \tag{2.26}$$

between $\left(r_t^{(2)}\right)_{t=1,2,\ldots,n}$ and $\left(a + b r_t^{(1)}\right)_{t=1,2,\ldots,n}$, *i.e.*

$$\begin{cases} \widehat{a} = \dfrac{1}{n}\displaystyle\sum_{k=1}^{n}\left(r_k^{(2)} - \widehat{b} r_k^{(1)}\right), \\ \text{and} \\ \widehat{b} = \dfrac{\displaystyle\sum_{k=1}^{n} r_k^{(1)} r_k^{(2)} - \frac{1}{n}\sum_{k,l=1}^{n} r_k^{(1)} \tilde{r}_l^{(2)}}{\displaystyle\sum_{k=1}^{n} (r_k^{(1)})^2 - \frac{1}{n}\sum_{k,l=1}^{n} r_k^{(1)} r_l^{(1)}} = \dfrac{\displaystyle\sum_{k=1}^{n}\left(r_k^{(1)} - \frac{1}{n}\sum_{l=1}^{n} r_l^{(1)}\right)\left(r_k^{(2)} - \frac{1}{n}\sum_{l=1}^{n} \tilde{r}_l^{(2)}\right)}{\displaystyle\sum_{k=1}^{n}\left(r_k^{(1)} - \frac{1}{n}\sum_{k=1}^{n} r_k^{(1)}\right)^2}. \end{cases}$$

see Exercise 2.6. The coefficient a in (2.25) is also called the *premium*.

```
price.pair=stock.price[,2:3]["2017-02-01::"]; reg=lm(log(price.pair[,2]) - log(price.pair[,1]))
b=as.numeric(reg$coef[2]);a=as.numeric(reg$coef[1]);myPars=chart_pars();myPars$cex=1.4
myTh=chart_theme();myTh$col$line.col="blue"; dev.new(width=12,height=8)
chart_Series(price.pair[,2],theme=myTh,pars=myPars, name= symb[3])
add_TA(exp(a+b*log(price.pair[,1])), col='purple', lw=2, on= 1)
```

Listing 2.55: R code - Plotting price pair data.

Fig. 2.19: Comparison graph after linear regression.

```
import numpy as np, matplotlib.pyplot as plt;pr_p=stock_p[symb[1:3]]['2017-2-1':]
x, y=np.log(pr_p[symb[1]]),np.log(pr_p[symb[2]]);b,a=np.polyfit(x, y,1)
plt.figure(figsize=(16, 8));plt.plot(pr_p.index, pr_p[symb[2]], 'b-', lw=2, label=symb[2])
plt.plot(pr_p.index,np.exp(a+b*x),'purple',lw=2,label=f'Fitted (ratio: {b:.3f})')
plt.title(f"{symb[2]} vs Fitted"); plt.grid(True); plt.legend(); plt.tight_layout(); plt.show()
```

Listing 2.56: Python code - Plotting price pair data.

This allows us to define the *spread* $(X_t)_{t\geqslant 1}$ via the linear relationship

$$X_t := \log(S_t^{(2)}) - (a + b\log(S_t^{(1)})), \qquad t \geqslant 0.$$

```
spread<- log(price.pair[,2]) - ( a+ b * log(price.pair[,1]))
dev.new(width=12,height=7); plot(spread,col='blue', main= "Spread",cex.axis=1.3)
```

Listing 2.57: R code - Plotting spread data.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

Fig. 2.20: Spread graph.

```
spread=np.log(pr_p[symb[2]])-a-b*np.log(pr_p[symb[1]]);plt.figure(figsize=(16,7));plt.grid(True)
plt.plot(spread.index,spread,'b-',label='Spread');plt.title("Spread");plt.tight_layout();plt.show()
```

Listing 2.58: Python code - Plotting spread data.

Next, we model the spread X_t using an AR(1) time series and check for its stationarity, in which case the log-price processes $\log\left(S_t^{(1)}\right)_{t\geqslant 0}$ and $\log\left(S_t^{(2)}\right)_{t\geqslant 0}$ are said to be *cointegrated*. This will be interpreted as the existence of a statistically significant connection between $\left(S_t^{(1)}\right)_{t\geqslant 0}$ and $\left(S_t^{(2)}\right)_{t\geqslant 0}$, also named *cointegration*. See Engle and Granger (1987) and Chapter 6 of Enders (2009) for more information on *cointegration*.

