Principles of Econometrics with *R*

Constantin Colonescu

2016-09-01

Contents

-

Chapter 1

Introduction

```r
rm(list=ls()) # Caution: this clears the Environment
```

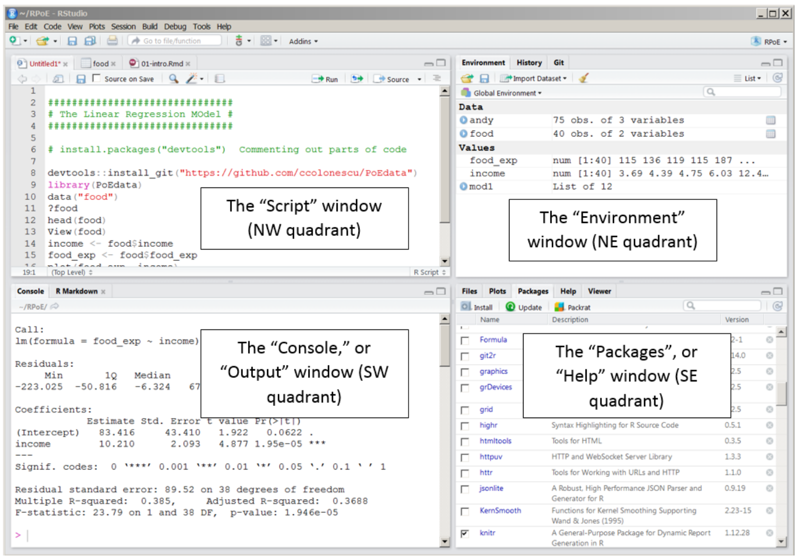
```r
library(bookdown)
library(PoEdata)
library(knitr)
library(xtable)
library(printr)
library(stargazer)
library(rmarkdown)
```

Although this manual is self-contained, it can be used as a supplementary resource for the "Principles of Econometrics" textbook by Carter Hill, William Griffiths and Guay Lim, 4-th edition (Hill, Griffiths, and Lim 2011).

The following list gives some of the R *packages* that are used in this book more frequently:

- `devtools` (Wickham and Chang 2016)
- `PoEdata` (Colonescu 2016)
- `knitr` (Xie 2016b)
- `bookdown` (Xie 2016a)
- `xtable` (Dahl 2016)
- `printr` (Xie 2014)
- `stargazer` (Hlavac 2015)
- `rmarkdown` (Allaire et al. 2016)

The function *install_git* from the package `devtools` installs packages such as `PoEdata` from the GitHub web site. Here is the code that installs `devtools` and `bookdown`:


~/RPoE - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
RPoE
Untitled1*
food
01-intro.Rmd
Source on Save
Run
Source
19:1
(Top Level)
R Script
Environment History Git
Import Dataset
List
Global Environment
Console R Markdown
~/RPoE/
Files Plots Packages Help Viewer
Install Update Packrat
The "Script" window (NW quadrant)
The "Environment" window (NE quadrant)
The "Console," or "Output" window (SW quadrant)
The "Packages", or "Help" window (SE quadrant)

```
##############################
# The Linear Regression MOdel #
##############################

# install.packages("devtools")  Commenting out parts of code

devtools::install_git("https://github.com/ccolonescu/PoEdata")
library(PoEdata)
data("food")
?food
head(food)
View(food)
income <- food$income
food_exp <- food$food_exp
```

```
Data
andy        75 obs. of 3 variables
food        40 obs. of 2 variables
Values
food_exp    num [1:40] 115 136 119 115 187 ...
income      num [1:40] 3.69 4.39 4.75 6.03 12.4…
mod1        List of 12
```

```
Call:
lm(formula = food_exp ~ income)

Residuals:
     Min       1Q   Median
-223.025  -50.816   -6.324    67

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   83.416     43.410   1.922   0.0622 .
income        10.210      2.093   4.877 1.95e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 89.52 on 38 degrees of freedom
Multiple R-squared:  0.385,	Adjusted R-squared:  0.3688
F-statistic: 23.79 on 1 and 38 DF,  p-value: 1.946e-05

> 
```

	Name	Description	Version
☐	Formula		2-1
☐	git2r		14.0
☐	graphics		2.5
☐	grDevices		2.5
☐	grid		2.5
☐	highr	Syntax Highlighting for R Source Code	0.5.1
☐	htmltools	Tools for HTML	0.3.5
☐	httpuv	HTTP and WebSocket Server Library	1.3.3
☐	httr	Tools for Working with URLs and HTTP	1.1.0
☐	jsonlite	A Robust, High Performance JSON Parser and Generator for R	0.9.19
☐	KernSmooth	Functions for Kernel Smoothing Supporting Wand & Jones (1995)	2.23-15
✓	knitr	A General-Purpose Package for Dynamic Report Generation in R	1.12.28

Figure 1.1: The four quadrants of an RStudio screen

```r
install.packages("devtools")
devtools::install_git(
  "https://github.com/ccolonescu/PoEdata")
```

The computing environment for using R (R Development Core Team 2008) is RStudio (RStudio Team 2015). You need to install on your computer the following resources:

- R (https://cloud.r-project.org/)
- RStudio (https://www.rstudio.com/products/rstudio/download/)
- PoEdata package (https://github.com/ccolonescu/PoEdata)

This brief introduction to R does not intend to be exhaustive, but to cover the minimum material used in this book. Please refer to the R documentation and to many other resources for additional information. For beginners, I would recommend (Lander 2013).

