Chapter 1. Time Series Analysis

Time series analysis is concerned with the analysis of data collected over time. Adjacent observations are typically dependent. Time series analysis hence deals with techniques for the analysis of this dependence.

The objective of this chapter is to introduce some common modeling techniques by means of specific applications. We will see how to use R to solve these real-world examples. We begin with some thoughts about how to store and process time series data in R. Afterwards, we deal with linear time series analysis and how it can be used to model and forecast house prices. In the subsequent section, we use the notion of cointegration to improve on the basic minimal variance hedge ratio by taking long-run trends into consideration. The chapter concludes with a section on how to use volatility models for risk management purposes.

Working with time series data

The native R classes suitable for storing time series data include `vector`, `matrix`, `data.frame`, and `ts` objects. But the types of data that can be stored in these objects are narrow; furthermore, the methods provided by these representations are limited in scope. Luckily, there exist specialized objects that deal with more general representation of time series data: `zoo`, `xts`, or `timeSeries` objects, available from packages of the same name.

It is not necessary to create time series objects for every time series analysis problem, but more sophisticated analyses require time series objects. You could calculate the mean or variance of time series data represented as a vector in R, but if you want to perform a seasonal decomposition using `decompose`, you need to have the data stored in a time series object.

In the following examples, we assume you are working with `zoo` objects because we think it is one of the most widely used packages. Before we can use `zoo` objects, we need to install and load the `zoo` package (if you have already installed it, you only need to load it) using the following command:

```
> install.packages("zoo")
> library("zoo")
```

In order to familiarize ourselves with the available methods, we create a `zoo` object called `aapl` from the daily closing prices of Apple's stock, which are stored in the CSV file `aapl.csv`. Each line on the sheet contains a date and a closing price separated by a comma. The first line contains the column headings (**Date** and **Close**). The date is formatted according to the recommended primary standard notation of ISO 8601 (YYYY-MM-DD). The closing price is adjusted for stock splits, dividends, and related changes.

Tip

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

We load the data from our current working directory using the following command:

```
> aapl<-read.zoo("aapl.csv",+   sep=",", header = TRUE, format
= "%Y-%m-%d")
```

To get a first impression of the data, we plot the stock price chart and specify a title for the overall plot (using the `main` argument) and labels for the x and y axis (using `xlab` and `ylab` respectively).

```
> plot(aapl, main = "APPLE Closing Prices on NASDAQ",+   ylab
= "Price (USD)", xlab = "Date")
```

We can extract the first or last part of the time series using the following commands:

```
> head(aapl)
2000-01-03 2000-01-04 2000-01-05 2000-01-06 2000-01-07 2000-
01-10
     27.58      25.25      25.62      23.40      24.51
24.08
> tail(aapl)
2013-04-17 2013-04-18 2013-04-19 2013-04-22 2013-04-23 2013-
04-24
    402.80     392.05     390.53     398.67     406.13
405.46
```

Apple's all-time high and the day on which it occurred can be found using the following command:

```
> aapl[which.max(aapl)]
```

```
2012-09-19
    694.86
```

When dealing with time series, one is normally more interested in returns instead of prices. This is because returns are usually stationary. So we will calculate simple returns or continuously compounded returns (in percentage terms).

```
> ret_simple <- diff(aapl) / lag(aapl, k = -1) * 100
> ret_cont   <- diff(log(aapl)) * 100
```

Summary statistics about simple returns can also be obtained. We use the `coredata` method here to indicate that we are only interested in the stock prices and not the index (dates).

```
> summary(coredata(ret_simple))
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-51.86000  -1.32500   0.07901   0.12530   1.55300  13.91000
```

The biggest single-day loss is -51.86%. The date on which that loss occurred can be obtained using the following command:

```
> ret_simple[which.min(ret_simple)]
2000-09-29
 -51.85888
```

A quick search on the Internet reveals that the large movement occurred due to the issuance of a profit warning. To get a better understanding of the relative frequency of daily returns, we can plot the histogram. The number of cells used to group the return data can be specified using the `break` argument.

```
> hist(ret_simple, breaks=100, main = "Histogram of Simple
Returns",+  xlab="%")
```

We can restrict our analysis to a subset (a `window`) of the time series. The highest stock price of Apple in 2013 can be found using the following command lines:

```
> aapl_2013 <- window(aapl, start = '2013-01-01', end = '2013-
12-31')
> aapl_2013[which.max(aapl_2013)]
2013-01-02
    545.85
```

The quantiles of the return distribution are of interest from a risk-management perspective. We can, for example, easily determine the 1 day 99% Value-at-Risk using a naive historical approach.

```
> quantile(ret_simple, probs = 0.01)
       1%
-7.042678
```

Hence, the probability that the return is below 7% on any given day is only 1%. But if this day occurs (and it will occur approximately 2.5 times per year), 7% is the minimum amount you will lose.

Linear time series modeling and forecasting

An important class of linear time series models is the family of **Autoregressive Integrated Moving Average** (**ARIMA**) models, proposed by *Box and Jenkins (1976)*. It assumes that the current value can depend only on the past values of the time series itself or on past values of some error term.