Dickey–Fuller test

Consider an AR(1) time series $(X_n)_{n\geqslant 0}$ given by

$$X_n := Z_n + \alpha_1 X_{n-1}.$$

The Dickey–Fuller test allows us to test the null hypothesis H_0, *i.e.* "$|\alpha_1| = 1$", against the alternative stationarity hypothesis "$|\alpha_1| \neq 1$".

```
install.packages('tseries'); library('tseries'); adf.test(spread)
```

Listing 2.59: R code - ADF test.

Its output would lead us to reject the nonstationarity (null) hypothesis H_0 at the confidence level 5% when the p-value is lower than 0.05.

```
        Augmented Dickey–Fuller Test
data: spread
Dickey–Fuller = -2.8771, Lag order = 6, p-value = 0.2077
alternative hypothesis: stationary
```

```
from statsmodels.tsa.stattools import adfuller; adf_result= adfuller(spread)
print(f"ADF Stat.: {adf_result[0]}");print(f"p-value: {adf_result[1]}");print("Critical Values:");
for key, value in adf_result[4].items(): print(f"\t{key}: {value}")
```

Listing 2.60: Python code - ADF test.

Pair trading

The trading signal is $\{-1, 1\}$-valued and determined by the alternating crossing times of a threshold level by the spread.

```
signal=spread;threshold=.02; signal[1]=
 sign(as.numeric(spread[1]));i=1;threshold=-as.numeric(signal[1])*threshold
while (i<length(spread)){i=i+1; while (i<length(spread) &&
 sign(as.numeric(spread[i+1])-threshold)==sign(as.numeric(spread[i])-threshold))
 {signal[i]=sign(as.numeric(spread[i-1])-threshold);i=i+1;}
signal[i]=sign(as.numeric(spread[i-1])-threshold);threshold=-threshold;}
signal[i]=sign(as.numeric(spread[i-1])-threshold);threshold=abs(threshold)
ratio1=range(spread)[1]/threshold;ratio2=range(spread)[2]/threshold
tblue<-rgb(0, 0, 1, alpha=.8);tred<-rgb(1, 0, 0, alpha=.5); dev.new(width=12,height=7)
barplot(spread,col= tblue, lwd= 3, main='', cex.axis=1.4, cex=1.6, las=1); par(new=T);
barplot(-signal,offset=(range(spread)[1]+range(spread)[2])/threshold,ylim=c(ratio2,ratio1),
 xpd=F,col=tred,space= 0,border="blue",xaxt='n',yaxt='n',xlab='',ylab='')
```

Listing 2.61: R code - Pair trading signal.

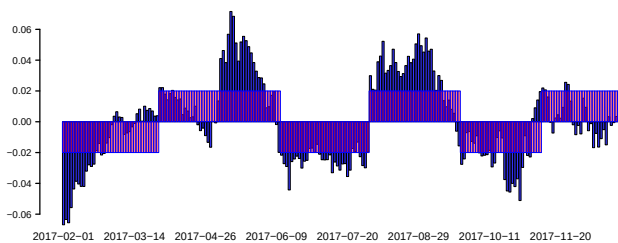


Fig. 2.21: Pair trading signals.

```
spread=np.log(pr__p[symb[2]])-(b*np.log(pr__p[symb[1]])+a);signal=spread.copy()*0
threshold=.02;signal.iloc[0]= np.sign(spread.iloc[0]); current__position= signal.iloc[0]
[ (signal.iloc.___setitem___(i, -1 if signal.iloc[i-1]==1 and spread.iloc[i]<-threshold else 1 if
 signal.iloc[i-1]==-1 and spread.iloc[i]>threshold else signal.iloc[i-1]), signal.iloc.___setitem___(i,
 signal.iloc[i])) for i in range(1, len(spread)) ]; plt.figure(figsize=(16,7))
plt.bar(range(len(spread)),spread.values,color='b',alpha=.8,width=.8,label='Spread')
signal__heights= np.where(signal > 0,.02,-.02)
[plt.bar(i, height, bottom=0, color='r', alpha=.5, width=.4, label='Signal' if i== 0 else "") for
 i, height in enumerate(signal__heights)]
ticks=range(0,len(spread),20);labels=[spread.index[i].strftime('%Y-%m-%d') for i in ticks]
plt.xticks(ticks,labels,rotation=45);plt.ylabel('');plt.title('Spread and Trading Signals')
plt.legend();plt.grid(True,alpha=.3);plt.tight__layout();plt.show()
```

Listing 2.62: Python code - Pair trading signal.