1.1 The RStudio Screen

A typical RStudio Screen is divided in four quadrants, as Figure 1.1 shows. The NW quadrant is for writing your script and for viewing data.

```r
knitr::include_graphics(
  "01-intro_files/RStudio_screen_1.PNG")
```

1.1.1 The `Script`, or `data view` window

Here are a few tips for writing and executing script in the `Script` window:

- You may start your script with a comment showing a title and a brief description of what the script does. A "comment" line starts with the hash character (`#`). Comments can be inserted anywhere in the script, even in line with code, but what follows the hash character to the end of the line will be disregarded by R.
- Code lines may be continued on the next line with no special character to announce a line continuation. However, code will be continued on the next line only if the previous line ends in a way that requires continuation, for instance with a comma or unclosed brackets.
- When you want to run a certain line of code, place the cursor anywhere on the line and press `Ctrl+Enter`; if you want to run a sequence of several code lines, select the respective sequence and press `Ctrl+Enter`.

1.1.2 The `console`, or `output` window

While it is always advisable to work in the script mode because it can be saved and re-used for different data, sometimes we need to run commands that are out of the script context. Such commands can be typed at the bottom of the console at the sign `>`. Pressing `Enter` executes such a command, and the up and down arrows allow re-activating and editing older lines of code that had been previously typed into the console.

1.2 How to Open a Data File

To open a data file for the Principles of Econometrics textbook, (Hill, Griffiths, and Lim 2011), first check if the `devtools` package is installed. If it is not, run the code `install.packages("devtools")` in the console.

```r
devtools::install_git(
  "https://github.com/ccolonescu/PoEdata")
library(PoEdata) # Makes datasets ready to use
```

Now, we can load and inspect a particular dataset, for example "andy." When the dataset is available, it sould show in the `Environment` window (look up and right).

```r
library(PoEdata)
data("andy")  # makes the dataset "andy" ready to use
?andy         # shows information about the dataset
```

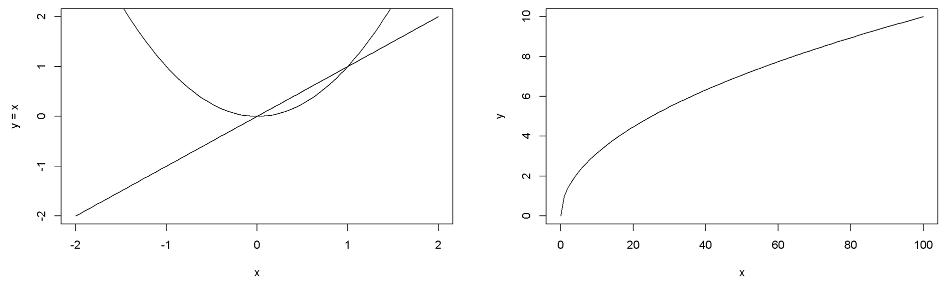


Figure 1.2: Examples of using the function curve()

```r
# Show head of dataset, with variables as column names:
head(andy)

# Show a few rows in dataset:
some(andy)
```

1.3 Creating Graphs

The basic tools for graph creating are the following R functions

- `plot(x, y, xlab="income in 100", ylab="food expenditure, in $", type="p")`, where `x` and `y` stand for the variable names to be plotted, `xlab` and `ylab` are the labels you wish to see on the plot, and `type` refers to the style of the plot; `type` can be one of the following: "p" (points), "l" for lines, and "b" for both points and lines, "n" for no plot. The type value "n" creates an empty graph which seres other functions such as `abline()`, which is described below.
- The function `curve()` plots a curve described by a mathematical function, say f, over a specified interval . When the argument `add = TRUE` is present, the fuction adds the curve to a previously plotted graph. Figure 1.2 is an example.

```r
curve(x^1, from=-2, to=2, xlab="x", ylab="y = x" )
# Add another curve to the existing graph:
curve(x^2, add = TRUE)
#plot(1:100, type='n')
curve(sqrt(x), from=0, to=100, xlab="x", ylab="y")
```

- The function `abline()` adds a line defined by its intercept `a` and slope `b` to the current graph. The arguments of the function are: besides `a` and `b`, the arguments of the function are: `h`, the y-value for a horizontal line; `v`, the

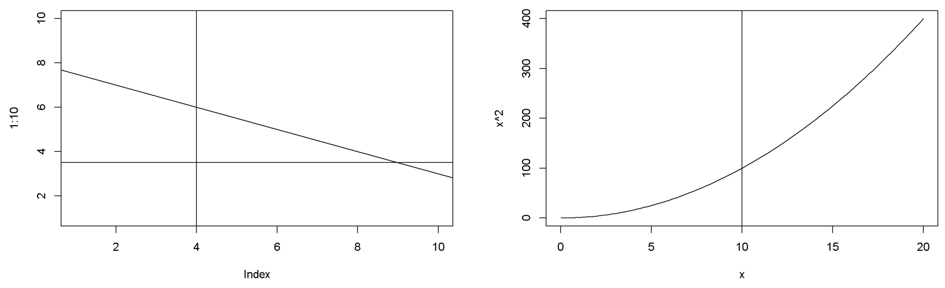


Figure 1.3: Examples of using the function ‘abline()‘

x-value for a vertical line; `coef`, the name of a simple linear regression object, which includes the intercept and slope of a regression line.

```r
plot(1:10, type="n") # creates an empty graph
# Add straight lines to graph:
abline(a=8, b=-0.5, h=3.5, v=4)
curve(x^2, from=0, to=20)
abline(v=10)
```

1.4 An R Cheat Sheet

Here is an overview of some R commands used in this book.

`lm(y~x, data = datafile)` regresses y on x using the data in *datafile*

`nrow(datafile)` returns the number of observations (raws) in *datafile*

`nobs(modelname)` gives the number of observations used by a model. This may be different from the number of observation in the data file because of missing values or sub-sampling

`set.seed(number)` sets the seed for the random number generator to make results reproducible. This is needed to construct random subsamples of data

`rm(list=ls())` removes all objects in the current `Environment` except those that have names starting with a dot (.)