According to Box and Jenkins, building an ARIMA model consists of three stages:

1. Model identification.
2. Model estimation.
3. Model diagnostic checking.

The model identification step involves determining the order (number of past values and number of past error terms to incorporate) of a tentative model using either graphical methods or information criteria. After determining the order of the model, the parameters of the model need to be estimated, generally using either the least squares or maximum likelihood methods. The fitted model must then be carefully examined to check for possible model inadequacies. This is done by making sure the model residuals behave as white noise; that is, there is no linear dependence left in the residuals.

Modeling and forecasting UK house prices

In addition to the `zoo` package, we will employ some methods from the `forecast` package. If you haven't installed it already, you need to use the following command to do so:

```
> install.packages("forecast")
```

Afterwards, we need to load the class using the following command:

```
> library("forecast")
```

First, we store the monthly house price data (source: Nationwide Building Society) in a `zoo` time series object.

```
> hp <- read.zoo("UKHP.csv", sep = ",",+   header = TRUE,
format = "%Y-%m", FUN = as.yearmon)
```

The `FUN` argument applies the given function (`as.yearmon`, which represents the monthly data points) to the date column. To make sure we really stored monthly data (12 subperiods per period), by specifying `as.yearmon`, we query for the frequency of the data series.

```
> frequency(hp)
[1] 12
```

The result means that we have twelve subperiods (called months) in a period (called year). We again use simple returns for our analysis.

```
> hp_ret <- diff(hp) / lag(hp, k = -1) * 100
```

Model identification and estimation

We use the `auto.arima` function provided by the `forecast` package to identify the optimal model and estimate the coefficients in one step. The function takes several arguments besides the return series (`hp_ret`). By specifying `stationary = TRUE`,we restrict the search to stationary models. In a similar vein, `seasonal = FALSE` restricts the search to non-seasonal models. Furthermore, we select the Akaike information criteria as the measure of relative quality to be used in model selection.

```
> mod <- auto.arima(hp_ret, stationary = TRUE, seasonal =
FALSE,+   ic="aic")
```

To determine the fitted coefficient values, we query the model output.

```
> mod
Series: hp_ret
ARIMA(2,0,0) with non-zero mean

Coefficients:
         ar1     ar2  intercept
      0.2299  0.3491     0.4345
s.e.  0.0573  0.0575     0.1519

sigma^2 estimated as 1.105:  log likelihood=-390.97
AIC=789.94   AICc=790.1   BIC=804.28
```

An AR(2) process seems to fit the data best, according to Akaike's Information Criteria. For visual confirmation, we can plot the partial autocorrelation function using the command `pacf`. It shows non-zero partial autocorrelations until lag two, hence an AR process of order two seems to be appropriate. The two AR coefficients, the intercept (which is actually the mean if the model contains an AR term), and the respective standard errors are given. In the following example, they are all significant at the 5% level since the respective confidence intervals do not contain zero:

```
> confint(mod)
                2.5 %    97.5 %
ar1       0.1174881 0.3422486
ar2       0.2364347 0.4617421
intercept 0.1368785 0.7321623
```

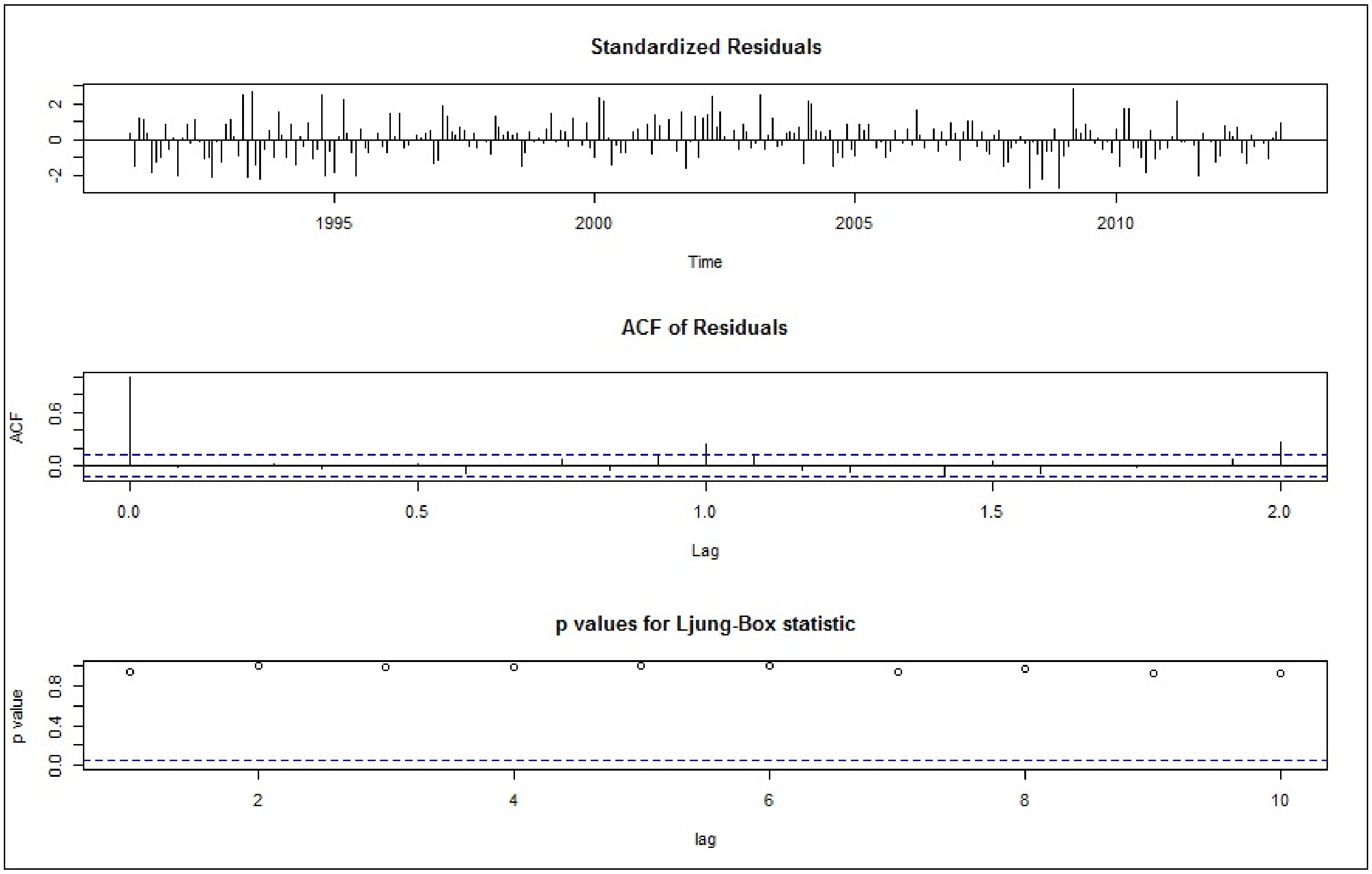
If the model contains coefficients that are insignificant, we can estimate the model anew using the `arima` function with the `fixed` argument, which takes as input a vector of elements `0` and `NA`. `NA` indicates that the respective coefficient shall be estimated and `0` indicates that the respective coefficient should be set to zero.

