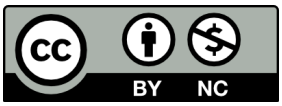
Business 4720

Time Series Analysis

Joerg Evermann



Learning Goals

After reading this chapter, you should be able to:

- Explain different statistical models for time series data and create time series for each model.
- Perform basic operations on time series data.
- Perform time series smoothing using different smoothing methods.
- Understand the importance of stationarity for time series analysis and how to assess time series data for stationarity.
- Address non-stationary time series data using differencing and detrending.
- Explain the different components of an ARIMA model and fit an ARIMA model to time series data and evaluate its fit.
- Use the ACF and PACF to select the appropriate ARIMA model.
- Explain GARCH models and fit a GARCH model to time series data.

Sources and Further Reading

The material in this chapter is based on the following sources. Consult them for additional information and details.

Resources

Robert H. Shumway and David S. Stoffer (2017) *Time Series Analysis and Its Applications*, 4th Edition. Springer.

`https://www.stat.pitt.edu/stoffer/tsa4/`

The book by Shumway and Stoffer provides a very comprehensive but also somewhat technical introduction to the subject of time series analysis. The authors have also published the `astsa` library for R to accompany their book. This library provides a number of data sets and functions for time series analysis.

Resources

Rob J. Hyndman and George Athanasopoulos (2018) *Forecasting: Principles and Practice*, 2nd edition. OTexts.

`https://otexts.com/fpp2/`

The book by Hyndman and Athanasopoulos is somewhat less technical in nature than the book by Shumway and Stoffer and also provides R code. The coverage of the two

books also differs somewhat, but it this more accessible than Shumway and Stoffer for undergraduate students.

Resources

In addition to these books, there are a number of very useful tutorials available on the internet that can augment or summarize the material in the books. They are less focused on theory and more focused on actually performing time series analysis:

- https://github.com/nickpoison/tsa4
- https://a-little-book-of-r-for-time-series.readthedocs.io/en/latest/src/timeseries.html
- https://rc2e.com/timeseriesanalysis
- https://atsa-es.github.io/atsa-labs/chap-tslab.html

1 Introduction

This section provides an introduction to time series analysis. Time series analysis is a complex topic with a multitude of different methods and techniques and this section can provide only a glimpse at the basic ideas and concepts.

Time series analysis is a set of statistical techniques that involve analyzing time-ordered data points or observations to extract meaningful statistics and other characteristics. This type of analysis is important across various fields such as economics, where it may be used to model unemployment rates, finance, where it may be used to model stock prices, social science, where it may be used to model high school graduation rates, natural sciences, where it may be used to model weather and climate trends, ecology, where it could be used to model animal population numbers, or epidemiology where it may be used to model the spread of epidemics. Understanding trends, cycles, and patterns over time can lead to useful insights and informed decision-making. Figure 1 shows an example of a basic time series of the quarterly earnings per share of a company.

At its core, a time series is a sequence of data points recorded at successive time intervals. The data is typically collected at uniform intervals – be it hourly, daily, monthly, or yearly. Time series analysis helps in understanding the inherent structure and functions that generate the series. It aims to model the underlying context of the data, whether to understand the past behavior or to forecast future values.

The analysis of time series can be divided broadly into two types: descriptive and inferential. Descriptive analysis focuses on visualizing and summarizing the main features of the data, such as trends (long-term direction), seasonality (regular pattern of fluctuation within a year), and irregular components (unpredictable, random fluctuations). Inferential analysis, on the other hand, involves using models to predict future values

based on known past values, testing hypotheses, and deriving estimates of population parameters.

In time series analysis, two fundamental approaches to examining data are the time-domain and the frequency-domain approaches. The *time-domain approach* analyzes data as it evolves over time, focusing on the relationship between current and past values to predict future values. This approach is primarily concerned with understanding and modeling the temporal sequence directly in the time dimension. This approach is particularly useful for forecasting, where understanding how values are correlated through time is essential. It provides direct and often simple models that are interpretable in terms of the original time series data.

The *frequency-domain approach*, on the other hand, analyzes data based on the rate at which the data's features repeat over time. This approach transforms the time series data into the frequency domain using mathematical transformations (the most common being the Fourier Transform). It decomposes the time series into a combination of sinusoid functions with different frequencies and amplitudes. The frequency-domain approach is useful for identifying hidden periodicities or cyclical behaviors in the data, which may not be apparent in the time domain.

2 Time Series Statistical Models

Time series statistical models are essential tools used to analyze and forecast time-dependent data. Four common models are the moving average (MA) model, the autoregressive (AR) model, the random walk with drift, and the signal in noise model. Each model has different characteristics and applications, suited to different types of time series data.

Moving Average Model

The *Moving Average* model is a fundamental time series model that expresses the current value of the time series as a function of past errors or deviations, with the as-

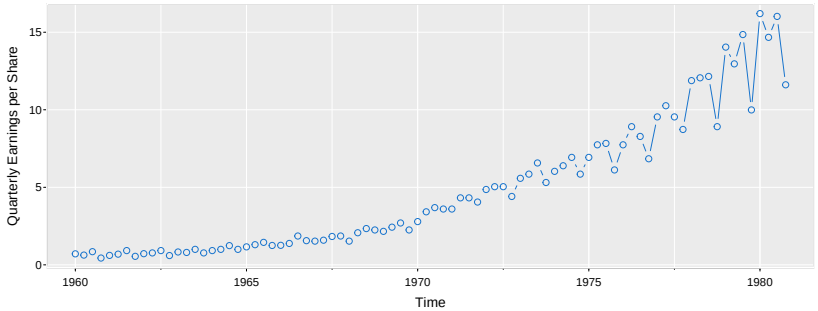


Figure 1: Example of time series data

sumption that these errors are white noise, that is, random. An example model is given by:

$$v_t = \frac{1}{3}(w_{t-1} + w_t + w_{t+1})$$

where w_t are the white noise error terms. The sum of the weights in a moving average model must be 1. In this example, all white noise terms are weighted equally by $1/3$.

MA models are particularly useful in smoothing out noise and forecasting when the series exhibits a random behavior with no trend or seasonality.

The following R code block uses the `filter` function to generate the example model. The filter operates on the white noise, extending to both sides of the current time step, and creates a weighted sum of the closest three value in `w`, specified by the `mode='convolution'` argument. The resulting plots are shown in Figure 2.

R

```r
# Load ASTSA library
library(astsa)

# Random numbers as errors
w <- rnorm(500,0,1)

# Moving average
v <- filter(w, sides=2, filter=c(1/3,1/3,1/3), method='convolution')

# Plot timeseries
par(mfrow=c(2,1))
tsplot(w, main="white noise", col=3, gg=T)
tsplot(v, ylim=c(-3,3), main="moving average", col=4, gg=T)
```

Autoregressive Model

The *Autoregressive (AR) model* is based on the concept that current values of a series can be forecasted from previous values. An example model is:

$$x_t = x_{t-1} - 0.9x_{t-2} + w_t$$

where w_t is white noise. For stability of the time series, all coefficient of x are usually less than or equal to 1.

AR models are widely used in economic and financial time series where data points are influenced significantly by their previous values.

The following R code uses the `filter` function in "recursive" mode with parameters `1` and `-.9` to create the time series corresponding to the example model[1]. The resulting plot is shown in Figure 3.

[1]The R code for this and following examples are based on material Shumway & Stoffer

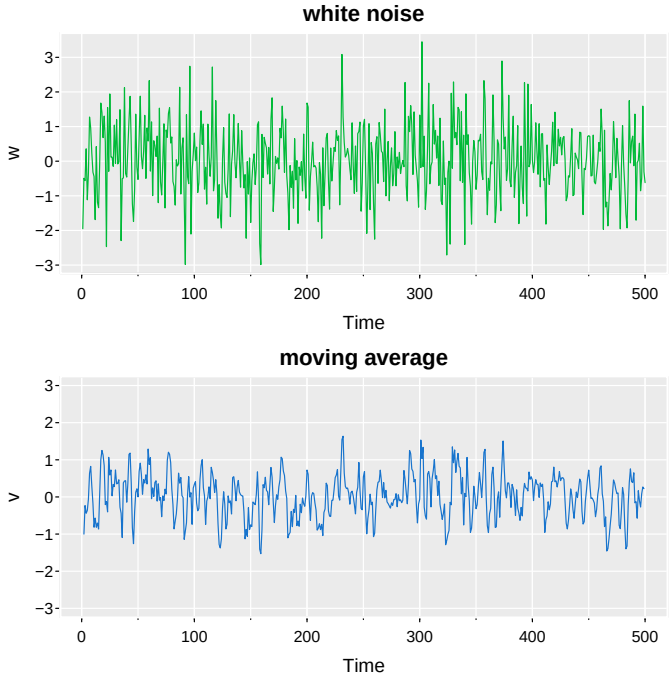


Figure 2: Example white noise time series and its moving average

```R
# Random numbers (errors)
w = rnorm(550,0,1)

# Autoregressive time series
x = filter(w, filter=c(1,-.9), method="recursive")

# Remove first 50 values for startup
x = x[-(1:50)]

# Plot time series
tsplot(x, main="autoregression", col=4)
```

Random Walk with Drift

A *random walk with drift* adds a constant to the standard random walk, allowing the series to drift upwards or downwards over time. An example model is given by:

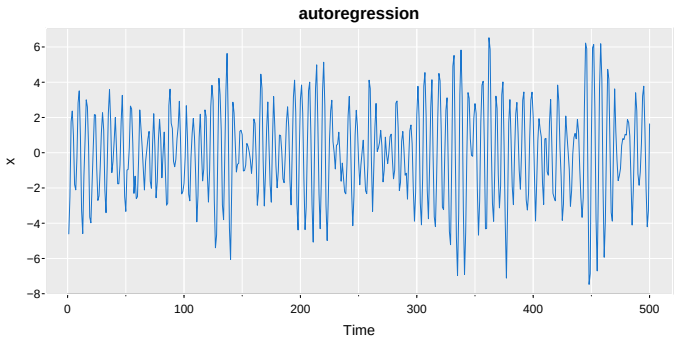


Figure 3: Example autoregressive time series

$$
\begin{aligned}
x_t &= \delta + x_{t-1} + w_t \\
&= \delta t + \sum_{j=1}^{t} w_j
\end{aligned}
$$

where δ represents the drift (constant term), and w_t is the noise component. Importantly, x_{t-1} is the only lagged x term and its coefficient must be 1.

This model is commonly applied in financial markets to model stock prices or other investments, reflecting that prices are serially correlated and can trend over time.

The following R code block uses the the `cumsum()` function to calculate the cumulative sum. The resulting time series are shown in Figure 4 and show the random walk and the drift component that is added to it.