Chapter 2

The Simple Linear Regression Model

```r
rm(list=ls()) # Caution: this clears the Environment
```

2.1 The General Model

A simple linear regression model assumes that a linear relationship exists between the conditional expectation of a dependent variable, y, and an independent variable, x. Sometimes I call the independent variable 'response' or 'response variable', and the independent variables 'regressors.' The assumed relationship in a linear regression model has the form

$$y_i = \beta_1 + \beta_2 x_i + e_i, \tag{2.1}$$

where

- y is the *dependent variable*
- x is the *independent variable*
- e is an *error term*
- σ^2 is the variance of the error term
- β_1 is the *intercept* parameter or coefficient
- β_2 is the *slope* parameter or coefficient
- i stands fot the i -th observation in the dataset, $i = 1, 2, ..., N$
- N is the number of observations in the dataset

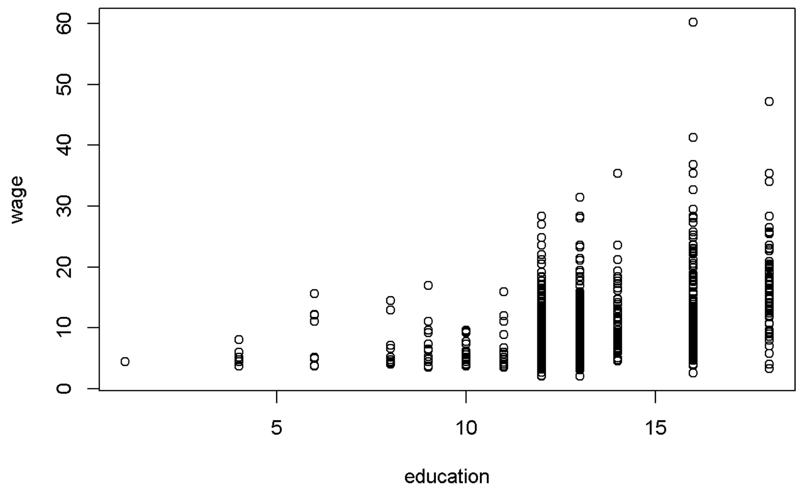


Figure 2.1: Example of several observations for any given x

The *predicted*, or estimated value of y given x is given by Equation 2.2; in general, the *hat* symbol indicates an estimated or a predicted value.

$$\hat{y} = b_1 + b_2 x \tag{2.2}$$

The simple linear regression model assumes that the values of x are previously chosen (therefore, they are non-random), that the variance of the error term, σ^2, is the same for all values of x, and that there is no connection between one observation and another (no correlation between the error terms of two observations). In addition, it is assumed that the expected value of the error term for any value of x is zero.

The subscript i in Equation 2.1 indicates that the relationship applies to each of the N observations. Thus, there must be specific values of y, x, and e for each observation. However, since x is not random, there are, typically, several observations sharing the same x, as the scatter diagram in Figure 2.1 shows.

```r
library(PoEdata)
data("cps_small")
plot(cps_small$educ, cps_small$wage,
     xlab="education", ylab="wage")
```

2.2 Example: Food Expenditure versus Income

The data for this example is stored in the R package `PoEdata` (To check if the package `PoEdata` is installed, look in the `Packages` list.)

```
library(PoEdata)
data(food)
head(food)
```

food_exp	income
115.22	3.69
135.98	4.39
119.34	4.75
114.96	6.03
187.05	12.47
243.92	12.98

It is always a good idea to visually inspect the data in a **scatter diagram**, which can be created using the function `plot()`. Figure 2.2 is a scatter diagram of food expenditure on income, suggesting that there is a positive relationship between income and food expenditure.

```
data("food", package="PoEdata")
plot(food$income, food$food_exp,
     ylim=c(0, max(food$food_exp)),
     xlim=c(0, max(food$income)),
     xlab="weekly income in $100",
     ylab="weekly food expenditure in $",
     type = "p")
```

2.3 Estimating a Linear Regression

The R function for estimating a linear regression model is `lm(y~x, data)` which, used just by itself does not show any output; It is useful to give the model a name, such as `mod1`, then show the results using `summary(mod1)`. If you are interested in only some of the results of the regression, such as the estimated coefficients, you can retrieve them using specific functions, such as the function `coef()`. For the food expenditure data, the regression model will be

$$food\_exp = \beta_1 + \beta_2 income + e \tag{2.3}$$

where the subscript i has been omitted for simplicity.