Model diagnostic checking

A quick way to validate the model is to plot time-series diagnostics using the following command:

```
> tsdiag(mod)
```

The output of the preceding command is shown in the following figure:

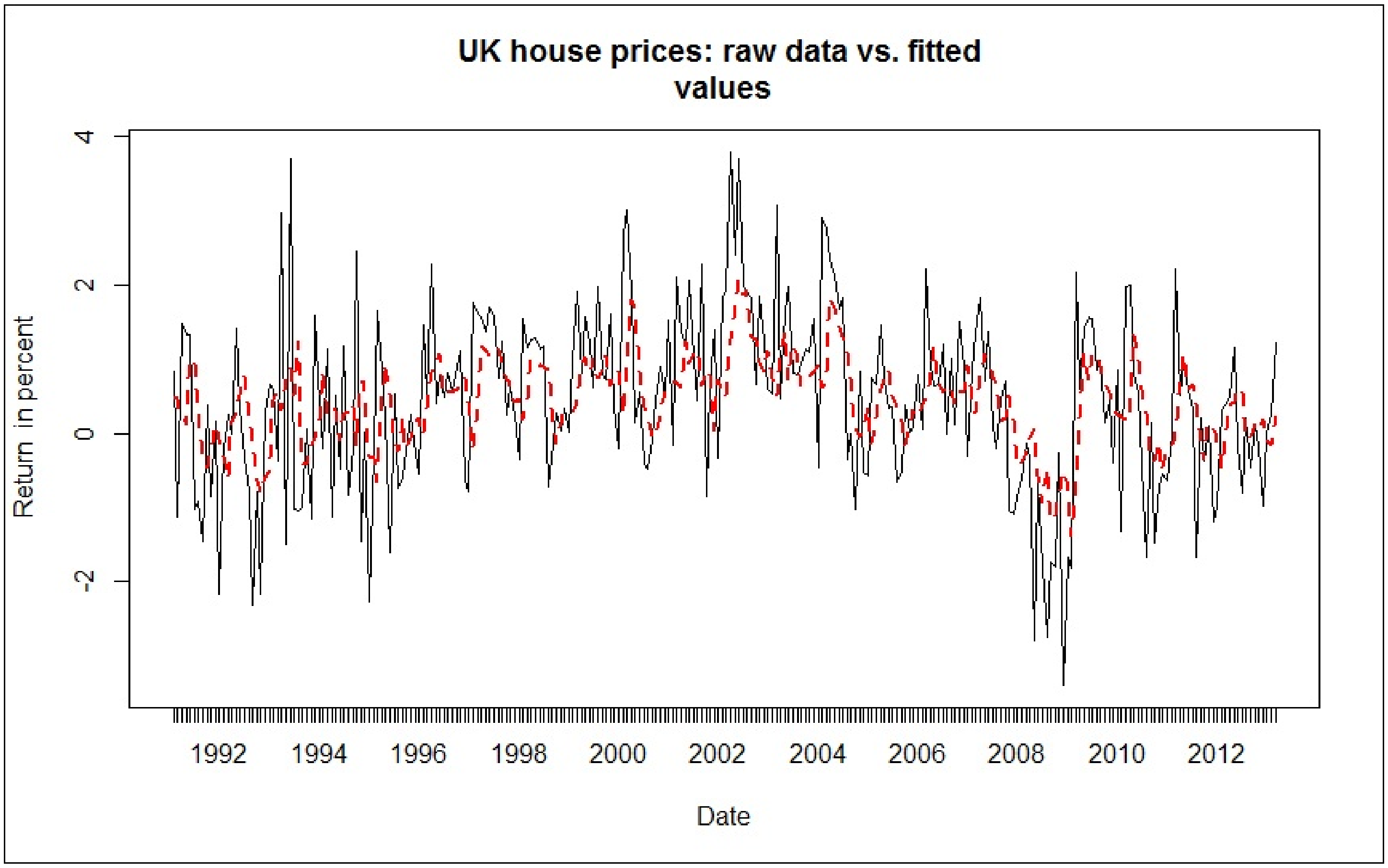


Our model looks good since the standardized residuals don't show volatility clusters, no significant autocorrelations between the residuals according to the ACF plot, and the Ljung-Box test for autocorrelation shows high p-values, so the null hypothesis of independent residuals cannot be rejected.

To assess how well the model represents the data in the sample, we can plot the raw monthly returns (the thin black solid line) versus the fitted values (the thick red dotted line).

```
> plot(mod$x, lty = 1, main = "UK house prices: raw data vs.
fitted+   values", ylab = "Return in percent", xlab = "Date")
> lines(fitted(mod), lty = 2,lwd = 2, col = "red")
```

The output is shown in the following figure:



Furthermore, we can calculate common measures of accuracy.

```
> accuracy(mod)
ME      RMSE      MAE          MPE   MAPE    MASE
0.00120 1.0514    0.8059       -Inf  Inf     0.792980241
```

This command returns the mean error, root mean squared error, mean absolute error, mean percentage error, mean absolute percentage error, and mean absolute scaled error.

Forecasting

To predict the monthly returns for the next three months (April to June 2013), use the following command:

```
> predict(mod, n.ahead=3)
$pred
           Apr       May       Jun
2013 0.5490544 0.7367277 0.5439708

$se
          Apr      May      Jun
2013 1.051422 1.078842 1.158658
```

So we expect a slight increase in the average home prices over the next three months, but with a high standard error of around 1%. To plot the forecast with standard errors, we can use the following command:

```
> plot(forecast(mod))
```

Cointegration

The idea behind cointegration, a concept introduced by *Granger (1981)* and formalized by *Engle and Granger (1987)*, is to find a linear combination between non-stationary time series that result in a stationary time series. It is hence possible to detect stable long-run relationships between non-stationary time series (for example, prices).

Cross hedging jet fuel