We construct a discrete-time self-financing portfolio strategy $\left(\xi_t^{(1)}, \xi_t^{(2)}\right)_{t\geqslant 1}$ where $\xi_t^{(k)}$ denotes the (possibly fractional) quantity of asset n^o k, $k = 1, 2$, held in the portfolio over the time interval $(t-1, t]$, $t = 1, 2 \ldots, N$. Note that the portfolio allocation

$$\bar{\xi}_t = \left(\xi_t^{(1)}, \xi_t^{(2)}\right)$$

is decided at time $t-1$ and remains constant over the interval $(t-1, t]$ while the stock price changes from $S_{t-1}^{(k)}$ to $S_t^{(k)}$ over this time interval. In other words, the quantity

$$\xi_t^{(k)} S_{t-1}^{(k)}$$

represents the amount invested in asset n^o k at the beginning of the time interval $(t-1, t]$, and

$$\xi_t^{(k)} S_t^{(k)}$$

represents the value of this investment at the end of the time interval $(t-1, t]$, $t = 1, 2, \ldots, N$.

Definition 2.13. *The portfolio prices are given at times* $t = 0, 1, \ldots, N-1$ *by*

$$V_t := \xi_{t+1}^{(1)} S_t^{(1)} + \xi_{t+1}^{(2)} S_t^{(2)}, \quad t = 0, 1, \ldots, N-1.$$

Self-financing portfolio strategies

The opening price of the portfolio at the beginning of the trading period $(t-1, t]$ is

$$\xi_t^{(1)} S_{t-1}^{(1)} + \xi_t^{(2)} S_{t-1}^{(2)}.$$

At the end of the time interval $(t-1, t]$, it takes the closing value

$$\xi_t^{(1)} S_t^{(1)} + \xi_t^{(2)} S_t^{(2)}, \tag{2.27}$$

$t = 1, 2, \ldots, N$. After the new portfolio allocation $\left(\xi_{t+1}^{(1)}, \xi_{t+1}^{(2)}\right)$ is designed we get the new portfolio opening price

$$\xi_{t+1}^{(1)} S_t^{(1)} + \xi_{t+1}^{(2)} S_t^{(2)}, \tag{2.28}$$

at the beginning of the next trading session $(t, t+1]$, $t = 0, 1, \ldots, N-1$. We say that the portfolio strategy $\left(\xi_t^{(1)}, \xi_t^{(2)}\right)_{t=1,2,\ldots,N}$ is *self-financing* when (2.27) coincides with (2.28) for $t = 0, 1, \ldots, N-1$.

Definition 2.14. *A portfolio strategy* $\left(\xi_t^{(1)}, \xi_t^{(2)}\right)_{t=1,2,\ldots,N}$ *is said to be* self-financing *if*

$$\underbrace{\xi_t^{(1)} S_t^{(1)} + \xi_t^{(2)} S_t^{(2)}}_{\text{Closing value}} = \underbrace{\xi_{t+1}^{(1)} S_t^{(2)} + \xi_{t+1}^{(2)} S_t^{(2)}}_{\text{Opening price}}, \qquad t = 1, 2, \ldots, N-1. \quad (2.29)$$

Figure 2.22 is an illustration of the self-financing condition.

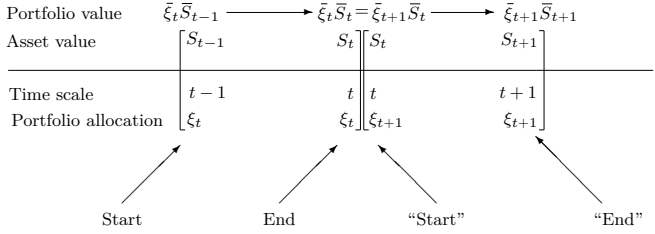


Fig. 2.22: Illustration of the self-financing condition (2.29).