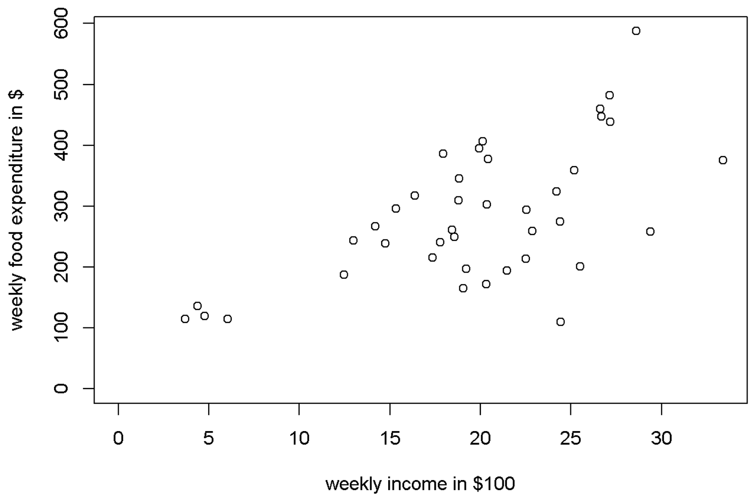


Figure 2.2: A scatter diagram for the food expenditure model

```
library(PoEdata)
mod1 <- lm(food_exp ~ income, data = food)
b1 <- coef(mod1)[[1]]
b2 <- coef(mod1)[[2]]
smod1 <- summary(mod1)
smod1
```

```
##
## Call:
## lm(formula = food_exp ~ income, data = food)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -223.03  -50.82   -6.32   67.88  212.04
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    83.42      43.41    1.92    0.062 .
## income         10.21       2.09    4.88 0.000019 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

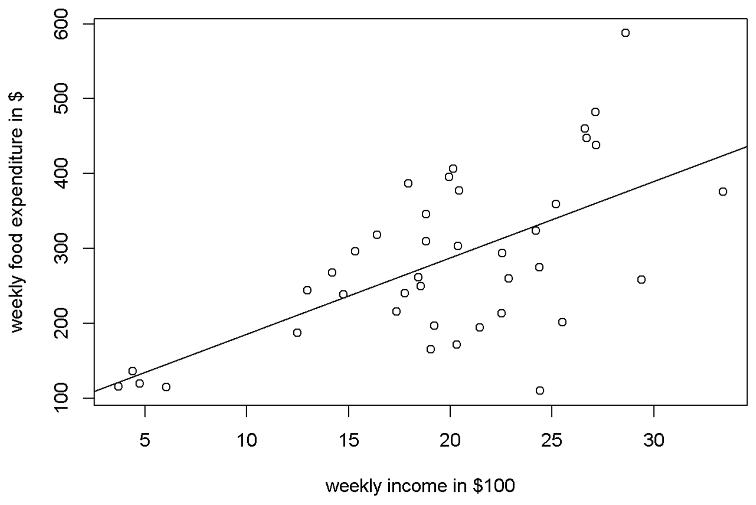


Figure 2.3: Scatter diagram and regression line for the food expenditure model

```
## Residual standard error: 89.5 on 38 degrees of freedom
## Multiple R-squared:  0.385,  Adjusted R-squared:  0.369
## F-statistic: 23.8 on 1 and 38 DF,  p-value: 0.0000195
```

The function `coef()` returns a list containing the estimated coefficients, where a specific coefficient can be accessed by its position in the list. For example, the estimated value of β_1 is `b1 <- coef(mod1)[[1]]`, which is equal to 83.416002, and the estimated value of β_2 is `b2 <- coef(mod1)[[2]]`, which is equal to 10.209643.

The intercept parameter, β_1, is usually of little importance in econometric models; we are mostly interested in the slope parameter, β_2. The estimated value of β_2 suggests that the food expenditure for an average family increases by 10.209643 when the family income increases by 1 unit, which in this case is \$100. The *R* function `abline()` adds the regfression line to the prevoiusly plotted scatter diagram, as Figure 2.3 shows.

```
plot(food$income, food$food_exp,
     xlab="weekly income in $100",
     ylab="weekly food expenditure in $",
     type = "p")
abline(b1,b2)
```

How can one retrieve various regression results? These results exist in two R *objects* produced by the `lm()` function: the regression object, such as `mod1` in the above code sequence, and the regression summary, which I denoted by `smod1`. The next code shows how to list the names of all results in each object.

```
names(mod1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
names(smod1)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

To retrieve a particular result you just refer to it with the name of the object, followed by the `$` sign and the name of the result you wish to retrieve. For instance, if we want the vector of coefficients from `mod1`, we refer to it as `mod1$coefficients` and `smod1$coefficients`:

```
mod1$coefficients
```

```
## (Intercept)      income
##     83.4160     10.2096
```

```
smod1$coefficients
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	83.4160	43.41016	1.92158	0.062182
income	10.2096	2.09326	4.87738	0.000019

As we have seen before, however, some of these results can be retrieved using specific functions, such as `coef(mod1)`, `resid(mod1)`, `fitted(mod1)`, and `vcov(mod1)`.

2.4 Prediction with the Linear Regression Model

The estimated regression parameters, b_1 and b_2 allow us to *predict* the expected food expenditure for any given income. All we need to do is to plug the estimated parameter values and the given income into an equation like Equation 2.2. For example, the expected value of $food\_exp$ for an income of \$2000 is calculated in Equation 2.4. (Remember to divide the income by 100, since the data for the variable *income* is in hundreds of dollars.)

$$\widehat{food\_exp} = 83.416002 + 10.209643 * 20 = \$287.608861 \tag{2.4}$$

R, however, does this calculations for us with its function called `predict()`. Let us extend slightly the example to more than one income for which we predict food expenditure, say *income* = \$2000, \$2500, and \$2700. The function `predict()` in R requires that the new values of the independent variables be organized under a particular form, called a data frame. Even when we only want to predict for one income, we need the same data-frame structure. In R, a set of numbers is held together using the structure `c()`. The following sequence shows this example.

```
# library(PoEdata) (load the data package if you have not done so yet)
mod1 <- lm(food_exp~income, data=food)
newx <- data.frame(income = c(20, 25, 27))
yhat <- predict(mod1, newx)
names(yhat) <- c("income=$2000", "$2500", "$2700")
yhat  # prints the result
```

```
## income=$2000        $2500        $2700
##      287.609      338.657      359.076
```

2.5 Repeated Samples to Assess Regression Coefficients

The regression coefficients b_1 and b_2 are random variables, because they depend on sample. Let us construct a number of random subsamples from the *food* data and re-calculate b_1 and b_2. A random subsample can be constructed using the function `sample()`, as the following example illustrates only for b_2.