Airlines are natural buyers of jet fuel. Since the price of jet fuel can be very volatile, most airlines hedge at least part of their exposure to jet fuel price changes. In the absence of liquid jet fuel OTC instruments, airlines use related exchange traded futures contracts (for example, heating oil) for hedging purposes. In the following section, we derive the optimal hedge ratio using first the classical approach of taking into account only the short-term fluctuations between the two prices; afterwards, we improve on the classical hedge ratio by taking into account the long-run stable relationship between the prices as well.

We first load the necessary libraries. The `urca` library has some useful methods for unit root tests and for estimating cointegration relationships.

```
> library("zoo")
> install.packages("urca")
> library("urca")
```

We import the monthly price data for jet fuel and heating oil (in USD per gallon).

```
> prices <- read.zoo("JetFuelHedging.csv", sep = ",",+   FUN =
as.yearmon, format = "%Y-%m", header = TRUE)
```

Taking into account only the short-term behavior (monthly price changes) of the two commodities, one can derive the minimum variance hedge by fitting a linear model that explains changes in jet fuel prices by changes in heating oil prices. The beta coefficient of that regression is the optimal hedge ratio.

```
> simple_mod <- lm(diff(prices$JetFuel) ~
diff(prices$HeatingOil)+0)
```

The function `lm` (for linear model) estimates the coefficients for a best fit of changes in jet fuel prices versus changes in heating oil prices. The `+0` term means that we set the intercept to zero; that is, no cash holdings.

```
> summary(simple_mod)
Call:
lm(formula = diff(prices$JetFuel) ~ diff(prices$HeatingOil) +
    0)
Residuals:
     Min       1Q   Median       3Q      Max
-0.52503 -0.02968  0.00131  0.03237  0.39602

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
diff(prices$HeatingOil)  0.89059    0.03983   22.36   <2e-16
***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0846 on 189 degrees of freedom
Multiple R-squared:  0.7257,   Adjusted R-squared:  0.7242
F-statistic: 499.9 on 1 and 189 DF,  p-value: < 2.2e-16
```

We obtain a hedge ratio of 0.89059 and a residual standard error of 0.0846. The cross hedge is not perfect; the resulting hedged portfolio is still risky.