R

```r
# White noise
w = rnorm(200)
# Random walk
x = cumsum(w)

# Drift
drift = .2
w.drift = w + drift;
# Random walk with drift
x.drift = cumsum(w.drift)

# Plot
tsplot(x.drift, main="random walk", col=3, ylab='', ylim=c(-10,55))
abline(a=0, b=drift, lty=2, col=3)
lines(x, col=4)
abline(a=0, b=0, lty=2, col=4)
```

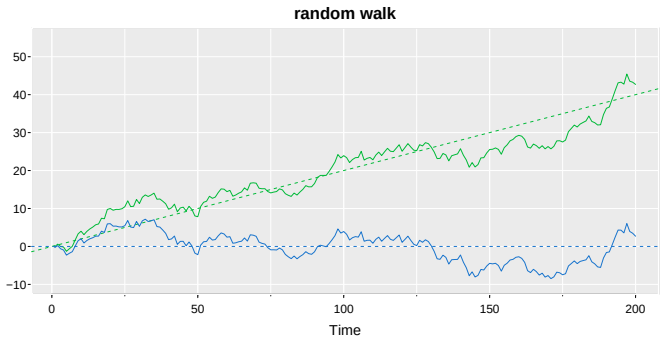


Figure 4: Example random walk with drift time series

Signal in Noise Model

The *Signal in noise* model views the time series as a combination of a true signal and random noise. An example model with a sinusoidal signal characterized by its amplitude, frequency and phase shift is:

$$x_t = A\cos(2\pi\omega t + \phi)$$

for example,

$A = 2$	amplitude
$\omega = 1/50$	frequency
$\phi = .6\pi$	phase shift

This model is fundamental in signal processing and is used to understand underlying trends in the presence of noisy observations. Techniques like filtering and smoothing are often applied to extract the signal from x_t.

The following R code block creates a sinusoidal signal and overlays it with different amounts of white (Gaussian) noise. The resulting time series are shown in Figure 5.

R

```r
# Create signal
cs = 2*cos(2*pi*1:500/50 + .6*pi)

# Create noise
w = rnorm(500,0,1)

# Overlay with gaussian noise and plot
par(mfrow=c(3,1), mar=c(3,2,2,1), cex.main=1.5)
tsplot(cs, main='Signal', col=2, gg=T)
tsplot(cs+w, main='Signal and N(0,1) noise', col=3, gg=T)
tsplot(cs+5*w, main='Signal and N(0,25) noise', col=4, gg=T)
```

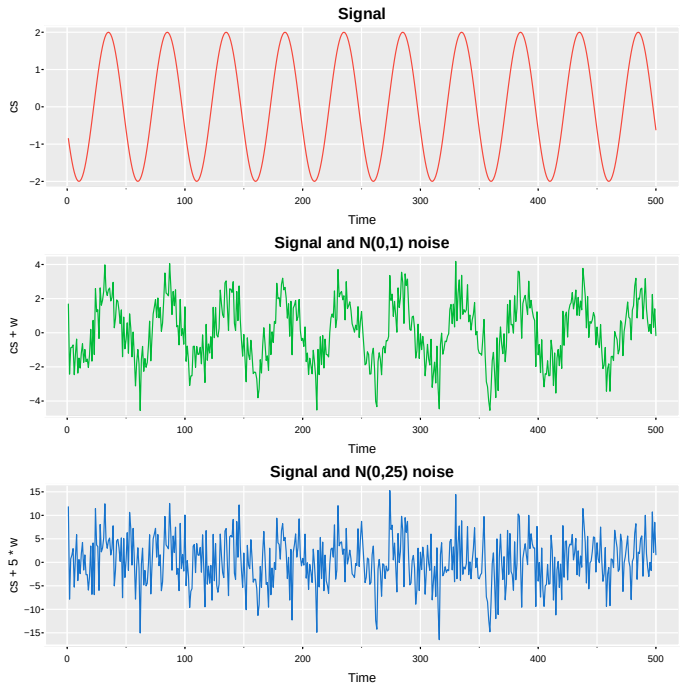


Figure 5: Example signal in noise time series

3 Basic Time Series Operations in R

A time series can be constructed from an ordinary data set using the `ts()` function in R and supplying a start time stamp and a sampling frequency. For example, the following R code creates a time series of montly observations, beginning in January 2020 with the values 1 through 24. R will try to sensibly interpret the `start` and `frequency` arguments: Frequencies of 4 are interpreted as quarters of the year, 7 is interpreted as days of a week, 12 is interpreted as months of the year.

```R
# Creating a time series object with monthly data
ts_data <- ts(1:24, frequency = 12, start = c(2020, 1))
```

To see the first and last last observations of a time series, use the `head()` and `tail()` functions:

```R
head(ts_data)
tail(ts_data)
```

Missing values in a time series cannot be imputed in the usual manner because the

observations are not independent of each other. Two simple ways of "filling in" missing data are either to simply carry the last observation forward, which assumes there are negligible changes in the value over time, or to interpolate the missing values. In linear interpolation, a line is imagined between the last observation before a series of missing values and the first observation after such a series of missing values. Missing values are then assumed to be on that line. This assumes that the time series is approximately linear, at least for short gaps.

The `zoo` library for R contains the functions `na.locf()` and `na.approx()` that implement these methods of handling missing values. Missing values at the beginning or end of a time series can be removed with the `na.trim()` function. The following R code block illustrates the use of all three functions:

R

```r
# Load zoo library
library(zoo)

# Introduce NA values into the time series
ts_data[c(5, 10, 15)] <- NA

# Using na.trim to remove leading/trailing NA values
trimmed_ts <- na.trim(ts_data)

# Using na.locf (Last Observation Carried Forward) to handle NA values
locf_ts <- na.locf(ts_data)

# Using na.approx to interpolate NA values
approx_ts <- na.approx(ts_data)
```

Two time series can be combined using the `ts.intersect()` or `ts.union()` functions. The former function combines the series only for overlapping, that is, intersecting, times, possibly cutting off the head or tail of one or the other series. The latter function retains all dates of both series and "pads" the head or tail of one or the other series with "NA" values.

R

```r
# Creating another time series
ts_data2 <- ts(c(1:24), frequency = 12, start = c(2020, 7))

# Using ts.intersect to determine intersection of two time series
intersect_ts <- ts.intersect(ts_data, ts_data2)

# Using ts.union to determin union of two time series
union_ts <- ts.union(ts_data, ts_data2)
```

An important operation in time series analysis is to "lag" a time series, that is, to shift it forwards or backwards in time. R provides the `lag()` function for this purpose. A positive argument shifts the time series *backwards* by the specified number of time periods, while a negative argument shifts it forward:

```R
# Positive k shifts backwards in time
lag_ts <- lag(ts_data, 2)

# Negative k shifts forwards in time
lag_ts <- lag(ts_data, 3)
```

4 Smoothing a Time Series

Time series smoothing is a technique used to remove noise and reveal signals or underlying trends in the data. Four commonly used methods for time series smoothing are moving average, kernel smoothing, lowess regression, and smoothing splines.

Moving Average Smoothing

Moving average smoothing is one of the simplest and most widely used methods for smoothing time series data. It involves calculating the weighted mean of the consecutive data points within a specified window that moves along with the data:

$$m_t = \sum_{j=-k}^{k} a_j x_{t-j} \quad \text{where} \quad \sum_{j=-k}^{k} a_j = 1$$

where a are the weights that sum to one. A simple filter uses uniform weights, but other shapes are possible.

This model can be implemented using the `filter` function in R. The filter in the example below is two-sided and is centered on the current time stamp, that is, it uses data before and after the current time point. When the `sides=1` argument is used, the filter is over past values only. The `filter` argument specifies the weights for the moving average. The results for an example time series are shown in Figure 6.

```R
# Use the soi dataset from the astsa library as an example
library(astsa)
?soi
# Apply moving average filter
f = 1/12 * c(0.5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.5)
filter(soi, sides=2, filter=f)
```

Kernel Smoothing or Kernel Regression

Instead of a filter with simple weights as in moving average smoothing, *kernel smoothing* uses a weighted average of neighbouring points where the weights are determined by a function known as the *kernel*. A common choice is a Gaussian kernel that uses

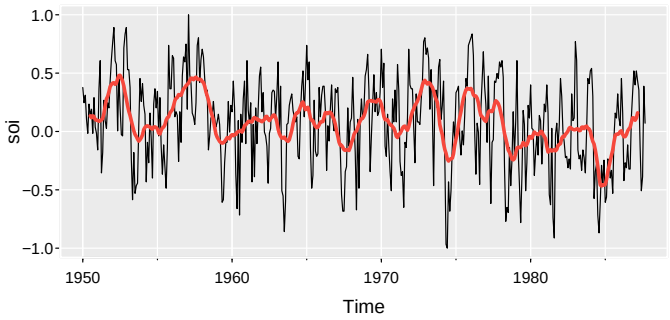


Figure 6: Moving average smoothing

the normal distribution density function, which produces a weighted average with a bell-shaped curve of weights around each data point. The weights for averaging the time series values are determined as follows:

$$a_i(t) = \frac{K\left(\frac{t-i}{b}\right)}{\sum_{j=1}^{n} K\left(\frac{t-j}{b}\right)}$$

where K is the Gaussian kernel:

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-z^2/2\right)$$

Here, b is the "bandwidth" of the kernel, that determines the shape of the kernel function, that is, how "wide" or "broad" it is.

The smoothed time series s_t is then given by:

$$\begin{aligned} s_t &= \sum_{i=1}^{n} a_i(t) x_t \\ &= \frac{\sum_{i=1}^{n} K\left(\frac{t-i}{b}\right) x_t}{\sum_{j=1}^{n} K\left(\frac{t-j}{b}\right)} \end{aligned}$$

The R function `ksmooth()` with the `normal` kernel argument provides exponential smoothing. The `bandwidth` parameter determines the "width" of the kernel by specifying the distance from the quartiles to the mean of the normal distribution function. Example smoothing results for various bandwidth values are shown in Figure 7.

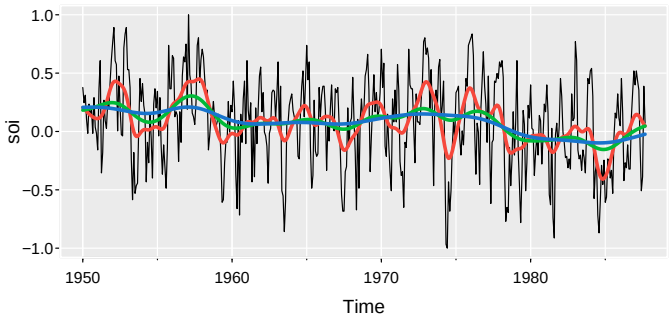


Figure 7: Kernel density smoothing with different kernel bandwidths

```R
# Apply gaussian kernel smoothing
ksmooth(time(soi), soi, kernel='normal', bandwidth=1)
```

Lowess Regression

Lowess Regression (locally weighted scatterplot smoothing) combines a multiple regression model with a k-nearest-neighbour-based model. Each point on the smoothed time series is estimated by a weighted least squares regression over a local neighbourhood of f observations that are closest in time to the target point. The weights decrease with distance from the target observation, thereby providing robustness against outliers and yielding a smooth curve that closely follows the data.