```
N <- nrow(food) # returns the number of observations in the dataset
C <- 50         # desired number of subsamples
S <- 38         # desired sample size

sumb2 <- 0
for (i in 1:C){   # a loop over the number of subsamples
  set.seed(3*i)   # a different seed for each subsample
  subsample <- food[sample(1:N, size=S, replace=TRUE), ]
  mod2 <- lm(food_exp~income, data=subsample)
  #sum b2 for all subsamples:
  sumb2 <- sumb2 + coef(mod2)[[2]]
}
print(sumb2/C, digits = 3)
```

```
## [1] 9.88
```

The result, $b_2 = 9.88$, is the average of 50 estimates of b_2.

2.6 Estimated Variances and Covariance of Regression Coefficients

Many applications require estimates of the variances and covariances of the regression coefficients. R stores them in the a matrix `vcov()`:

```
(varb1 <- vcov(mod1)[1, 1])
```

```
## [1] 1884.44
```

```
(varb2 <- vcov(mod1)[2, 2])
```

```
## [1] 4.38175
```

```
(covb1b2 <- vcov(mod1)[1,2])
```

```
## [1] -85.9032
```

2.7 Non-Linear Relationships

Sometimes the scatter plot diagram or some theoretical consideraions suggest a non-linear relationship. The most popular non-linear relationships involve logarithms of the dependent or independent variables and polinomial functions.

The *quadratic* model requires the square of the independent variable.

$$y_i = \beta_1 + \beta_2 x_i^2 + e_i \tag{2.5}$$

In R, independent variables involving mathematical operators can be included in a regression equation with the function `I()`. The following example uses the dataset `br` from the package `PoEdata`, which includes the sale prices and the attributes of 1080 houses in Baton Rouge, LA. `price` is the sale price in dollars, and `sqft` is the surface area in square feet.

```
library(PoEdata)
data(br)
mod3 <- lm(price~I(sqft^2), data=br)
b1 <- coef(mod3)[[1]]
b2 <- coef(mod3)[[2]]
sqftx=c(2000, 4000, 6000) #given values for sqft
pricex=b1+b2*sqftx^2 #prices corresponding to given sqft
DpriceDsqft <- 2*b2*sqftx # marginal effect of sqft on price
```

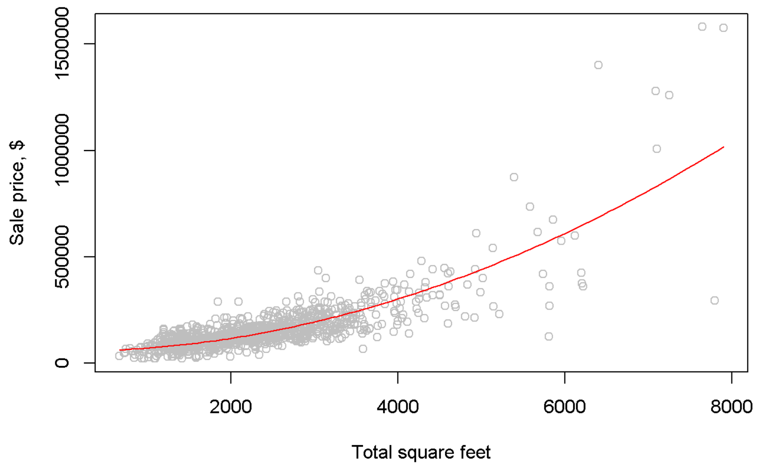


Figure 2.4: Fitting a quadratic model to the 'br' dataset

```
elasticity=DpriceDsqft*sqftx/pricex
b1; b2; DpriceDsqft; elasticity #prints results
```

```
## [1] 55776.6

## [1] 0.0154213

## [1]  61.6852 123.3704 185.0556

## [1] 1.05030 1.63125 1.81741
```

We woud like now to draw a scatter diagram and see how the quadratic function fits the data. The next chunk of code provides two alternatives for constructing such a graph. The first simply draws the quadratic function on the scatter diagram, using the R function `curve()`; the second uses the function `lines`, which requires ordering the dataset in increasing values of `sqft` before the regression model is evaluated, such that the resulting fitted values will also come out in the same order.

```
mod31 <- lm(price~I(sqft^2), data=br)
plot(br$sqft, br$price, xlab="Total square feet",
     ylab="Sale price, $", col="grey")
#add the quadratic curve to the scatter plot:
curve(b1+b2*x^2, col="red", add=TRUE)
```

An alternative way to draw the fitted curve:

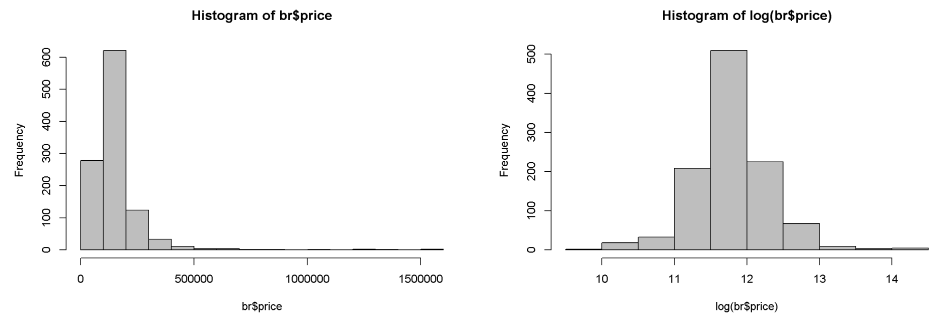


Figure 2.5: A comparison between the histograms of 'price' and 'log(price)'