We now try to improve on this hedge ratio by using an existing long-run relationship between the levels of jet fuel and heating oil futures prices. You can already guess the existence of such a relationship by plotting the two price series (heating oil prices will be in red) using the following command:

```
> plot(prices$JetFuel, main = "Jet Fuel and Heating Oil
Prices",+   xlab = "Date", ylab = "USD")
> lines(prices$HeatingOil, col = "red")
```

We use Engle and Granger's two-step estimation technique. Firstly, both time series are tested for a unit root (non-stationarity) using the augmented Dickey-Fuller test.

```
> jf_adf <- ur.df(prices$JetFuel, type = "drift")
> summary(jf_adf)
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression drift


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max
-1.06212 -0.05015  0.00566  0.07922  0.38086
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.03050    0.02177   1.401  0.16283
z.lag.1     -0.01441    0.01271  -1.134  0.25845
z.diff.lag   0.19471    0.07250   2.686  0.00789 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.159 on 186 degrees of freedom
Multiple R-squared:  0.04099,   Adjusted R-squared:  0.03067
F-statistic: 3.975 on 2 and 186 DF,  p-value: 0.0204


Value of test-statistic is: -1.1335 0.9865

Critical values for test statistics:
      1pct  5pct 10pct
tau2 -3.46 -2.88 -2.57
phi1  6.52  4.63  3.81
```

The null hypothesis of non-stationarity (jet fuel time series contains a unit root) cannot be rejected at the 1% significance level since the test statistic of -1.1335 is not more negative than the critical value of -3.46. The same holds true for heating oil prices (the test statistic is -1.041).

```
> ho_adf <- ur.df(prices$HeatingOil, type = "drift")
> summary(ho_adf)
```

We can now proceed to estimate the static equilibrium model and test the residuals for a stationary time series using an augmented Dickey-Fuller test. Please note that different critical values [for example, from *Engle and Yoo (1987)*] must now be used since the series under investigation is an estimated one.

```
> mod_static <- summary(lm(prices$JetFuel ~
prices$HeatingOil))
> error <- residuals(mod_static)
> error_cadf <- ur.df(error, type = "none")
> summary(error_cadf)
```

The test statistic obtained is -8.912 and the critical value for a sample size of 200 at the 1% level is -4.00; hence we reject the null hypothesis of non-stationarity. We have thus discovered two cointegrated variables and can proceed with the second step; that is, the specification of an **Error-Correction Model** (**ECM**). The ECM represents a dynamic model of how (and how fast) the system moves back to the static equilibrium estimated earlier and is stored in the `mod_static` variable.

```
> djf <- diff(prices$JetFuel)
> dho <- diff(prices$HeatingOil)
> error_lag <- lag(error, k = -1)
> mod_ecm <- lm(djf ~ dho + error_lag)
> summary(mod_ecm)

Call:
lm(formula = djf ~ dho + error_lag + 0)

Residuals:
     Min       1Q   Median       3Q      Max
-0.19158 -0.03246  0.00047  0.02288  0.45117

Coefficients:
          Estimate Std. Error t value Pr(>|t|)
dho        0.90020
0.03238  27.798   <2e-16 ***
error_lag -0.65540    0.06614  -9.909   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06875 on 188 degrees of freedom
Multiple R-squared:  0.8198,   Adjusted R-squared:  0.8179
F-statistic: 427.6 on 2 and 188 DF,  p-value: < 2.2e-16
```

By taking into account the existence of a long-run relationship between jet fuel and heating oil prices (cointegration), the hedge ratio is now slightly higher (0.90020) and the residual standard error significantly lower (0.06875). The coefficient of the error term is negative (-0.65540): large deviations between the two prices are going to be corrected and prices move closer to their long-run stable relationship.

Modeling volatility

As we saw earlier, ARIMA models are used to model the conditional expectation of a process, given its past. For such a process, the conditional variance is constant. Real-world financial time series exhibit, among other characteristics, volatility clustering; that is, periods of relative calm are interrupted by bursts of volatility.

In this section we look at GARCH time series models that can take this stylized fact of real-world (financial) time series into account and apply these models to VaR forecasting.

Volatility forecasting for risk management

Financial institutions measure the risk of their activities using a Value-at-Risk (VaR), usually calculated at the 99% confidence level over a 10 business day horizon. This is the loss that is expected to be exceeded only 1% of the time.

We load the `zoo` library and import monthly return data for Intel Corporation from January 1973 to December 2008.