By (2.27) and (2.28) the self-financing condition (2.29) can be rewritten as

$$\xi_t^{(1)} S_t^{(1)} + \xi_t^{(2)} S_t^{(2)} = \xi_{t+1}^{(1)} S_t^{(1)} + \xi_{t+1}^{(2)} S_t^{(2)}, \qquad t = 0, 1, \ldots, N-1,$$

or

$$\left(\xi_{t+1}^{(1)} - \xi_t^{(1)}\right) S_t^{(1)} + \left(\xi_{t+1}^{(2)} - \xi_t^{(2)}\right) S_t^{(2)} = 0, \qquad t = 0, 1, \ldots, N-1.$$

Note that any portfolio strategy $\left(\xi_t^{(1)}, \xi_t^{(2)}\right)_{t=1,2,\ldots,N}$ which is constant over time is self-financing by construction.

Under the self-financing condition (2.29), the portfolio closing values V_t at times $t = 1, 2, \ldots, N$ rewrite as

$$V_t = \bar{\xi}_t \cdot \bar{S}_t = \xi_t^{(1)} S_t^{(1)} + \xi_t^{(2)} S_t^{(2)}, \quad t = 1, 2, \ldots, N. \quad (2.30)$$

Letting

$$\text{hedge.ratio}_t := \frac{S_t^{(2)}}{S_t^{(1)}}, \qquad t = 0, 1, \ldots, N-1,$$

we define the portfolio strategy as

$$\begin{cases} \xi_1^{(1)} := \text{signal}_1 \times \dfrac{\text{hedge.ratio}_0}{1 + \text{hedge.ratio}_0} \\ \xi_1^{(2)} := (-\text{signal}_1) \times \dfrac{1}{1 + \text{hedge.ratio}_0}, \end{cases}$$

and subsequently

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html

$$\begin{cases} \xi_{t+1}^{(1)} := \xi_t^{(1)} + \dfrac{\text{signal}_{t+1} - \text{signal}_t}{2} \times \dfrac{\text{hedge.ratio}_t}{1 + \text{hedge.ratio}_t} \\ \xi_{t+1}^{(2)} := \xi_t^{(2)} - \dfrac{\text{signal}_{t+1} - \text{signal}_t}{2} \times \dfrac{1}{1 + \text{hedge.ratio}_t}, \end{cases}$$

$t = 1, 2, \ldots, N = 1$, so that

$$\xi_1^{(1)} S_0^{(1)} + \xi_1^{(2)} S_0^{(2)} = 0,$$

and

$$\begin{aligned} &\left(\xi_{t+1}^{(1)} - \xi_t^{(1)}\right) S_t^{(1)} + \left(\xi_{t+1}^{(2)} - \xi_t^{(2)}\right) S_t^{(2)} \\ &\quad = \frac{\text{signal}_{t+1} - \text{signal}_t}{2} \times \frac{\text{hedge.ratio}_t}{1 + \text{hedge.ratio}_t} S_t^{(1)} \\ &\qquad - \frac{\text{signal}_{t+1} - \text{signal}_t}{2} \times \frac{1}{1 + \text{hedge.ratio}_t} S_t^{(2)} \\ &\quad = 0, \quad t = 1, 2, \ldots, N, \end{aligned}$$

i.e. $\left(\xi_t^{(1)}, \xi_t^{(2)}\right)_{t=1,2,\ldots,N}$ is self-financing.

Backtesting

The performance of the pair trading algorithm can be estimated by the following code.

```
hdg.rtio=price.pair[,1]/price.pair[,2]; df.xi1=diff(signal)[-1]/(1+hdg.rtio)/2
df.xi2=-diff(signal)[-1]*hdg.rtio/(1+hdg.rtio)/2;xi1=cumsum(c(signal[1]/(1+hdg.rtio[1]),df.xi1))
xi2=cumsum(c(-signal[1]*hdg.rtio[1]/(1+hdg.rtio[1]),df.xi2))
dev.new(width=12,height=7); portfolio=xi1*price.pair[,1]+xi2*price.pair[,2]
benchmark=as.numeric(xi1[1])*price.pair[,1]+as.numeric(xi2[1])*price.pair[,2]
plot(benchmark,col='orange',main= '', lwd= 3, cex.axis=1, cex=1, las=1)
lines(xi1,col='purple',main= "Xi1", lwd= 3, cex.axis=1, cex=2)
lines(xi2,col='blue',main= "Xi2",lwd= 3, cex.axis=1, cex=2)
lines(portfolio,col='red',main= "Portfolio performance" ,lwd= 4, cex.axis=1,cex=2,las=1)
legend("topleft", legend=c("Pair trading", "Benchmark", "Xi1", "Xi2"), col=c("red", "orange",
  "purple", "blue"), lty=1:2, cex=1.5,xpd=T)
```