```r
ordat <- br[order(br$sqft), ] #sorts the dataset after `sqft`
mod31 <- lm(price~I(sqft^2), data=ordat)
plot(br$sqft, br$price,
     main="Dataset ordered after 'sqft' ",
     xlab="Total square feet",
     ylab="Sale price, $", col="grey")
lines(fitted(mod31)~ordat$sqft, col="red")
```

The *log-linear* model regresses the log of the dependent variable on a linear expression of the independent variable (unless otherwise specified, the `log` notation stands for natural logarithm, following a usual convention in economics):

$$log(y_i) = \beta_1 + \beta_2 x_i + e_i \tag{2.6}$$

One of the reasons to use the *log* of an independent variable is to make its distribution closer to the normal distribution. Let us draw the histograms of `price` and `log(price)` to compare them (see Figure 2.5). It can be noticed that that the `log` is closer to the normal distribution.

```r
hist(br$price, col='grey')
hist(log(br$price), col='grey')
```

We are interested, as before, in the estimates of the coefficients and their interpretation, in the fitted values of `price`, and in the marginal effect of an increase in `sqft` on `price`.

```r
library(PoEdata)
data("br")
mod4 <- lm(log(price)~sqft, data=br)
```

The coefficients are $b_1 = 10.84$ and $b_2 = 0.00041$, showing that an increase in the surface area (`sqft`) of an apartment by one unit (1 sqft) increases the price of the apartment by 0.041 percent. Thus, for a house price of \$100,000, an increase of 100 sqft will increase the price by approximately $100 * 0.041$ percent, which is equal to \$4112.7. In general, the marginal effect of an increase in x on y in Equation 2.6 is

$$\frac{dy}{dx} = \beta_2 y, \tag{2.7}$$

and the elasticity is

$$\epsilon = \frac{dy}{dx}\frac{x}{y} = \beta_2 x. \tag{2.8}$$

The next lines of code show how to draw the fitted values curve of the loglinear model and how to calculate the marginal effect and the elasticity for the median price in the dataset. The fitted values are here calculated using the formula

$$\hat{y} = e^{b_1 + b_2 x} \tag{2.9}$$

```
ordat <- br[order(br$sqft), ] #order the dataset
mod4 <- lm(log(price)~sqft, data=ordat)
plot(br$sqft, br$price, col="grey")
lines(exp(fitted(mod4))~ordat$sqft,
      col="blue", main="Log-linear Model")
```

```
pricex<- median(br$price)
sqftx <- (log(pricex)-coef(mod4)[[1]])/coef(mod4)[[2]]
(DyDx <- pricex*coef(mod4)[[2]])
```

```
## [1] 53.465
```

```
(elasticity <- sqftx*coef(mod4)[[2]])
```

```
## [1] 0.936693
```

R allows us to calculate the same quantities for several *(sqft, price)* pairs at a time, as shown in the following sequence:

```
b1 <- coef(mod4)[[1]]
b2 <- coef(mod4)[[2]]
#pick a few values for sqft:
sqftx <- c(2000, 3000, 4000)
#estimate prices for those and add one more:
```

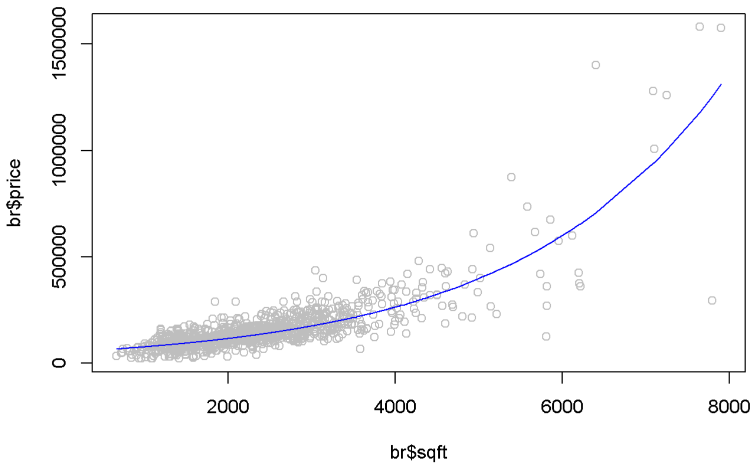


Figure 2.6: The fitted value curve in the log-linear model

```
pricex <- c(100000, exp(b1+b2*sqftx))
#re-calculate sqft for all prices:
sqftx <- (log(pricex)-b1)/b2
#calculate and print elasticities:
(elasticities <- b2*sqftx)
```

```
## [1] 0.674329 0.822538 1.233807 1.645075
```

2.8 Using Indicator Variables in a Regression

An indicator, or binary variable marks the presence or the absence of some attribute of the observational unit, such as gender or race if the observational unit is an individual, or location if the observational unit is a house. In the dataset `utown`, the variable `utown` is 1 if a house is close to the university and 0 otherwise. Here is a simple linear regression model that involves the variable `utown`:

$$price_i = \beta_1 + \beta_2 utown_i \tag{2.10}$$

The coefficient of such a variable in a simple linear model is equal to the difference between the average prices of the two categories; the intercept coefficient of the model in Equation 2.10 is equal to the average price of the houses that are not close to university. Let us first calculate the average prices for each category, wich are denoted in the following sequence of code `price0bar` and `price1bar`:

```
data(utown)
price0bar <- mean(utown$price[which(utown$utown==0)])
price1bar <- mean(utown$price[which(utown$utown==1)])
```

The results are: $\overline{price} = 277.24$ close to university, and $\overline{price} = 215.73$ for those not close. I now show that the same results yield the coefficients of the regression model in Equation 2.10:

```
mod5 <- lm(price~utown, data=utown)
b1 <- coef(mod5)[[1]]
b2 <- coef(mod5)[[2]]
```

The results are: $\overline{price} = b_1 = 215.73$ for non-university houses, and $\overline{price} = b_1 + b_2 = 277.24$ for university houses.

2.9 Monte Carlo Simulation

A Monte Carlo simulation generates random values for the dependent variable when the regression coefficients and the distribution of the random term are given. The following example seeks to determine the distribution of the independent variable in the food expenditure model in Equation 2.3.