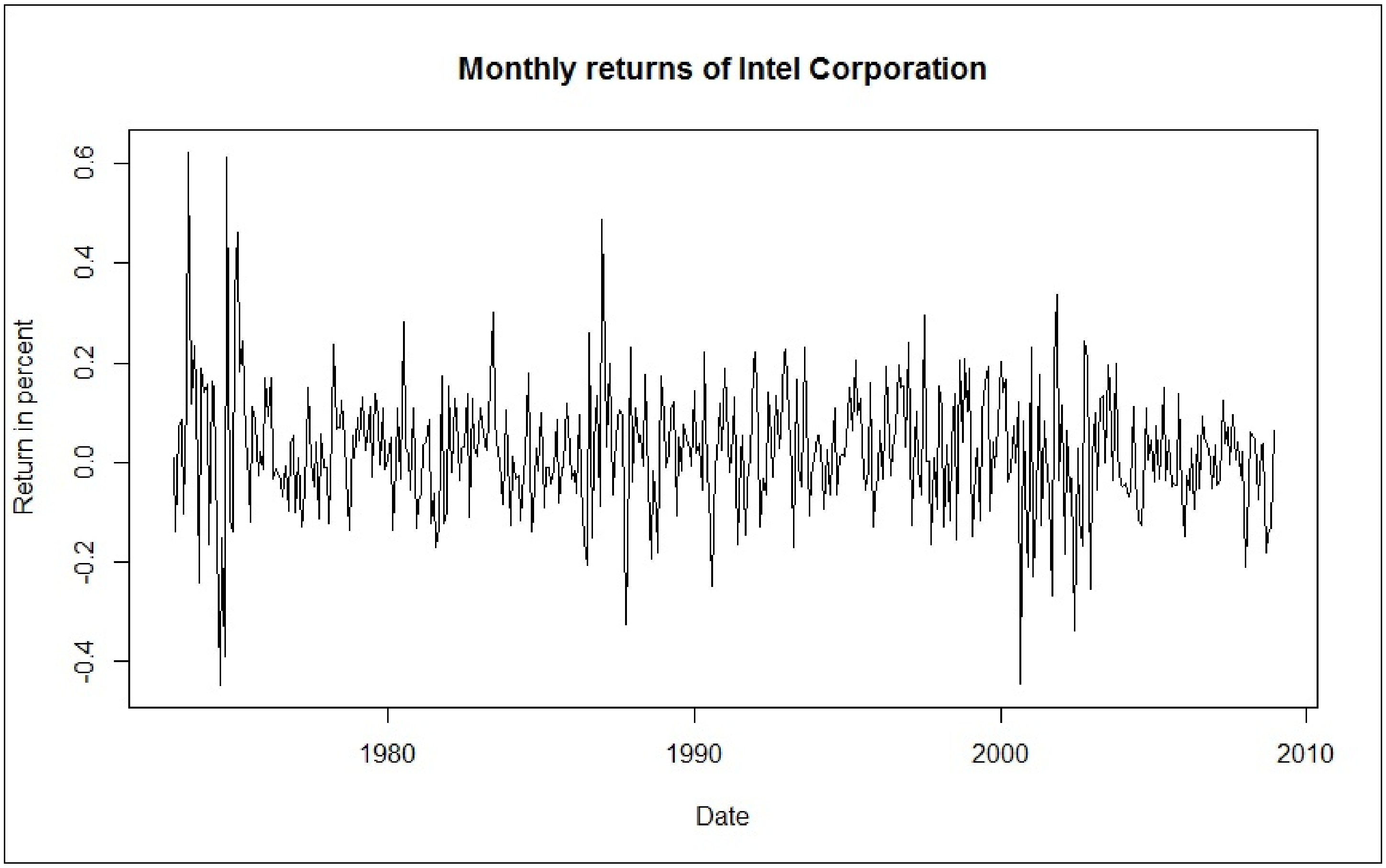
```
> library("zoo")
> intc <- read.zoo("intc.csv", header = TRUE,+   sep = ",",
format = "%Y-%m", FUN = as.yearmon)
```

Testing for ARCH effects

A plot of the returns indicates that ARCH effects might exist in the monthly return data.

```
> plot(intc, main = "Monthly returns of Intel Corporation",+
xlab = "Date", ylab = "Return in percent")
```

The output of the preceding commands is as shown in the following figure:



We can use statistical hypothesis tests to verify our inkling. Two commonly used tests are as follows:

- The Ljung-Box test for autocorrelation in squared returns (as a proxy for volatility)
- The **Lagrange Multiplier** (**LM**) test by *Engle (1982)*

First, we perform the Ljung-Box test on the first 12 lags of the squared returns using the following command:

```
> Box.test(coredata(intc^2), type = "Ljung-Box", lag = 12)

   Box-Ljung test

data:  coredata(intc^2)
X-squared = 79.3451, df = 12, p-value = 5.502e-12
```

We can reject the null hypothesis of no autocorrelations in the squared returns at the 1% significance level. Alternatively, we could employ the LM test from the `FinTS` package, which gives the same result.

```
> install.packages("FinTS")
> library("FinTS")
> ArchTest(coredata(intc))

   ARCH LM-test; Null hypothesis: no ARCH effects

data:  coredata(intc)
Chi-squared = 59.3647, df = 12, p-value = 2.946e-08
```

Both tests confirm that ARCH effects exist in the monthly Intel returns; hence, an ARCH or GARCH model should be employed in modeling the return time series.