The R function `lowess()` uses the `f` parameter for specifying the proportion of observations used for the regressions. An example result is shown in Figure 8.

```R
# Apply lowess smoothing
lowess(soi, f=0.1)
```

Smoothing Splines

Smoothing splines are a method that fits a smooth, flexible "spline" function to the data. Smoothing splines balance the fit of the spline to the data against the smoothness of the spline curve and are essentially penalized polynomial regression models that fit the model:

$$m_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3$$

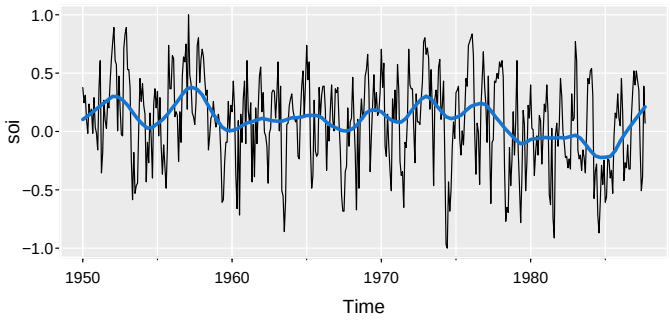


Figure 8: Lowess regression smoothing example

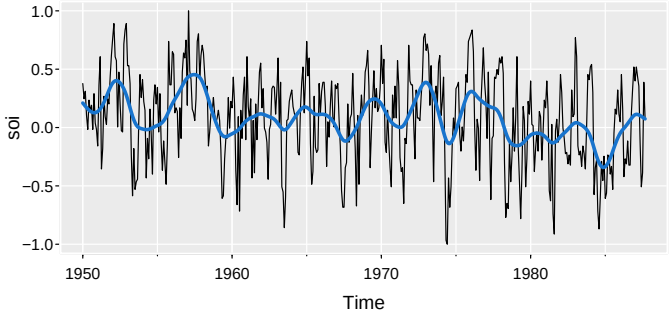


Figure 9: Smoothing spline example

by minimizing the loss function

$$\sum_{t=1}^{n}(x_t - m_t)^2 + \lambda \int \left(\frac{\mathrm{d}^2 m}{\mathrm{d}t^2}\right)^2 dt$$

The `smooth.spline()` function uses cubic splines, that is polynomials of degree 3. The smoothing parameter `spar` controls the regression penalty λ in the equation above and thereby the degree of smoothing. The result for this example is shown in Figure 9.

R

```
# Apply smoothing splines
smooth.spline(time(soi), soi, spar=0.5)
```

Hands-On Exercise

1. Generate 100 observations from the autoregression model $x_t = -.9x_{t-2} + w_t$ with $\sigma_w^2 = 1$
   (a) Smooth the time series using a moving average filter $v_t = (x_t + x_{t-1} + x_{t-2} + x_{t-3})/4$, plot x_t as a line and superimpose v_t. Comment on the behaviour of x_t and how applying the moving average filter changes that behavior
   (b) Smooth the time series using kernel smoothing, produce plots as above, and experiment with different kernel bandwidths. Comment on the behvaiour of the smoothed series.
   (c) Smooth the time series using Lowess, produce plots as above, and experiment with different values for the fraction of observations to include in each regression. How does the smoothed series change as you vary that fraction?
   (d) Smooth the time series using smoothing splines, produce plots as above, and experiment with different values for the smoothing parameter that controls the regression penalty. How does the smoothed series change as you vary that parameter?
2. Generate 100 observations from the sinusoidal series $x_t = \cos(2\pi t/4)$ and add $N(0, 1)$ noise. Repeat the four smoothing exercises. Compare and contrast the results of these exercises. Which smoothing is more appropriate for which type of time series data?

5 Time Series Regression

Time series regression refers to using time series data in ordinary least squares regression. The focus is not necessarily on modeling data series over time or describing the future values of a time series as a function of earlier values, although lagged time series can certainly be used in time series regression. Instead, time series regression predicts the value of one time series from one or more other time series.

Consider an epidemiological example with three weekly time series, one expressing cardiovascular mortality (”cmort”, the likelihood of dying of heart attack), another describing ambient temperature (”tempr”) and a third one describing air pollution (”part”). Figure 10 shows these three time series superimposed in one graph. Visual inspection of the graph suggests that mortality may be correlated with temperature and air pollution.

Time series regression uses ordinary least squares (OLS) regression models where each series is a predictor variable. It essentially neglects the time aspect of the time series data. The following R code block shows examples of time series regression to predict or explain mortality. Note the use of the `time()` function extract the timestamps from

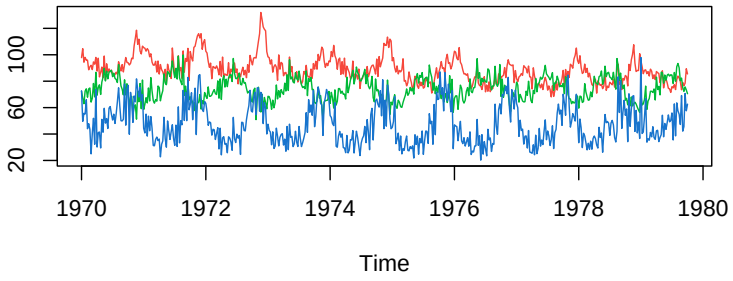


Figure 10: Three time series

the time series data.

```R
# Use data from the astsa library
library(astsa)

# Plot the three time series
ts.plot(cmort, tempr, part, col=2:4)

# Center the temperature variable
temp = tempr - mean(tempr)

# Square the temperature variable
temp.2 = temp^2

# Fit different linear models and provide summaries
summary(lm(cmort ~ time(cmort)))
summary(lm(cmort ~ time(cmort) + temp))
summary(lm(cmort ~ time(cmort) + temp + temp.2))
summary(lm(cmort ~ time(cmort) + temp + temp.2 + part))
```

Time series regression can also use lagged time series data, that is, data of the same series that is shifted backward in time. This is somewhat similar to the autoregressive models defined below. The following R code block lags the temeratur by two weeks and by four weeks. It then uses `ts.intersect()` to combine the time series in a data frame for the times where they intersect. The data frame is then be used in an OLS regression; different models could be fitted to identify the best explanation of mortality.

```R
# Lag the temperature
temp.l.2 = lag(temp, 2)
temp.l.4 = lag(temp, 4)

# Intersect all time series to omit leading/trailing NA
temp.df <- ts.intersect(cmort, time(cmort), part,
                        temp, temp.2, temp.l.2, temp.l.4,
                        dframe=TRUE)

# Fit the linear model including lagged temperature
summary(lm(cmort ~ time.cmort. + temp + temp.2 +
                   temp.l.2 + temp.l.4 + part, data=temp.df))
```

6 Stationarity

The concept of *stationarity* is central to time series analysis. Stationarity means that the statistical characteristics of a time series do not change over time. That is, its mean, variance, and autocorrelation (the correlation of a time series with a lagged copy of itself) remain constant over time. Understanding and ensuring stationarity in a time series is important for the effective application of many statistical forecasting methods and models.

Stationary data with a constant mean and variance is more predictable and therefore easier to model. Changes in mean and variance can lead to forecasts that are biased or that degrade in accuracy over time. Stationarity ensures that the properties of the series used to generate forecasts will be similar in the future, which is crucial for planning and decision-making. If a time series is non-stationary, the behavior of the data could change over time, leading to models that are invalid or inaccurate when applied to future data points.

Statistical inference in time series analysis relies heavily on the assumption of stationarity. Many time series statistical models, including linear regression and ARMA models, are based on the assumption of stationarity. These models provide meaningful and reliable results only if the stationarity assumption is satisfied.

Strict stationarity is defined as the requirement that the probabilistic behaviour of every set of values of the series

$$\{x_{t_1}, x_{t_2}, \ldots, x_{t_k}\}$$

is identical to that of the set of values shifted by time h:

$$\{x_{t_1+h}, x_{t_2+h}, \ldots, x_{t_k+h}\}$$

That is,

$$\Pr\{x_{t_1} \leq c_1, \ldots, x_{t_k} \leq c_k\} = \Pr\{x_{t_1+h} \leq c_1, \ldots, x_{t_k+h} \leq c_k\}$$

Because strong stationarity is hard to test, a more commonly used and practical form of stationarity is *weak stationarity*, which requires only that the the mean, variance, and the *autocovariance* (the covariance of the series with a lagged version itself) are constant over time. Most statistical tests and models assume weak stationarity. In summary, a weakly stationary time series is a finite variance process such that:

1. The mean and variance are constant and do not depend on time: $\mu_t = \mu$, $\sigma_t = \sigma$
2. The autocovariance γ depends on s and t only through their difference $h = |s - t|$.

Let $s = t + h$, then under the assumption of weak stationarity:

$$\begin{aligned} \gamma(s,t) &= \gamma(t+h,t) && \text{(because of condition 2)} \\ &= cov(x_{t+h}, x_t) && \text{(because of condition 1)} \\ &= cov(x_h, x_0) = \gamma(h) && \text{(autocovariance for lag } h\text{)} \end{aligned}$$

and

$$\rho(h) = \gamma(h)/\gamma(0) \qquad \text{(autocorrelation for lag } h\text{)}$$

The *autocovariance* and *autocorrelations* are measures of dependence of the time series on lagged versions of itself. For a weakly stationary time series, the theoretical autocovariance for a lag h is defined as the covariance between two points $t, t + h$ on time series x

$$\gamma(h) = cov(x_t, x_{t+h}) = E[(x_t - \mu)(x_{t+h} - \mu)]$$

Note that this definition implies weak stationarity because a constant term for the mean μ is used in the expectation on the right-hand side.

A large autocovariance indicates a "smooth" time series, as each future value is strongly dependent on the previous value(s). In contrast, a small autocovariance indicates the "choppy" time series, as there is less dependence on prior values and values of the time series are less constrained and allowed to vary more.

The sample autocovariance that can be estimated from a finite sample for lag h is defined as

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-h} (x_t - \bar{x})(x_{t+h} - \bar{x})$$

The *autocorrelation function* (ACF) for lag h is defined as usual as the autocovariance divided by the root of the product of the variances of the two time series:

$$\rho_x(h) = \frac{\gamma(t+h,t)}{\sqrt{\gamma(t+h,t+h)\gamma(t,t)}} = \frac{\gamma(h)}{\gamma(0)} \quad \text{(weak stationarity)}$$

Note that this assumes weak stationarity. The time series properties at any time t are the same as at time 0 so that the above equation can be reduced to the right-most term.

Similar to the sample autocovariance, the sample ACF for lag h is defined as

$$\hat{\rho}_x(h) = \frac{\hat{\gamma}(h)}{\sqrt{\hat{\gamma}(h)\hat{\gamma}(0)}} = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)} \quad \text{(weak stationarity)}$$

where the last step again assumes weak stationarity.

To test whether the ACF of any sequence for lag h is statistically different from 0, note that the large-sample distribution of $\hat{\rho}_x(h)$ is normal with mean 0 and standard deviation

$$\sigma_{\hat{\rho}_x} = 1/\sqrt{n}$$

if the generating processes is independent white noise. Hence, the approximate 95% confidence interval on the ACF is

$$-\frac{1}{n} \pm \frac{2}{\sqrt{n}}$$

If the sample ACF of n values of a time series for a given lag exceeds the lower or upper bounds of the confidence interval, the ACF is statistically significantly different from 0, and the time series is unlikely to be white noise.