Listing 2.63: R code - Portfolio computation.

```
import numpy as np, matplotlib.pyplot as plt
h_r= pr_p[symb[1]] / pr_p[symb[2]]; df_sgn= np.diff(signal)
df_xi1=df_sgn/(1+h_r.iloc[1:])/2;df_xi2=-df_sgn*h_r.iloc[1:]/(1+h_r.iloc[1:])/ 2
xi1= np.cumsum(np.concatenate([[signal.iloc[0] / (1 + h_r.iloc[0])], df_xi1]))
xi2= np.cumsum(np.concatenate([[-signal.iloc[0] * h_r.iloc[0] / (1 + h_r.iloc[0])], df_xi2]))
portfolio= xi1 * pr_p[symb[1]].values + xi2 * pr_p[symb[2]].values
benchmark=xi1[0]*pr_p[symb[1]]+xi2[0]*pr_p[symb[2]]; plt.figure(figsize=(16, 7))
plt.plot(benchmark.index, benchmark.values, c='orange', lw=3, label='Benchmark')
plt.plot(pr_p.index,xi1,c='m',lw=3,label='Xi1')
plt.plot(pr_p.index,xi2,c='b',lw=3,label='Xi2');plt.tight_layout(pad=0)
plt.grid(True);plt.plot(pr_p.index,portfolio,c='r',lw=4,label='Pair trading')
plt.legend(loc='upper left');plt.xlim(pr_p.index[0],pr_p.index[-1]);plt.show()
```

Listing 2.64: Python code - Portfolio computation.

The performance of the pair trading portfolio return is plotted in Figure 2.23.

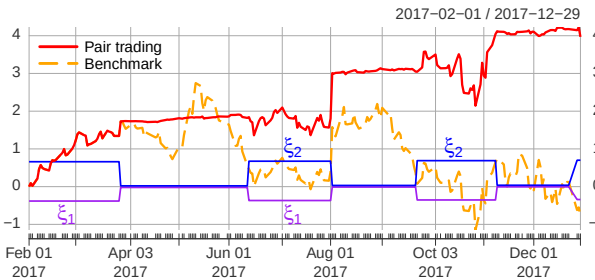


Fig. 2.23: Pair trading performance.

The portfolio strategy used as a benchmark is built by investing a constant allocation in Stock 1 and Stock 2, and its return from time 0 to time n, given by

$$B_t := \xi_1^{(1)} S_t^{(1)} + \xi_1^{(2)} S_t^{(2)}, \quad t = 1, \ldots, N,$$

is compared to the pair trading portfolio performance. The ADF test allows us to reject the non-stationarity hypothesis at the level $p = 8.5\%$.

```
> source("pairtrading.R")
Examples of pairs: 005930.KS vs AAPL, 2600.HK vs 1919.HK
Enter Stock 1 (Ex: GOOG):1800.HK
Enter Stock 2 (Ex: AAPL):1919.HK

        Augmented Dickey–Fuller Test
data: spread
Dickey–Fuller = -3.2232, Lag order = 6, p-value = 0.08465
alternative hypothesis: stationary
```

Figure 2.24 presents another example of pair trading backtesting.*

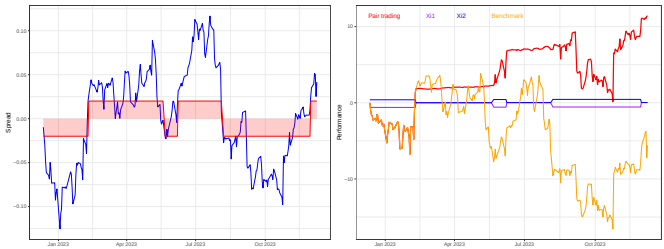


Fig. 2.24: Pair trading performance.

See also the R PairTrading package in Takayanagi and Ishikawa (2017).

Exercises

Exercise 2.1 Consider the MA(1) time series $(X_n)_{n\in\mathbb{Z}}$ defined as

$$X_n := Z_n + aZ_{n-1}, \quad n \in \mathbb{Z},$$

where $(Z_n)_{nin\mathbb{Z}}$ is a white noise sequence and $a \in \mathbb{R}$.

a) Compute the autocovariance function

$$\rho(k) = \mathrm{Cov}(X_n, X_{n+k})$$

for all $k \in \mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$, and plot it on the graph below when $a = 2$.