GARCH model specification

The most commonly used GARCH model, and one that is usually appropriate for financial time series as well, is a GARCH(1,1) model. We use the functions provided by the `rugarch` library for model specification, parameter estimation, backtesting, and forecasting. If you haven't installed the package, use the following command:

```
> install.packages("rugarch")
```

Afterwards, we can load the library using the following command:

```
> library("rugarch")
```

First, we need to specify the model using the function `ugarchspec`. For a GARCH(1,1) model, we need to set the `garchOrder` to `c(1,1)` and the model for the mean (`mean.model`) should be a white noise process and hence equal to `armaOrder = c(0,0)`.

```
> intc_garch11_spec <- ugarchspec(variance.model = list(+
garchOrder = c(1, 1)),+  mean.model = list(armaOrder = c(0,
0)))
```

GARCH model estimation

The actual fitting of the coefficients by the method of maximum likelihood is done by the function `ugarchfit` using the model specification and the return data as inputs.

```
> intc_garch11_fit <- ugarchfit(spec = intc_garch11_spec,+
data = intc)
```

For additional arguments, see the Help on `ugarchfit`. The output of the fitted model (use the command `intc_garch11_fit`) reveals useful information, such as the values of the optimal parameters, the value of the log-likelihood function, and the information criteria.

Backtesting the risk model

A useful test for checking the model performance is to do a historical backtest. In a risk model backtest, we compare the estimated VaR with the actual return over the period. If the return is more negative than the VaR, we have a VaR exceedance. In our case, a VaR exceedance should only occur in 1% of the cases (since we specified a 99% confidence level).

The function `ugarchroll` performs a historical backtest on the specified GARCH model (here the model is `intc_garch11_spec`). We specify the backtest as follows:

- The return data to be used is stored in the `zoo` object `intc`
- The start period of the backtest (`n.start`) shall be 120 months after the beginning of the series (that is, January 1983)
- The model should be reestimated every month (`refit.every = 1`)
- We use a `moving` window for the estimation
- We use a `hybrid` solver
- We'd like to calculate the VaR (`calculate.VaR = TRUE`) at the 99% VaR tail level (`VaR.alpha = 0.01`)
- We would like to keep the estimated coefficients (`keep.coef = TRUE)`

The following command shows all the preceding points for a backtest:

```
> intc_garch11_roll <- ugarchroll(intc_garch11_spec, intc,+
n.start = 120, refit.every = 1, refit.window = "moving",+
solver = "hybrid", calculate.VaR = TRUE, VaR.alpha = 0.01,+
keep.coef = TRUE)
```

We can examine the backtesting report using the `report` function. By specifying the `type` argument as `VaR`, the function executes the unconditional and conditional coverage tests for exceedances. `VaR.alpha` is the tail probability and `conf.level` is the confidence level on which the conditional coverage hypothesis test will be based.

```
> report(intc_garch11_roll, type = "VaR", VaR.alpha = 0.01,+
conf.level = 0.99)
VaR Backtest Report
===========================================
Model:              sGARCH-norm
Backtest Length:     312
Data:

==========================================
alpha:              1%
Expected Exceed:     3.1
Actual VaR Exceed:      5Actual %:         1.6%
```

```
Unconditional Coverage (Kupiec)
Null-Hypothesis:   Correct Exceedances
LR.uc Statistic:   0.968
LR.uc Critical:      6.635
LR.uc p-value:      0.325
Reject Null:      NO

Conditional Coverage (Christoffersen)
Null-Hypothesis:   Correct Exceedances and
               Independence of Failures
LR.cc Statistic:   1.131
LR.cc Critical:      9.21
LR.cc p-value:      0.568
Reject Null:      O
```

Kupiec's unconditional coverage compares the number of expected versus actual exceedances given the tail probability of VaR, while the Christoffersen test is a joint test of the unconditional coverage and the independence of the exceedances. In our case, despite the actual five exceedances versus an expectation of three, we can't reject the null hypothesis that the exceedances are correct and independent.

A plot of the backtesting performance can also be generated easily. First, create a `zoo` object using the extracted forecasted VaR from the `ugarchroll` object.

```
> intc_VaR <- zoo(intc_garch11_roll@forecast$VaR[, 1])
```

We overwrite the `index` property of the `zoo` object with `rownames` (year and month) from this object as well.

```
> index(intc_VaR) <-
as.yearmon(rownames(intc_garch11_roll@forecast$VaR))
```