The following R code block illustrates the autocorrelation function using the standard `cor()` function to compute the correlations at different lags and the `acf1()` function of the `astsa` library that will automatically lag the time series and output and optionally plot the ACF values at different lags, creating a plot as in Figure 11.

R

```
library(astsa)
# Create Gaussian white noise
t <- ts(rnorm(500))

# The hard way:
cor(ts.intersect(t, lag(t,1), dframe=T))
cor(ts.intersect(t, lag(t,2), dframe=T))
# etc.

# The easy way:
# Without plot
acf <- acf1(t, plot=FALSE)
# With plot
acf1(t, gg=T, col=7, lwd=3)
```

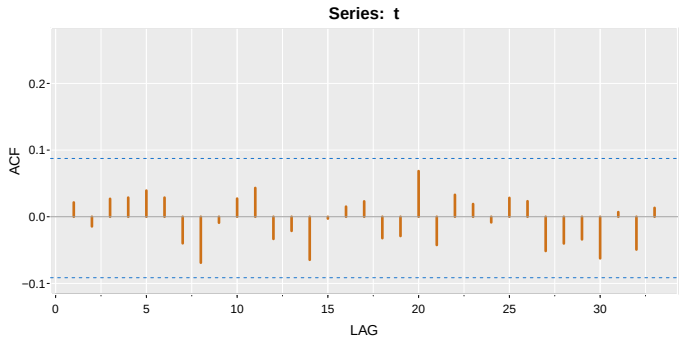


Figure 11: ACF of Gaussian white noise

The following example in R uses the `soi` data set and the `lag1.plot()` from the `astsa` library to provide also a graphical display of the autocorrelations at various lags, as shown in Figure 12.

```R
library(astsa)

# Compute and plot the ACF for different lags
acf1(soi, gg=T, co=3, lwd=2)

# Scatterplot of original versus or lags up to 6, with ACF values
lag1.plot(soi, max.lag = 6, gg=T, col=4, lwl=3)
```

The *partial autocorrelation function* (PACF) of a time series is a measure of the correlation between observations at two points in time, accounting for the correlations of the observations at all shorter intervals. Essentially, it reflects the direct effect of past data points on the future data point, after removing the effects of intermediate data points. PACF can be thought of as the correlation between a variable and its lag h that is not explained by correlations at all lower-order lags. It is formally defined as the correlation between x_{t+h} and x_t with the linear dependence of $\{x_{t+1}, \ldots, x_{t+h-1}\}$ on each removed:

$$\phi_{hh} = \begin{cases} \rho(1) & h = 1 \\ \text{corr}(x_{t+h} - \hat{x}_{t+h}, x_t - \hat{x}_t) & h \geq 2 \end{cases}$$

The following R code block illustrates the use of the partial autocorrelation function of a time series, first using the standard `cor()` function for a lag of 3 and then the `acf1()` function of the `astsa` library that automatically computes the PACF for different lags.

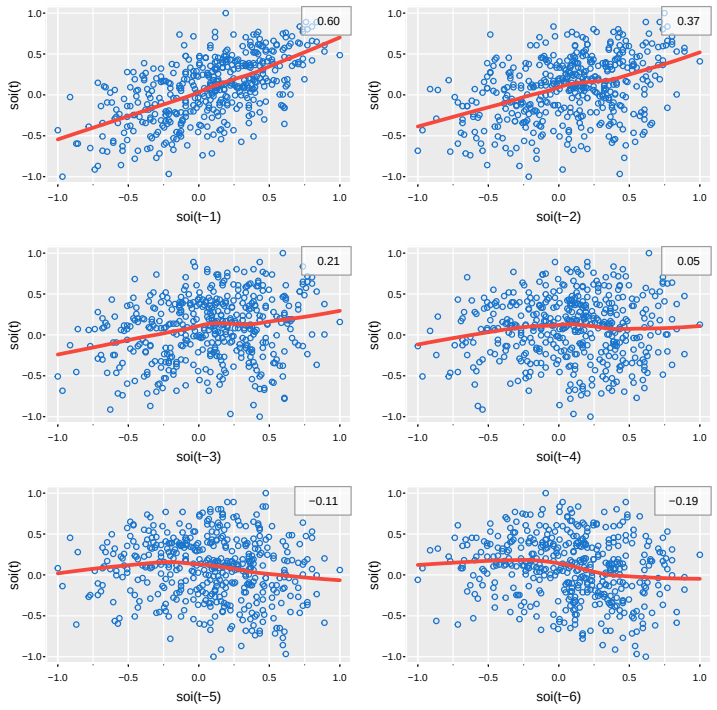


Figure 12: Autocorrelations at six different lags

R

```R
t <- ts(rnorm(500))
# The hard way
# Shift the series to create lagged versions
t1 <- lag(t, 1)
t2 <- lag(t, 2)
t3 <- lag(t, 3)
data <- ts.intersect(t, t1, t2, t3, dframe=T)

# Using linear models to adjust for intervening lags
model_lag1 <- lm(t ~ t1 + t2, data)
model_lag2 <- lm(t1 ~ t2, data)
# Residuals for lag 3
residuals_lag1 <- residuals(model_lag1)
residuals_lag2 <- residuals(model_lag2)
final_model <- lm(residuals_lag1 ~ residuals_lag2)
# Correlation between residuals and lag 3 data
pacf_lag3 <- cor(residuals(final_model), data$t3)

# The easy way
acf1(t, plot=F, pacf=T)
```

7 Dealing with Non-Stationarity

When a time series is non-stationary, it can often be transformed into a stationary series through techniques such as logarithmic or square root transformations, detrending, and differencing. These transformations can stabilize the mean and reduce variance dependency over time.

Transformations

Popular *time series transformations* are the log transformation, the square root transformation and the Box-Cox power transformation, defined as follows:

$$
\begin{aligned}
y_t &= \log x_t && && \text{Log transformation} \\
y_t &= \sqrt{x_t} && && \text{Square root transformation} \\
y_t &= \begin{cases} (x_t^\lambda - 1)/\lambda & \lambda \neq 0 \\ \log x_t & \lambda = 0 \end{cases} && && \text{Box-Cox power transformation}
\end{aligned}
$$

Detrending

Detrending a time series involves removing the trend component from the data, thereby isolating the non-trend components such as seasonality and irregular fluctuations. This is particularly useful in time series analysis because many statistical methods assume stationarity (constant mean and variance), and a trend violates these assumptions.

A common detrending method is to fit a regression model to the trend component and then subtract the fitted values, that is, the trend, from the original series. Linear regression is widely used for linear trends, but polynomial or more complex models can be fitted depending on the nature of the trend.

For example, assume that

$$x_t = \mu_t + y_t$$

where μ_t is the trend and y_t a stationary series. Then detrending comprises the following two steps:

1. Estimate trend, e.g. with a linear model such as $\mu_t = \beta_0 + \beta_1 t$
2. Work with residuals, e.g. $\hat{y}_t = x_t - \hat{\mu}_t = x_t - \hat{\beta}_0 - \hat{\beta}_1 t$

The following R code block shows how to detrend a time series using linear regression, producing the graphs shown in Figure 13.

Figure 13: Time series and detrended time series

R

```r
# Simulate a time series with a linear trend
t <- ts(1:100 + rnorm(100) * 10)

# Fit a linear model to the time series
trend_model <- lm(t ~ time(t))
# Calculate detrended series
detrended <- residuals(trend_model)

# Plot original and detrended
par(mfrow=c(2,1))
tsplot(t, type="l", main="original", col=3)
tsplot(detrended, type="l", main="detrend", col=2)
```

Differencing

Differencing involves computing the differences between consecutive observations in the original time series. The primary goal of differencing is to remove trends and seasonality in order to stabilize the mean of the time series by reducing changes in the level of a time series over time. Assume again that

$$x_t = \mu_t + y_t$$

where μ_t is the trend and y_t a stationary series. Differencing models the trend stochastically as a random walk with drift:

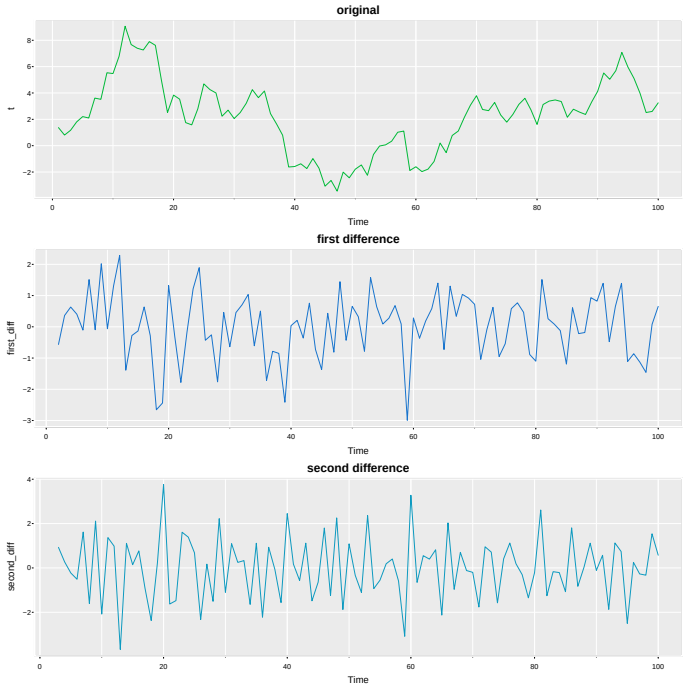


Figure 14: Original, first and second differences of a simulated time series

$$\mu_t = \delta + \mu_{t-1} + w_t$$

where w_t is white noise. Differencing then yields

$$\begin{aligned} x_t - x_{t-1} &= (\mu_t + y_t) - (\mu_{t-1} + y_{t-1}) \\ &= \delta + w_t + y_t - y_{t-1} \end{aligned}$$

which is stationary.

As seen above, the first difference can remove a linear trend. However, sometimes the first difference is not enough to achieve stationarity. In such cases, the second difference can be used to remove a quadratic trend and higher-order differences can be computed if the series still shows non-stationary behavior after the second differencing.