```
N <- 40
x1 <- 10
x2 <- 20
b1 <- 100
b2 <- 10
mu <- 0
sig2e <- 2500
sde <- sqrt(sig2e)
yhat1 <- b1+b2*x1
yhat2 <- b1+b2*x2
curve(dnorm(x, mean=yhat1, sd=sde), 0, 500, col="blue")
curve(dnorm(x, yhat2, sde), 0,500, add=TRUE, col="red")
abline(v=yhat1, col="blue", lty=2)
abline(v=yhat2, col="red", lty=2)
legend("topright", legend=c("f(y|x=10)",
                            "f(y|x=20)"), lty=1,
       col=c("blue", "red"))
```

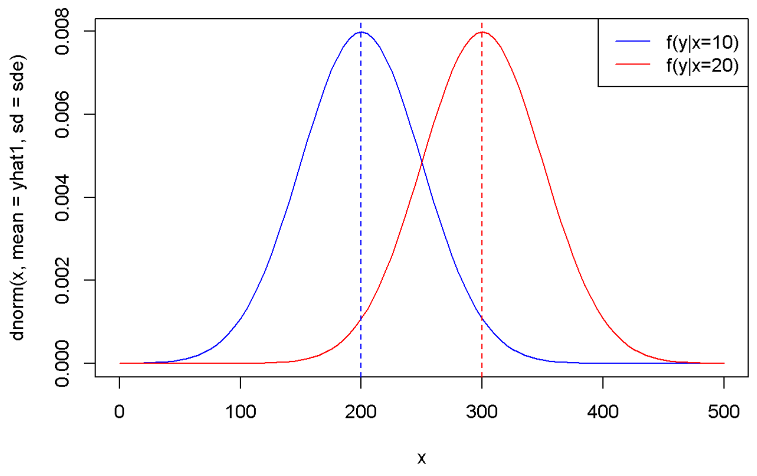


Figure 2.7: The theoretical (true) probability distributions of food expenditure, given two levels of income

Next, we calculate the variance of b_2 and plot the corresponding density function.

$$var(b_2) = \frac{\sigma^2}{\sum(x_i - \bar{x})} \tag{2.11}$$

```
x <- c(rep(x1, N/2), rep(x2,N/2))
xbar <- mean(x)
sumx2 <- sum((x-xbar)^2)
varb2 <- sig2e/sumx2
sdb2 <- sqrt(varb2)
leftlim <- b2-3*sdb2
rightlim <- b2+3*sdb2
curve(dnorm(x, mean=b2, sd=sdb2), leftlim, rightlim)
abline(v=b2, lty=2)
```

Now, with the same values of b_1, b_2, and error standard deviation, we can generate a set of values for y, regress y on x, and calculate an estimated values for the coefficient b_2 and its standard error.

```
set.seed(12345)
y <- b1+b2*x+rnorm(N, mean=0, sd=sde)
mod6 <- lm(y~x)
b1hat <- coef(mod6)[[1]]
```

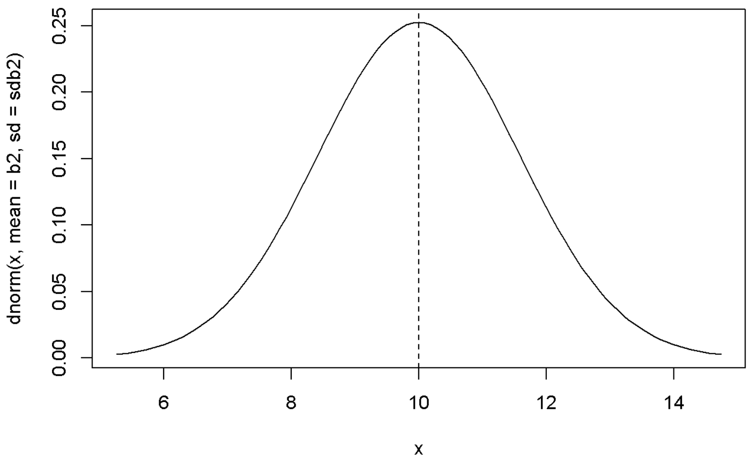


Figure 2.8: The theoretical (true) probability density function of b_2

```
b2hat <- coef(mod6)[[2]]
mod6summary <- summary(mod6) #the summary contains the standard errors
seb2hat <- coef(mod6summary)[2,2]
```

The results are $b_2 = 11.64$ and $se(b_2) = 1.64$. The strength of a Monte Carlo simulation is, however, the possibility of repeating the estimation of the regression parameters for a large number of automatically generated samples. Thus, we can obtain a large number of values for a parameter, say b_2, and then determine its sampling characteristics. For instance, if the mean of these values is close to the initially asumed value $b_2 = 10$, we conclude that our estimator (the method of estimating the parameter) is unbiased.

We are going to use this time the values of x in the *food* dataset, and generate y using the linear model with $b_1 = 100$ and $b_2 = 10$.