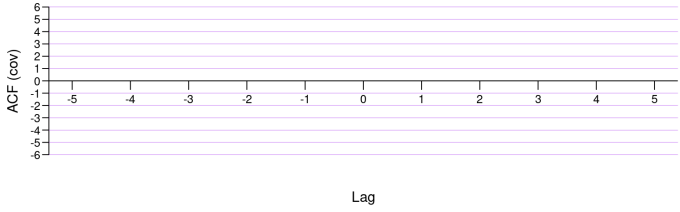


Fig. 2.25: Autocorrelation graph.

b) Is the time series $(X_n)_{n\in\mathbb{Z}}$ weakly stationary? Strictly stationary?

* Download the corresponding R code.

Exercise 2.2 Let $(Z_n)_{n\geqslant 1}$ denote a discrete-time white noise.

a) Check the (weak) stationarity of the AR(1) time series $(X_n)_{n\geqslant 1}$ given by

$$X_n = Z_n + X_{n-1}, \qquad n \geqslant 1.$$

b) Check the (weak) stationarity of the AR(2) time series $(Y_n)_{n\geqslant 1}$ given by

$$Y_n = Z_n + \frac{3}{4} \times Y_{n-1} - \frac{1}{8} \times Y_{n-2}, \qquad n \geqslant 2.$$

Hint: Consider the roots of $\varphi(z) = 1$ where $\varphi(z)$ is the polynomial defined by $X_n = Z_n + \varphi(L)X_n$ and L is the lag operator $LX_n = X_{n-1}$.

Exercise 2.3 Let $\alpha \in \mathbb{R}$. Consider an *i.i.d.* white noise sequence $(Z_n)_{n\geqslant 0}$ with mean $\mathbb{E}[Z_n] = 0$ and variance $\mathrm{Var}[Z_n] = 1$, $n \geqslant 1$, and the AR(1) time series $(X_n)_{n\geqslant 0}$ given by $X_0 := 0$ and

$$X_n := Z_n + \alpha X_{n-1}, \qquad n \geqslant 1. \tag{2.31}$$

a) Find a recurrence relation for the mean $\mathbb{E}[X_n]$ in the parameter $n \geqslant 0$, and deduce the value of $\mathbb{E}[X_n]$ for all $n \geqslant 0$.

b) For fixed $n \geqslant 1$, find a recurrence relation in the parameter $k \geqslant 0$ for the covariance

$$\mathrm{Cov}(X_{n+k}, X_n) = \mathbb{E}[X_{n+k}X_n] - \mathbb{E}[X_{n+k}]\mathbb{E}[X_n], \qquad k \geqslant 0.$$

c) Find a recurrence relation in the parameter $n \geqslant 1$ for the variance

$$\mathrm{Var}[X_n] = \mathbb{E}[X_n^2] - (\mathbb{E}[X_n])^2, \qquad n \geqslant 0.$$

d) When is the time series $(X_n)_{n\geqslant 0}$ weakly stationary?

Exercise 2.4 Consider an *i.i.d.* white noise sequence $(Z_n)_{n\geqslant 0}$ with mean $\mathbb{E}[Z_n] = 0$ and variance $\mathrm{Var}[Z_n] = 1$, $n \geqslant 1$, and the AR(3) time series $(X_n)_{n\geqslant 3}$ given by

$$X_n := Z_{n-1} - Z_{n-2} + \alpha Z_{n-3}, \qquad n \geqslant 3.$$

a) Find the autocovariances

$$\begin{cases} \mathrm{Cov}(X_n, X_n) = \mathrm{Var}[X_n], \\ \mathrm{Cov}(X_{n+1}, X_n), \\ \mathrm{Cov}(X_{n+2}, X_n), \\ \mathrm{Cov}(X_{n+k}, X_n), \qquad k \geqslant 3. \end{cases}$$

b) Show that $(X_n)_{n\geqslant 3}$ has same distribution as an MA(q) time series $(Y_n)_{n\geqslant 3}$ of the form

$$Y_n = Z_n + \sum_{k=1}^{q} \beta_k Z_{n-k},$$

whose order q and coefficients $(\beta_k)_{1\leqslant k\leqslant q}$ will be determined.