The following R code shows the effect of differencing on a simulated time series. Differencing uses the `diff()` function. The resulting plots are shown in Figure 14.

```R
set.seed(42)
# Simulating a time series with trend
t <- ts(cumsum(rnorm(100)))

# Plot original, first, and second difference
par(mfrow=c(3,1))
tsplot(t, type="l",
    main="original", col=3)

tsplot(diff(t, differences = 1), type="l",
    main="first difference", col=4)

tsplot(diff(t, differences = 2), type="l",
    main="second difference", col=5)
```

To see illustrate the effects of detrending and differencing on the ACF for a real time series, consider the chicken price data set `chicken` in the `astsa` library. Figure 15 shows the ACF for the original, the detrended, and the differenced series (first and second differences). While the original time series is clearly non-stationary with large ACF values (top left panel), the detrended series improves this somewhat, but still shows large ACF (top right panel). First differencing reduces the ACF values and shows a cyclical trend with a cycle of 6 months, with significant ACF values (bottom left panel). The second difference in the bottom right panel of Figure 15 still shows significant ACF values at the 6 month and 12 month lags but non-significant ACF for most other lags.

```R
acf1(chicken, max.lag=48, main="original", col=1, gg=T)

acf1(resid(fit), max.lag=48, main="detrend", col=2, gg=T)

acf1(diff(chicken), max.lag=48,
    main="first diff", col=3, gg=T)

acf1(diff(chicken, differences=2), max.lag=48,
    main="sec diff", col=4, gg=T)
```

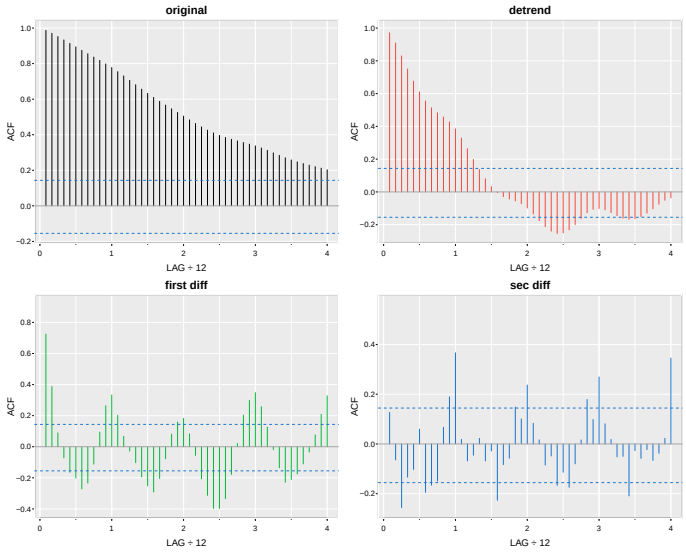


Figure 15: ACF for detrended and differenced time series

Hands-On Exercise

1. Extend the mortality, temperature and pollution/particulate model by adding another component to the regression that accounts to the particulate four weeks prior; that is, add the lagged pressure P_{t-4} to the regression.
2. Draw a scatterplot matrix of of mortality M_t, temperate T_t, pressure P_t and lagged pressure P_{t-4}, then calculate the pairwise correlations between them. Compare the relationship between M_t and P_t versus M_t and P_{t-4}

Source: Shumway & Stoffer, Chapter 2

Hands-On Exercise

1. Detrend the `soi` time series data by fitting a regression of S_t on time t. Is there a significant trend in the surface pressure?
2. Use two different smoothing techniques to estimate the trend in the global temperature series `gtemp_both` in the `astsa` library.

Source: Shumway & Stoffer, Chapter 2

Hands-On Exercise

Consider the two weekly time series `oil` and `gas` in the `astsa` library. The oil series is in dollars per barrel, while the gas series in in cents per gallon.

1. Plot the data on the same graph. Do you believe the series are stationary?
2. Apply the transformation $y_t = \log x_t$ and then apply the first difference to the data for both series
3. Plot the transformed series on the same graph, and calculate the ACFs for both series
4. Plot the CCF of the transformed series and comment.

Source: Shumway & Stoffer, Chapter 2

8 ARIMA Models

ARIMA models, which stands for Autoregressive Integrated Moving Average, are a type of statistical models for analyzing and forecasting time series data. ARIMA is particularly suited to time series data that show non-stationarities, such as trends and seasonal patterns, and it has become a standard tool in econometrics, finance, and other fields.

ARIMA models can be divided into the following model classes:

- **AR**: pure AutoRegressive models
- **MA**: pure Moving average models
- **ARMA**: model with AutoRegressive and Moving-Average terms
- **ARIMA**: AutoRegressive Integrated Moving-Average models (involves differencing for non-stationary time series with trend)

To simplify working with ARIMA models, the difference operator ∇ is defined as:

$$\nabla x_t = x_t - x_{t-1}$$

Building on this definition, the *Backshift operator* or *Lag Operator* B is defined as:

$$
\begin{aligned}
\mathrm{B}\, x_t &= x_{t-1} \\
\mathrm{B}^k\, x_t &= x_{t-k} \\
\nabla x_t &= (1-\mathrm{B})x_t \\
\nabla^2 x_t &= (1-\mathrm{B})^2 x_t \\
&= (1-2\,\mathrm{B}+\mathrm{B}^2)x_t \\
&= x_t - 2x_{t-1} + x_{t-2} \\
\nabla^d &= (1-\mathrm{B})^d
\end{aligned}
$$

An *autoregressive model* of order p, denoted by AR(p), models the current value of a time series as a linear combination of previous values. The number of lagged observations used in the model is denoted by the order p. The AR model captures the regression of the time series on its previous values, which indicates persistence, or memory, within the series. It is defined as:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + w_t$$

where w_t is white noise and the ϕ_i are model parameters[2].

The *autoregressive operator* $\phi(B)$ is defined using the backshift operator as:

$$
\begin{aligned}
\phi(B) &= 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \\
&= \left(1 - \sum_{j=1}^{p} \phi_j B^j\right)
\end{aligned}
$$

so that the AR(p) model becomes:

$$\phi(B) x_t = w_t$$

The theoretical ACF of a given AR(p) model can be calculated analytically. In R, the `ARMAacf()` function can be used for this by spacifying the autoregressive coefficients ϕ. The following R code block simulates 200 observations of an AR(2) time series and plots the simulated (blue) versus theoretical (red) ACF values, shown in Figure 16. The theoretical values can be used to determine whether a specific time series conforms to a particular AR(p) model. The ACF of an AR(p) model is characterized by a slow decline of its values past a lag of p, as shown in Figure 16.

[2]In contrast to an "ordinary" regression model, the x_i are random effects, not fixed, because each x_i has an associated error term w_t. This means that AR or ARIMA models in general are not estimated using OLS because the OLS assumptions are not met. Instead, AR and ARIMA models are estimated using maximum-likelihood or other methods.

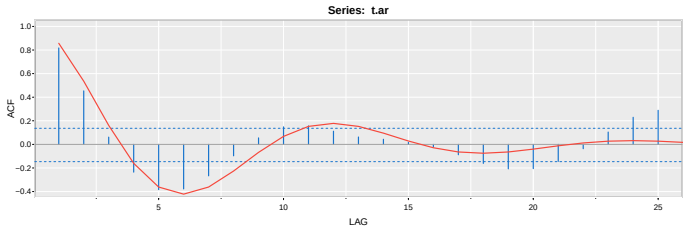


Figure 16: Simulated (blue) and theoretical (red) ACF of an AR(2) model

R

```r
# Theoretical ACF of an AR(2) model
ARMAacf(ar=c(1.5, -.75), lag.max=10)
# Simulate an ARIMA(2,0,0) model with those AR coefficients
t.ar = arima.sim(list(ar=c(1.5, -.75)), n=200)
# Compute and plot the ACF of the simulated series
acf1(t.ar, max.lag=25, gg=T, lwd=2, col=4)
# Add the theoretical values for comparison
lines(ARMAacf(ar=c(1.5, -.75), lag.max=26)[-1], lwd=2, col=2)
```

A *moving average model* or order q, denoted by MA(q), models the current value of the series as a linear combination of past forecast errors, which are computed as differences between past values and their respective forecasts. The parameter q specifies the number of lagged forecast errors in the prediction equation. The MA model is usefule for capturing "shock errors" in the model, providing a way to allow the model to adapt to sudden changes in the series. It is defined as:

$$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + \cdots + \theta_q w_{t-q}$$

where w_t are Gaussian errors and θ_i are model parameters.

The *moving average operator* $\theta(B)$ is defined using the backshift operator as:

$$\begin{aligned}\theta(B) &= 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q \\ &= \left(1 + \sum_{j=1}^{q} \theta_i B^i\right)\end{aligned}$$

so that the MA(q) model becomes:

$$x_t = \theta(B) w_t$$

The theoretical ACF of a given MA(q) model can be calculated analytically. In R, the `ARMAacf()` function can be used for this, by specifying the moving average coeffi-

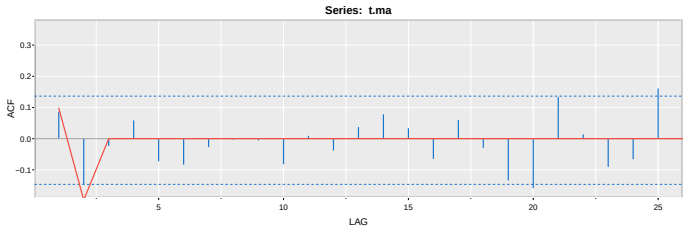


Figure 17: Simulated (blue) and theoretical (red) ACF of an MA(2) model

cients θ. Similar to the previous example, the following R code simulates 200 observations of an MA(2) model and plots the simulated (blue) versus theoretical (red) ACF values, shown in Figure 17. In contrast to an AR(2) model, the ACF does not gradually diminish, but becomes 0 after lag q. The simulated values in Figure 17 confirm this as they are largely stastistically non-significant past a lag of 2.

R

```r
# Theoretical ACF of an MA(2) model
ARMAacf(ma=c(1.5, -.75), lag.max=10)
# Simulate an ARIMA(0,0,2) model with those MA coefficients
t.ma = arima.sim(list(ma=c(1.5, -.75)), n=200)
# Compute and plot the ACF of the simulated series
acf1(t.ma, gg=T, lwd=2, col=4)
# Add the theoretical values for comparison
lines(ARMAacf(ma=c(1.5, -.75), lag.max=26)[-1], lwd=2, col=2)
```

An *autoregressive moving-average model* of order (p, q), denoted by *ARMA(p,q)*, combines both autoregressive and moving-average terms in the same model:

$$x_t = \alpha + \phi_1 x_{t-1} + \cdots + \phi_p x_{t-p} + w_t + \theta_1 w_{t-1} + \cdots + \theta_q w_{t-q}$$

Using the AR and MA operators defined above, this model can be written as:

$$\phi(B)x_t = \theta(B)w_t$$

It turns out that every ARMA model has an equivalent MA only model. However, this equivalent MA model in theory has an inifinite number of MA terms. In practice, a reasonable approximation can be achieved by retaining a limited number of MA terms.

Moreover, many ARMA models (the class of invertible ones) have an equivalent AR models. Again, this equivalent model has an infinite number of AR terms and again, in practice, reasonable approximations can be achieved by retaining a limited number of AR terms.