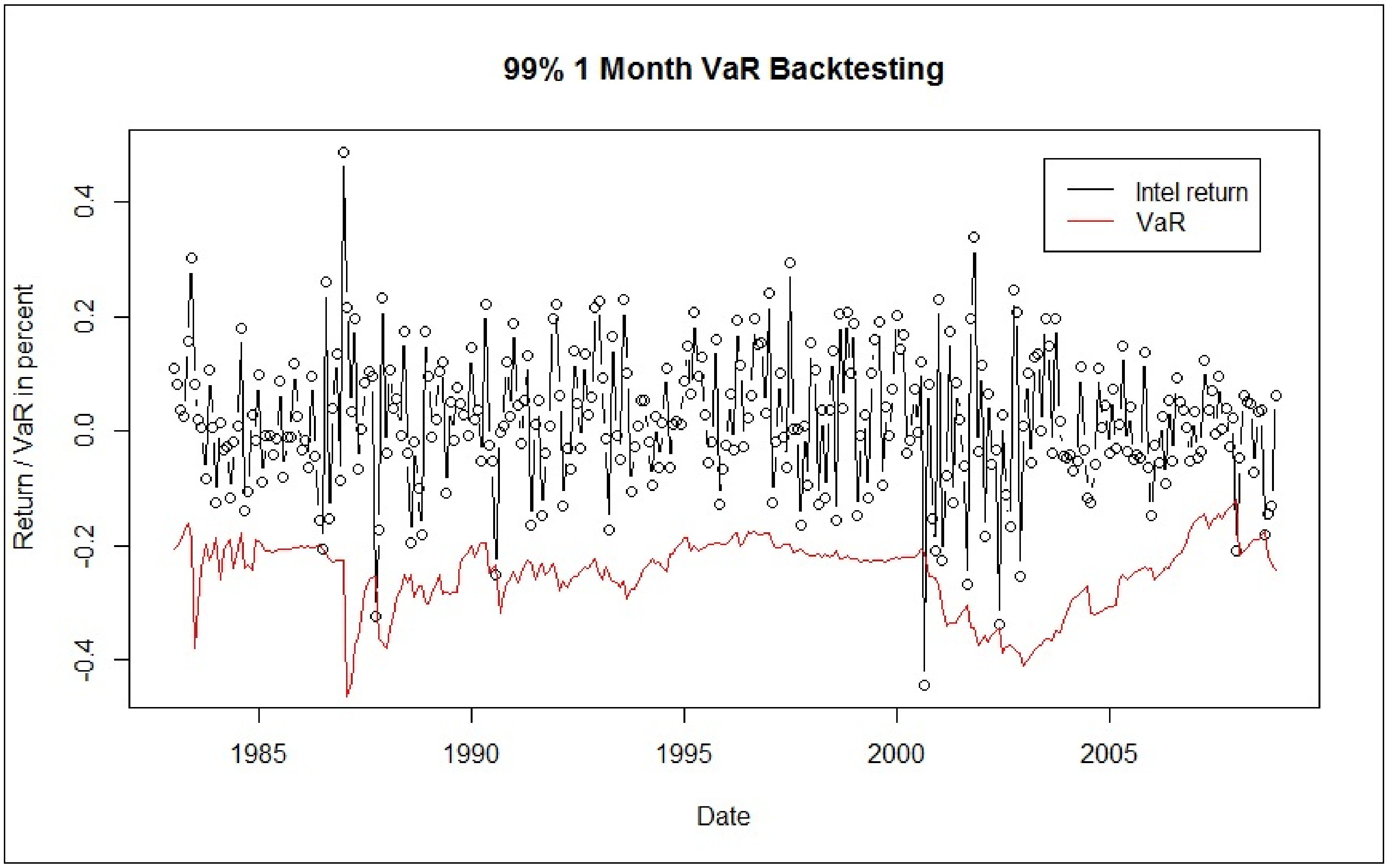
We do the same for the actual returns that are also stored in the `ugarchroll` object.

```
> intc_actual <- zoo(intc_garch11_roll@forecast$VaR[, 2])
> index(intc_actual) <-
as.yearmon(rownames(intc_garch11_roll@forecast$VaR))
```

Now, we are able to plot the VaR versus the actual returns of Intel using the following commands:

```
> plot(intc_actual, type = "b", main = "99% 1 Month VaR
Backtesting",+   xlab = "Date", ylab = "Return/VaR in
percent")
> lines(intc_VaR, col = "red")
> legend("topright", inset=.05, c("Intel return","VaR"), col =
c("black","red"), lty = c(1,1))
```

The following figure shows the output of the preceding command lines:



Forecasting

Now that we can be reasonably sure that our risk model works properly, we can produce VaR forecasts as well. The function `ugarchforecast` takes as arguments the fitted GARCH model (`intc_garch11_fit`) and the number of periods for which a forecast should be produced (`n.ahead = 12`; that is, twelve months).

```
> intc_garch11_fcst <- ugarchforecast(intc_garch11_fit,
n.ahead = 12)
```

The resulting forecast can be expected by querying the forecast object as shown in the following command lines:

```
> intc_garch11_fcst
*------------------------------------*
*       GARCH Model Forecast         *
*------------------------------------*
Model: sGARCH
Horizon: 12
Roll Steps: 0
Out of Sample: 0
```

```
0-roll forecast [T0=Dec 2008]:
      Series  Sigma
T+1  0.01911 0.1168
T+2  0.01911 0.1172
T+3  0.01911 0.1177
T+4  0.01911 0.1181
T+5  0.01911 0.1184
T+6  0.01911 0.1188
T+7  0.01911 0.1191
T+8  0.01911 0.1194
T+9  0.01911 0.1197
T+10 0.01911 0.1200
T+11 0.01911 0.1202
T+12 0.01911 0.1204
```

The one-period ahead forecast for the volatility (sigma) is 0.1168. Since we assume a normal distribution, the 99% VaR can be calculated using the 99% quantile (type in `qnorm(0.99)`) of the standard normal distribution. The one-month 99% VaR estimate for the next period is hence `qnorm(0.99)*0.1168 = 0.2717`. Hence, with 99% probability the monthly return is above -27%.

Summary

In this chapter, we have applied R to selected problems in time series analysis. We covered the different ways of representing time series data, used an ARMA model to forecast house prices, improved our basic minimum variance hedge ratio using a cointegration relationship, and employed a GARCH model for risk management purposes. In the next chapter, you'll learn how you can use R for constructing an optimal portfolio.