Exercise 2.5 Consider an AR(1) time series $(X_n)_{n\geqslant 0}$ given by $X_0 = 0$ and

$$X_n := Z_n + \alpha_1 X_{n-1}, \qquad n \geqslant 1,$$

and the difference operator

$$\nabla X_n := X_n - X_{n-1}, \qquad n \geqslant 1,$$

also written $\nabla = I - L$, which can be integrated by the telescoping identity

$$X_n = X_0 + \sum_{k=1}^{n} (X_k - X_{k-1}) = \sum_{k=1}^{n} \nabla X_k, \quad n \geqslant 0.$$

a) Show that the first-order difference process $(\nabla X_n)_{n\geqslant 1} = (X_n - X_{n-1})_{n\geqslant 1}$. forms an ARMA$(1,2)$ time series.

b) Show that the second-order difference process

$$(\nabla^2 X_n)_{n\geqslant 2} = (\nabla X_n - \nabla X_{n-1})_{n\geqslant 2}$$

forms an ARMA$(1,3)$ time series.

Exercise 2.6 Consider two sequences $\left(r_k^{(1)}\right)_{k\geqslant 1}$ and $\left(r_k^{(2)}\right)_{k\geqslant 1}$ of market returns. We aim at deriving a linear relationship of the form

$$r_k^{(2)} = a + b r_k^{(1)} + X_k, \quad k \geqslant 0,$$

between $\left(r_k^{(1)}\right)_{k\geqslant 1}$ and $\left(r_k^{(2)}\right)_{k\geqslant 1}$, where X_k is a random remainder term, by minimization of the quadratic residual distance

$$\sum_{k=1}^{n} \left(r_k^{(2)} - a - br_k^{(1)}\right)^2 \tag{2.32}$$

between $\left(r_k^{(2)}\right)_{k=1,2,\ldots,n}$ and $\left(a + br_k^{(1)}\right)_{k=1,2,\ldots,n}$.

a) Compute the partial derivatives of (2.32) with respect to the parameters a and b.
b) By equating the derivatives to zero, find the least square estimates $\widehat{a}$ and $\widehat{b}$ of the parameters a and b based on the sequences $\left(r_k^{(1)}\right)_{k\geqslant 1}$ and $\left(r_k^{(2)}\right)_{k\geqslant 1}$.

Exercise 2.7 Consider the following ADF test output on a time series:

```
        Augmented Dickey–Fuller Test
data: series
Dickey–Fuller = -2.8771, Lag order = 6, p-value = 0.02377
alternative hypothesis: stationary
```

Does this test result allow us to reject the nonstationarity (null) hypothesis H_0 at a 5% confidence level?

Exercise 2.8 Let $(Z_n)_{n\in\mathbb{Z}}$ denote a white noise sequence with zero mean and variance σ^2, and consider the AR(2) time series $(X_n)_{n\in\mathbb{Z}}$ given by

$$X_n = Z_n + \alpha_1 X_{n-1} + \alpha_2 X_{n-2}, \qquad n \in \mathbb{Z}.$$

a) Assume that $\alpha_1 := a$ and $\alpha_2 := 2a^2$ for some $a \geqslant 0$. For which values of the parameter a is the time series $(X_n)_{n\in\mathbb{Z}}$ stationary?
Hint: Consider the solutions of the equation $\varphi(z) = 1$, where $\varphi(z)$ is the polynomial defined by $X_n = Z_n + \varphi(L)X_n$ and L is the lag operator defined by $LX_n = X_{n-1}$.
In the sequel, we assume that the time series $(X_n)_{n\in\mathbb{Z}}$ is stationary.
b) Show that $\mathbb{E}[X_n] = 0$, $n \in \mathbb{Z}$, if $\alpha_1 + \alpha_2 \neq 1$.
c) We assume that $(X_n)_{n\in\mathbb{Z}}$ is *causal*, *i.e.* for any $n \in \mathbb{Z}$, X_n depends only on $(Z_k)_{k<n}$. Show that $\mathrm{Cov}(X_n, Z_n) = \sigma^2$ for all $n \in \mathbb{Z}$.
d) Taking $\alpha_1 := 1/4$ and $\alpha_2 := 1/2$, compute the autocovariance $\mathrm{Cov}(X_{n+1}, X_n)$ given that $\mathrm{Cov}(X_n, X_n) = 16$, $n \in \mathbb{Z}$.

This version: April 28, 2026
https://personal.ntu.edu.sg/nprivault/indext.html