Equivalent models can be found using the `ARMAtoMA()` and `ARMAtoAR()` functions in the `astsa` library, which return the MA and AR coefficients of the equivalent models, as shows in the following R code example:

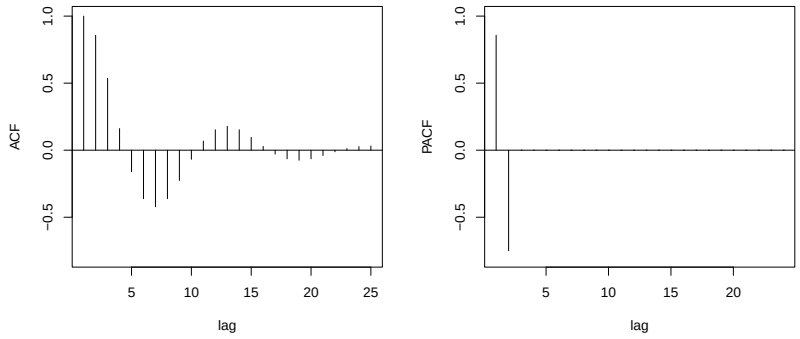


Figure 18: ACF and PACF of an AR(2) model

	AR(p)	MA(q)	ARMA (p, q)
ACF	Tails off	Cuts off after laq q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

Source: Shumway&Stoffer, Table 3.1

Table 1: Properties of the ACF and PACF for AR and MA models

```R
library(astsa)
# MA coefficients of equivalent MA models
ARMAtoMA(ar = c(1.5, -.75), lag.max=10)
ARMAtoMA(ar = c(-.5), ma = c(-.9), lag.max=10)
# AR coefficients of equivalent AR models
ARMAtoAR(ma = c(1.5, -.75), lag.max=10)
ARMAtoAR(ar = c(-.5), ma = c(-.9), lag.max=10)
```

As shown in Figures 16 and 17, the ACF of AR and MA models behave differently. Similarly, the PACF behaves differently for the two types of models. Figure 18 shows the ACF and the PACF for an AR(2) model. While the ACF diminishes gradually, the PACF is zero immediately after lag 2. These properties of the ACF and PACF can be used to select a suitable statistical model to fit a given time series, as shown in Table 1. When a AR gradually diminishes and the PACF cuts off suddenly after a lag p, this is an indication that an AR(p) model is suitable. Conversely, when the ACF cuts off suddenly after lag q and the PACF diminishes gradually, this is an indication for an MA(q) model. When neither ACF nor PACF cut off suddenly, a mixed ARMA(p, q) model should be fitted.

A full *autoregressive integrated moving average* ARIMA(p, d, q) model adds a differ-

encing term to the ARMA(p,q) model to achieve weak stationarity of the time series. The AR operator can be factorized by $(1 - B)$, so that:

$$\begin{aligned}\phi(B) &= \left(1 - \sum_{j=1}^{p'} \phi_j B^j\right) \\ &= \left(1 - \sum_{j=1}^{p'-d} \phi_j B^j\right)(1 - B)^d\end{aligned}$$

With $p = p' - d$, the ARIMA(p,d,q) model is then:

$$\left(1 - \sum_{j=1}^{p} \phi_j B^j\right)(1 - B)^d x_t = \left(1 + \sum_{j=1}^{q} \theta_j B^j\right) w_t$$

This can be generalized to:

$$\left(1 - \sum_{j=1}^{p} \phi_j B^j\right)(1 - B)^d x_t = \delta + \left(1 + \sum_{j=1}^{q} \theta_j B^j\right) w_t$$

9 Fitting an ARIMA Model

Fitting an ARIMA model to time series data involves the following steps, from initial data analysis and transformation to final model selection:

1. Plot the data
2. Possibly transform the data
3. Assess stationarity
4. Possibly difference the data
5. Identify the dependence orders (p, q) of the model
6. Estimate parameters
7. Model diagnostics
8. Model selection

Plotting the data is useful as an initial visual assessment of stationarity, trends, or seasonality. A number of transformations have been discussed earlier that may be useful to "stabilize" a time series. If the series after transformation is still not stationary, differencing can remove trends and seasonal components. Determining the order of differencing needed to achieve stationarity is often done by trial and error, reassessing stationarity after each difference. A slow decay in the sample ACF $\hat{\rho}(h)$ typically

indicates a need for differencing. However, over-differencing can introduce dependence where non actually exists. Typically, differencing should be done in small steps, beginning with a first difference, and then repeatedly checked with the ACF.

Identifying the initial ARMA order p and q should be done based on the ACF and PACF functions, using Table 1 as a basis. Often, multiple model may need to be tried, altering the AR and MA orders p and q in small steps.

The following R code example uses a quarterly time series of the US gross national product (`gnp`) from the `astsa` library as an example. The `acf2()` function of the `astsa` library produces simultaneous plots of ACF and PACF for ease-of-use. The time series is then log-transformed and differenced once. The results are shown in Figure 19.

R

```
# Plot data
plot(gnp)
# Plot ACF
acf2(gnp, 50)
# Log transform, and first order differencing
gnpgr = diff(log(gnp))
# Plot transformed and differenced data
plot(gnpgr)
# Plot ACF of transformed and differenced data
acf2(gnpgr, 24)
```

Note how the ACF of the original series diminishes very gradually, indicating the need for differencing. The sample ACF of the transformed and differenced series shows a gradual decline after a lag of 2, while the sample PACF of the transformed and differenced series cuts off to non-significance after a lag of 1. Together, this indicates that the time series may be appropriately modeled using an AR(1) model or an MA(2) model. Converting the AR(1) model to an equivalent MA model shows that the two initial models are approximately equivalent. Note that the following R code block uses the `sarima()` function from the `astsa` library because this function also produces the diagnostic plots shown in Figures 20 and 21. This function can fit ARIMA models, as in the following example, but can also fit seasonal ARIMA, or SARIMA, models.

R

```
# Fit an AR(1) model
sarima(gnpgr, 1, 0, 0)
# Fit an MA(2) model
sarima(gnpgr, 0, 0, 2)
# Models are roughly equivalent
ARMAtoMA(ar=0.35, ma=0, 10)
```

ARIMA model diagnostics focus on the residuals of the fitted model and typically assess the following criteria:

- Standardized residuals should be Gaussian ($\mu = 0, sd = 1$)
- Residuals should not be autocorrelated

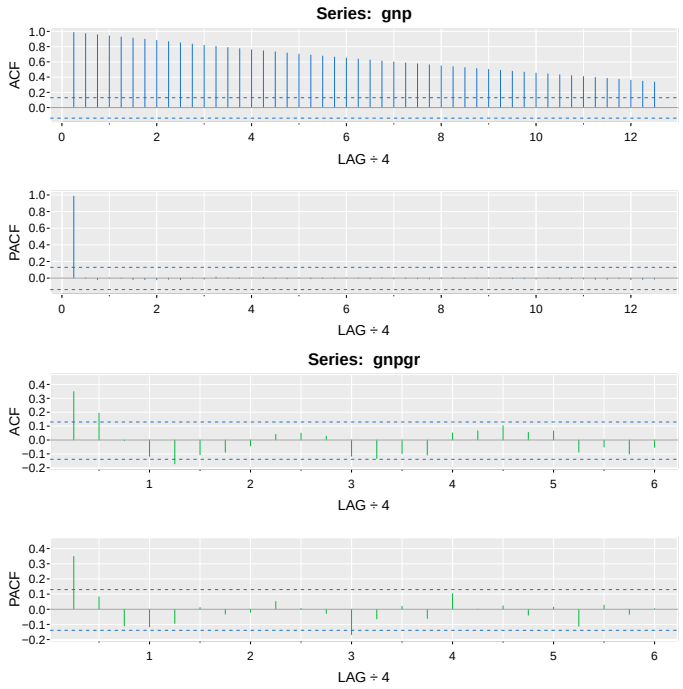


Figure 19: ACF and PACF of the original time series (top) and of the log-transformed and differenced series (bottom)

- Residual ACF should be Gaussian with $\mu = 0$ and $sd = 1/\sqrt{n}$
- Ljung-Box statistic Q of the error ACF $\hat{\rho}_e$ for different maximum lags H should be larger than the $1-\alpha$ quantile of the χ^2_{H-p-q} distribution (i.e. the test statistic is not statistically signifantly different from 0)

$$Q = n(n+2)\sum_{h=1}^{H}\frac{\hat{\rho}_e^2(h)}{n-h}$$

The diagnostic plots in Figure 20 and 21 shows the residuals in the top panels. There are no visible trends or regularities and a few large outliers, but these can be expected from a Gaussian distribution. The ACF of the residuals in the middle left panel show that they are not autocorrelated and the QQ plot of residuals shows some deviations from a linear diagonal in the bottom and top portions, indicating that the model over- and under-estimates extreme values. The bottom panel in each figure shows the probability values (p-values) for the Ljung Box statistics and the $1-\alpha$ dashed horizontal line. For both models, the p-values of the Ljung Box statistic are above the horizontal line, that is, they are not significantly different. In summary, both models show good fit to the data.

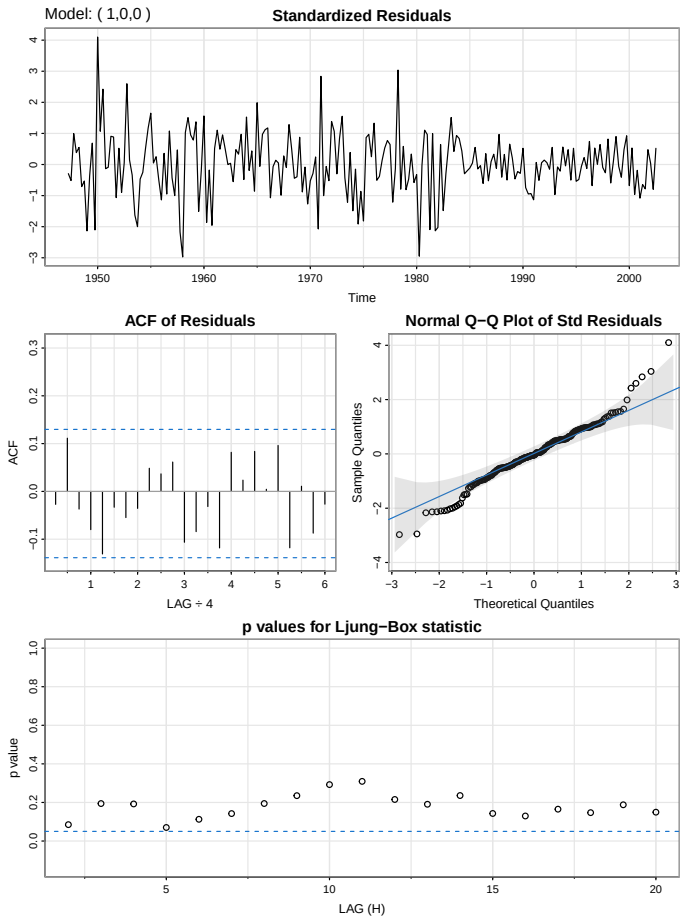


Figure 20: Diagnostics for an AR(1) model fitted to the GNP time series

Because fitting a model typically uses maximum-likelihood estimation (MLE), the model choice is often based on information criteria that are based on the log-likelihood L of the model. Because more complex models naturally achieve a better fit to the training data, that is, they have a smaller bias, the log-likelihood is adjusted (penalized) for model complexity, that is, the number of parameters k, and is also adjusted for sample size n. All information-theoretic criteria express a *relative* quality of fit with *smaller values being better*. There are no absolute cut-off values that would indicate a well-fitting model.

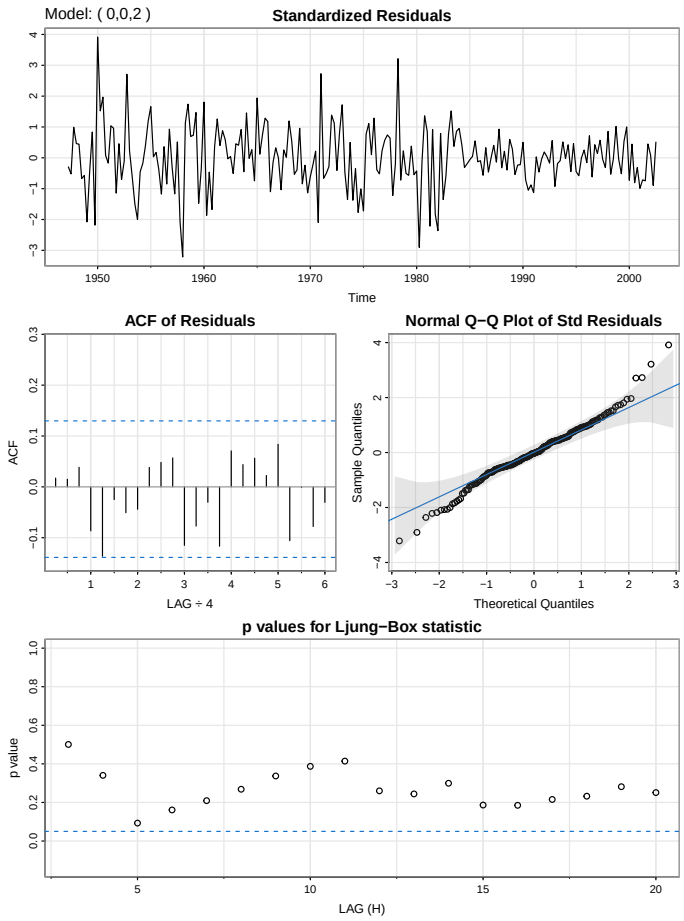


Figure 21: Diagnostics for an MA(2) model fitted to the GNP time series

$$\text{AIC} = -2\log L + 2k \qquad \text{Akaike Information Criterion}$$

$$\text{AICc} = AIC + \frac{2k(k+1)}{n-k-1} \qquad \text{Akaike Information Criterion, corrected}$$

$$\text{BIC} = -2\log L + k\log n \qquad \text{Bayesian Information Criterion}$$

The R output of the `sarima()` function shows very similar model fit values:

Text

```
> sarima(gnpgr, 1, 0, 0)

AIC = -6.44694  AICc = -6.446693  BIC = -6.400958
```

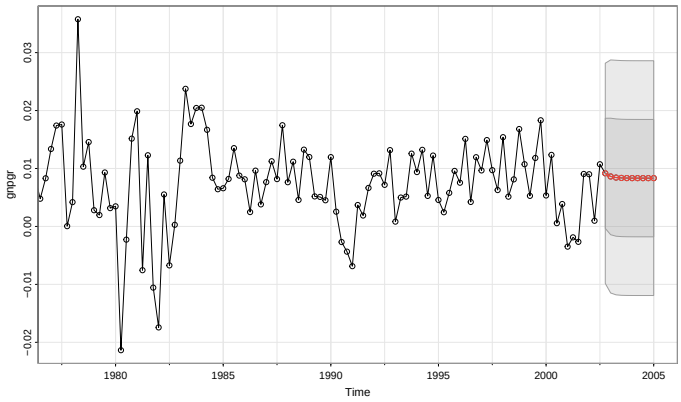


Figure 22: Forecasting from an ARIMA(1,0,0) model with estimated prediction errors

```
Text
> sarima(gnpgr, 0, 0, 2)

AIC = -6.450133  AICc = -6.449637  BIC = -6.388823
```

Once the analyst has selected the final model and is satisfied that it fits well, future values of the time series can be forecasted from the fitted model. An important property of ARIMA predictions is that they quickly settle to the mean, with a constant prediction error, reflecting the stationarity of the differenced and transformed time series.

The following R code example shows forecasting from using the `sarima.for()` function in the `astsa` library. The results are shown visually in Figure 22, where the prediction error is indicated by the gray shading.

```
R
forecasts <- sarima.for(gnpgr, n.ahead=10, p=1,d=0,q=0)
```

10 GARCH Models

General Autoregressive Conditional Heteroscedasticity (GARCH) models are a family of time series models that are used to estimate the volatility and conditional variance of time series data, particularly of financial time series that exhibit time-varying volatility and volatility clustering. GARCH models are fundamental in the field of financial econometrics for modeling financial time series data.

GARCH models predict the current variance (that is, the volatility or variability, not the actual values) as a function of past squared "innovations" (which represent unexpected shocks or news in the data) and past conditional variances. In other words, the variance at any time depends on the information available up to the previous period. GARCH

models are particularly effective at modeling volatility clustering, a phenomenon common in financial time series where high-volatility events tend to cluster together.

GARCH models are extensively used in risk management, asset pricing, and financial forecasting. They help in estimating the volatility of asset returns for pricing derivatives, calculating the value at risk for risk management, or forecasting volatility for portfolio optimization.

An ARCH model considers a series of "returns", which are defined as deviations from the prior value:

$$r_t = \frac{x_t - x_{t-1}}{x_{t-1}} \qquad \text{("Return")}$$

The series of returns is modelled as the product of a stochastic component ϵ_t and a time-dependent standard deviation σ_t

$$r_t = \sigma_t \epsilon_t$$

The ARCH model considers the time-dependent variance σ_t^2 at time t as a function of the previous returns. For example, in the ARCH(1) model the variance σ_t^2 at time t depends on the immediately prior squared return:

$$\sigma_t^2 = \alpha_0 + \alpha_1 r_{t-1}^2$$

where ϵ_t is Gaussian.

The general ARCH(q) model of order q is defined as follows. Again, the variance depends on the prior squared returns:

$$\begin{aligned} \sigma_t^2 &= \alpha_0 + \alpha_1 r_{t-1}^2 + \alpha_2 r_{t-2}^2 + \ldots + \alpha_q r_{t-q}^2 \\ &= \alpha_0 + \sum_{i=1}^{q} \alpha_i r_{t-i}^2 \end{aligned}$$

ARCH models can be combined with ARIMA models so that the ARCH model describes the error term. A simple example is an AR(1) model with ARCH(1) error terms:

$$x_t = \phi_0 + \phi_1 x_{t-1} + \sigma_t \epsilon_t \quad \text{where} \quad \sigma_t = \alpha_0 + \alpha_1 x_{t-1}^2$$

As an example, consider the US gross national product quarterly time series from the `astsa` library. An initial AR(1) model shows that the squared residuals have some dependence. This dependence can be accounted for by explicitly modeling the residuals

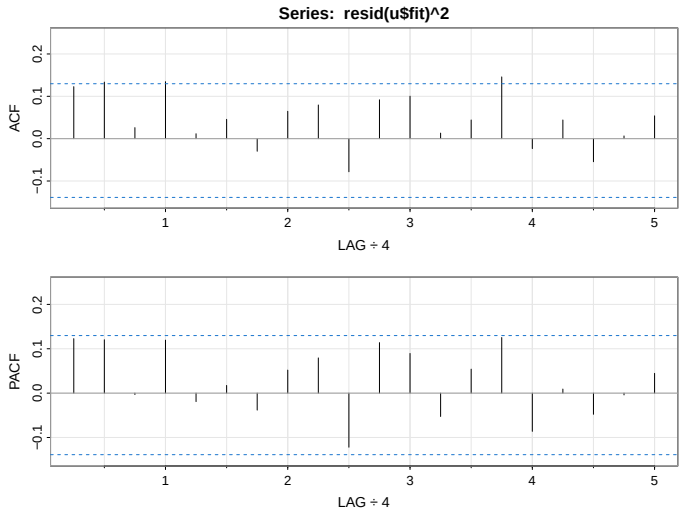


Figure 23: Squared residuals after fitting an AR(1)+ARCH(1) model to the US GNP time series data

as ARCH(1). The `fGarch` library for R provides the `garchFit()` function to these kinds of models. The squared residuals of the final model are shown in Figure 23.

R

```
library(astsa)
# Fit an AR(1) model to the differenced, log-transformed series
u = sarima(diff(log(gnp)), 1, 0, 0)
# Examine the squared residuals
acf2(resid(u$fit)^2, 20)

library(fGarch)
# Fit an AR(1) + ARCH(1) model to the differenced, log-transformed
# series and show the summary
summary(garchFit(~arma(1,0)+garch(1,0), diff(log(gnp))))
```

An extension to ARCH is to model the variance not only as a function of previous returns, but also as a function of the p prior variances. In other words, the variance is modelled as an autoregressive model in addition to the ARCH(q) terms. This leads to a Generalized ARCH model, that is, a GARCH(p, q) model:

$$\begin{aligned}\sigma_t^2 &= \omega + \alpha_1 r_{t-1}^2 + \cdots + \alpha_q r_{t-q}^2 \\ &\qquad + \beta_1 \sigma_{t-1}^2 + \cdots + \beta_p \sigma_{t-p}^2 \\ &= \omega + \sum_{j=1}^{q} \alpha_j r_{t-j}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2\end{aligned}$$

The following R example models the Dow Jones Industrial Average stock market time series values (data set `djiar` in library `astsa`) using an AR(1)+GARCH(1,1) model. Parameter estimates are shown below. Various diagnostic plots are available using the `plot()` function for the resulting `garchFit` object and are shown in Figure 24.

R

```
library(zoo)
library(fGarch)
# Log transform
djiar = diff(log(djia$Close))[-1]
# Fit an AR(1) + GARCH(1,1) model
djia.g <- garchFit(~arma(1,0)+garch(1,1), data=djiar)
# Show summary information
summary(djia.g)
# Different plots available
par(mfrow=c(5,2)
plot(djia.g, which=1:10)
```

Text

```
          Estimate  Std. Error  t value Pr(>|t|)
mu       8.585e-04   1.470e-04    5.842 5.16e-09 ***
ar1     -5.532e-02   2.023e-02   -2.735 0.006238 **
omega    1.610e-06   4.459e-07    3.611 0.000305 ***
alpha1   1.244e-01   1.660e-02    7.496 6.55e-14 ***
beta1    8.700e-01   1.526e-02   57.022  < 2e-16 ***
shape    5.979e+00   7.917e-01    7.551 4.31e-14 ***
---
Log Likelihood:
 8249.619    normalized:  3.27756
```

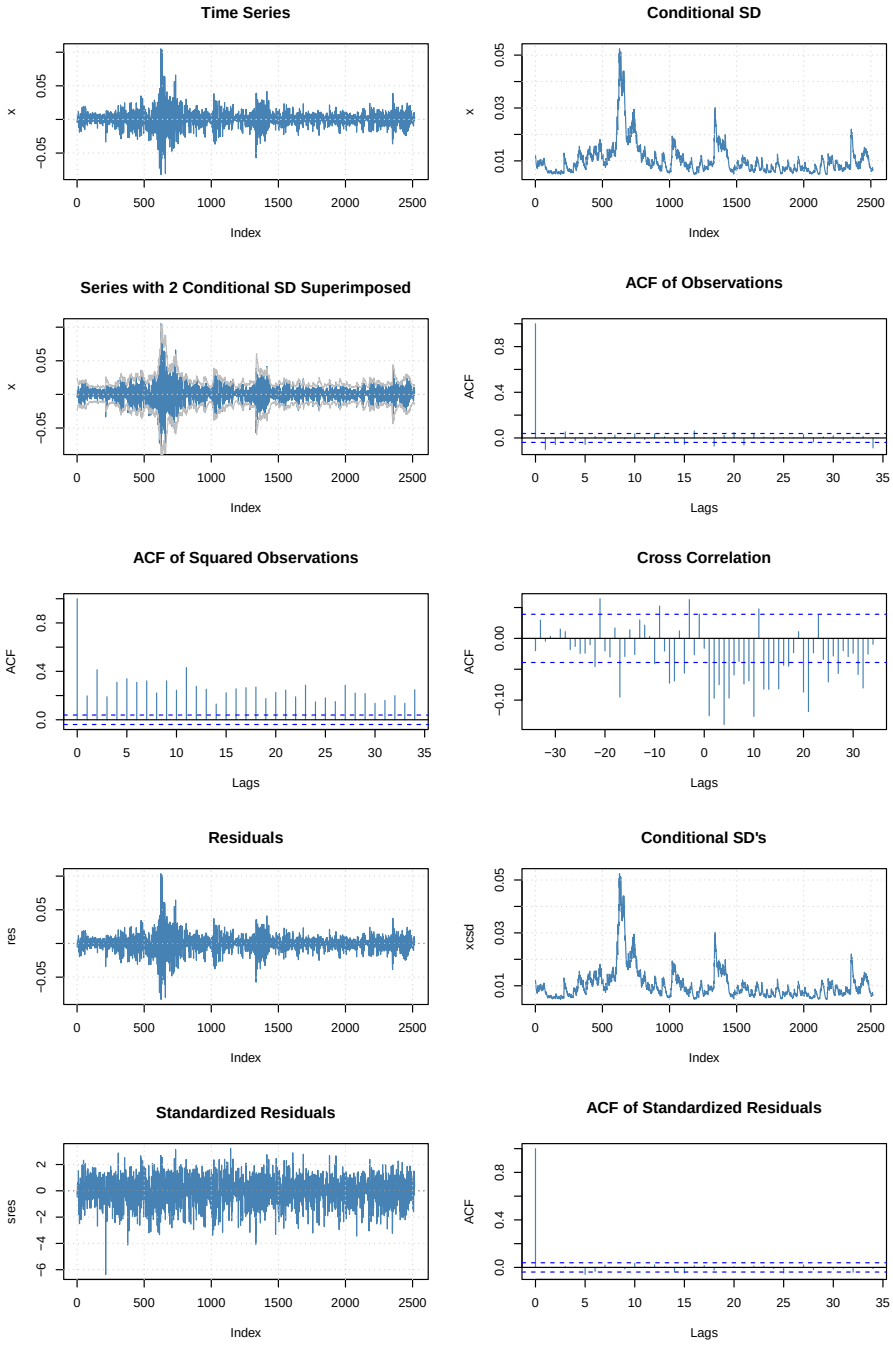


Figure 24: Diagnostic plots for a GARCH model

Appendix – Basic Time Series Functions in R

base or stats	
`filter`	Filters time series, through moving averages or autoregression
`lag`	Creates a lagged version of a time series by shifting the time-base back
`diff`	Creates lagged differences
`plot.ts`	Plot a time series
`ts.plot`	Plot multiple time series
`lag.plot`	Scatterplot of lagged values
`acf`	ACF and plot
`ccf`	CCF and plot
`time`	Creates the vector or times at which a time series was sampled
`cycle`	Gives the positions in the cycle of each observation
`frequency`	Number of samples per unit time
`ts.intersect`	Bind time series together that have a common frequency. Restrict to time covered by all series
`ts.union`	Bind time series together that have a common frequency. Pad with NA if necessary
`ar`	Fit an autoregressive model
`arima`	Fit an ARIMA model
astsa	
`tsplot`	Plot a time series
`acf1`	ACF and plot
`ccf2`	CCF and plot
`sarima`	Fit seasonal ARIMA models (and nice diagnostic plots)
`lag1.plot`	Scatterplot of lagged values

11 Review Questions

Introduction

1. What is time series analysis and why is it important in various fields such as economics, finance, and natural sciences?
2. What are some essential preprocessing steps required before performing time series analysis?
3. Compare and contrast the time-domain approach and the frequency-domain approach in time series analysis. Which approach is particularly useful for forecasting and why?

Time Series Statistical Models

4. Explain the Moving Average (MA) model and describe how it uses past error terms to forecast future values. What are the assumptions about these error terms?
5. Discuss how the Moving Average (MA) model can be utilized to detect underlying patterns in a time series that exhibits random fluctuations. What limitations does this model have in handling trend and seasonality?
6. Explain the implications of choosing different window sizes (the number of terms included) in the Moving Average model. How does it affect the forecasts and smoothing?
7. Describe how the Autoregressive (AR) model differs from the MA model and provide an example of its application in economic or financial time series.
8. In the context of the AR model equation $x_t = x_{t-1} - 0.9x_{t-2} + w_t$, analyze the impact of changing the coefficient -0.9 to values closer to 0 or 1.
9. What is a Random Walk with Drift? Describe how it models time series data and give an example of its application in financial markets.
10. Critique the usefulness of the Signal in Noise model in various fields such as economics, engineering, and environmental science. How might the assumptions of this model limit its application?
11. Critically evaluate the effectiveness of each model (MA, AR, Random Walk with Drift, Signal in Noise) in handling different types of time series data.

Smoothing a Time Series

12. Explain the purpose of smoothing in time series analysis. What are the general goals of this technique?
13. Describe the moving average smoothing method. How does this method use weights to smooth data, and what are the effects of changing these weights?
14. Discuss how the moving average method helps in reducing the impact of random fluctuations in the data. What challenges might arise when using this method on time series with trends or seasonality?
15. Explain what kernel smoothing is and how it uses a Gaussian kernel to weigh data points. How does the choice of bandwidth affect the smoothing?

16. Detail the Lowess regression method. How does this method determine the weights for smoothing and how do these weights contribute to the robustness against outliers?
17. Explain the role of the parameter `f` in the `lowess()` function in R. How does changing the value of `f` affect the results of the Lowess smoothing?
18. Define smoothing splines and describe how they fit a spline function to the data. What does the regularization term in the loss function achieve?
19. Compare and contrast the advantages and potential drawbacks of using moving average, kernel smoothing, Lowess regression, and smoothing splines. When might one method be preferred over the others based on the characteristics of the time series data?

Time Series Regression

20. Define time series regression and explain how it differs from other types of time series analysis such as autoregressive models.
21. Describe the significance of including time as a variable in the regression models. What does this imply about the data and its relationship over time?
22. Discuss the use of lagged variables in time series regression. What are the benefits of including lagged terms?

Stationarity

23. Explain the difference between strict and weak stationarity. Why is weak stationarity more commonly used in statistical analysis of time series?
24. Elaborate on the impact of non-stationarity on the predictive performance of time series models. How does failing to account for stationarity potentially mislead forecasting?
25. Critique the assumption of constant variance in the definition of weak stationarity. How might changes in variance over time affect the validity of time series models?
26. Explain how the concept of weak stationarity might still inadequately describe the nature of certain financial time series. What alternative forms of stationarity might be considered?
27. Discuss how the mean, variance, and autocovariance function must behave for a time series to be considered weakly stationary.
28. Define autocovariance and autocorrelation. How are these metrics useful in analyzing the properties of a time series?
29. Analyze the implications of having a high autocorrelation at large lags for a given time series. What might this indicate about the underlying data generation process?
30. What does it indicate if the ACF values are outside the 95% confidence interval? How does this help in determining whether a time series is white noise?
31. Describe how you would assess the stationarity of a time series using graphical methods in R. What plots would you use and what features would you look for?

Dealing with Non-Stationarity

32. Discuss how the Box-Cox transformation generalizes other forms of transformations like logarithmic and square root transformations. What is the significance of the parameter λ in this transformation?
33. Explain the statistical reasoning behind using logarithmic transformations for time series data. What types of data characteristics make this transformation particularly effective?
34. Explain the process of detrending a time series. Why is it necessary, and how does it differ from differencing?
35. Describe the impact of detrending and differencing on the forecasting accuracy of a time series model. How might these preprocessing steps improve or impair the model's performance?
36. Provide a detailed explanation of the first and second differences of a time series. Under what circumstances might second differencing be necessary?
37. Explain how the autocorrelation function (ACF) can be used to verify the effectiveness of detrending and differencing interventions on a time series.

ARIMA Models

38. Define an ARIMA model and explain the components of its notation: ARIMA(p, d, q).
39. Describe a moving average model of order q, MA(q). How does it model the current value of the series?
40. Describe the structure of an autoregressive model of order p, AR(p). What does it mean for the model to have "memory" or "persistence"?
41. Discuss the role of the differencing operator ∇ in making a time series stationary. How does this relate to the integrated component of an ARIMA model?
42. Explain the purpose of the backshift or lag operator B in the context of ARIMA models. Provide an example of how it is used to define the differencing of a series.
43. Explain how the autoregressive operator $\phi(B)$ is used to form the equation of an AR(p) model.
44. Explain the significance of the moving average operator $\theta(B)$ in an MA(q) model.
45. Describe the combined model ARMA(p, q) and how it integrates features of both AR and MA models.
46. Explain how the properties of the ACF and PACF can help in selecting an appropriate ARIMA model for a time series. Provide examples of what the ACF and PACF might look like for different models.

Fitting an ARIMA Model

47. Detail how the orders of the AR and MA components (p and q) are identified using the ACF and PACF plots.
48. Explain the importance of model diagnostics in the ARIMA modeling process. What are some key diagnostic checks that should be performed?

49. Explain how information criteria such as AIC, AICc, and BIC are used to compare the fit of different ARIMA models. What does each criterion take into account?
50. Detail the process and importance of conducting model diagnostics after fitting an ARIMA model. What specific plots and statistics are typically used?
51. Discuss the implications of the Ljung-Box test results when diagnosing the fit of an ARIMA model. What does a significant result suggest about the residuals?

General Autoregressive Conditional Heteroscedasticity (GARCH) Models

52. Define an ARCH and a GARCH model and explain their importance in financial econometrics.
53. Discuss how a GARCH models can account for volatility clustering in financial time series.
54. Describe the basic structure of an ARCH(1) model and how it models the variance of a time series.
55. Explain how an AR(1) model with ARCH(1) error terms is constructed. Include a description of how each component contributes to modeling the time series.
56. Explain the extension from an ARCH model to a GARCH model. What additional features does a GARCH model incorporate?
57. Describe how to interpret the output of a fitted GARCH model, including parameter estimates and their significance.
58. Discuss the significance of the parameters α and β in a GARCH(1,1) model. What does each parameter represent, and how do they affect the model's behavior?