```
data("food")
N <- 40
sde <- 50
x <- food$income
nrsim <- 1000
b1 <- 100
b2 <- 10
vb2 <- numeric(nrsim) #stores the estimates of b2
for (i in 1:nrsim){
```

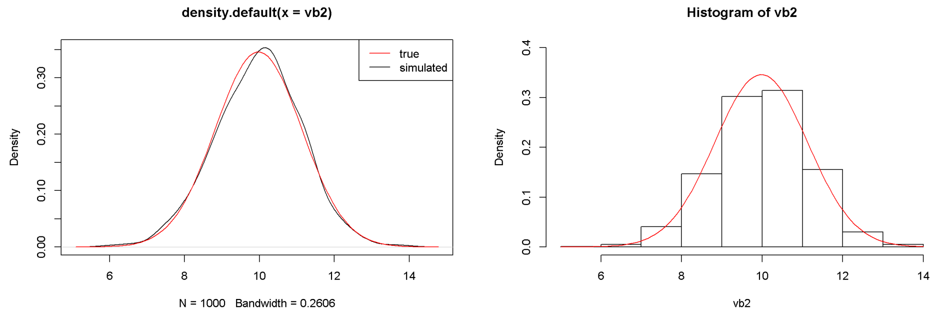


Figure 2.9: The simulated and theoretical distributions of b_2

```
  set.seed(12345+10*i)
  y <- b1+b2*x+rnorm(N, mean=0, sd=sde)
  mod7 <- lm(y~x)
  vb2[i] <- coef(mod7)[[2]]
}
mb2 <- mean(vb2)
seb2 <- sd(vb2)
```

The mean and standard deviation of the estimated 40 values of b_2 are, respectively, 9.974985 and 1.152632. Figure 2.9 shows the simulated distribution of b_2 and the theoretical one.

```
plot(density(vb2))
curve(dnorm(x, mb2, seb2), col="red", add=TRUE)
legend("topright", legend=c("true", "simulated"),
       lty=1, col=c("red", "black"))
hist(vb2, prob=TRUE, ylim=c(0,.4))
curve(dnorm(x, mean=mb2, sd=seb2), col="red", add=TRUE)
```

```
rm(list=ls()) # Caution: this clears the Environment
```

Chapter 3

Interval Estimation and Hypothesis Testing

```r
library(xtable)
library(PoEdata)
library(knitr)
```

So far we estimated only a number for a regression parameter such as β_2. This estimate, however, gives no indication of its reliablity, since it is just a realization of the random variable b_2. An *interval estimate*, which is also known as a *confidence interval* is an interval centerd on an estimated value, which includes the true parameter with a given probability, say 95%. A coefficient of the linear regression model such as b_2 is normally distributed with its mean equal to the population parameter β_2 and a variance that depends on the population variance σ^2 and the sample size:

$$b_2 \sim N\left(\beta_2,\ \frac{\sigma^2}{\sum(x_i - \bar{x})^2}\right), \tag{3.1}$$

3.1 The Estimated Distribution of Regression Coefficients

Equation 3.1 gives the theoretical distribution of a linear regression coefficient, a distribution that is not very useful since it requires the unknown population variance σ^2. If we replace σ^2 with an estimated variance $\hat{\sigma}^2$ given in Equation 3.2, the standardized distribution of b_2 becomes a t distribution with $N - 2$ degrees of freedom.

$$\hat{\sigma}^2 = \frac{\sum \hat{e}_i^2}{N-2} \tag{3.2}$$

Equation 3.3 shows the the *t-ratio*:

$$t = \frac{b_2 - \beta_2}{se(b_2)} \tag{3.3}$$

3.2 Confidence Interval in General

An interval estimate of b_2 based on the *t-ratio* is calculated in Equation 3.4, which we can consider as "an interval that includes the true parameter β_2 with a probability of $100(1-\alpha)\%$." In this context, α is called *significance level*, and the interval is called, for example, *a 95% confidence interval estimate for* β_2. The critical value of the t-ratio, t_c, depends on the chosen significance level and on the number of degrees of freedom. In *R*, the function that returns critical values for the t distribution is $qt(1-\frac{\alpha}{2}, df)$, where df is the number of degrees of freedom.

$$b_2 \pm t_c \times se(b_2) \tag{3.4}$$

A side note about using distributions in *R*. There are four types of functions related to distributions, each type's name beginning with one of the following four letters: `p` for the cummulative distribution function, `d` for density, `r` for a draw of a random number from the respective distribution, and `q` for quantile. This first letter is followed by a few letters suggesting what distribution we refer to, such as `norm`, `t`, `f`, and `chisq`. Now, if we put toghether the first letter and the distribution name, we get functions such as the following, where `x` and `q` stand for quantiles, `p` stands for probability, `df` is degree of freedom (of which F has two), `n` is the desired number of draws, and `lower.tail` can be TRUE (default) if probabilities are $P[X \leq x]$ or FALSE if probabilities are $P[X > x]$:

- For the uniform distribution:
  - dunif(x, min = 0, max = 1)
  - punif(q, min = 0, max = 1, lower.tail = TRUE)
  - qunif(p, min = 0, max = 1, lower.tail = TRUE)
  - runif(n, min = 0, max = 1)
- For the normal distribution:
  - dnorm(x, mean = 0, sd = 1)
  - pnorm(q, mean = 0, sd = 1, lower.tail = TRUE)

  - qnorm(p, mean = 0, sd = 1, lower.tail = TRUE)
  - rnorm(n, mean = 0, sd = 1)

- For the t distribution:
  - dt(x, df)
  - pt(q, df, lower.tail = TRUE)
  - qt(p, df, lower.tail = TRUE)
  - rt(n, df)

- For the F distribution:
  - df(x, df1, df2)
  - pf(q, df1, df2, lower.tail = TRUE)
  - qf(p, df1, df2, lower.tail = TRUE)
  - rf(n, df1, df2)

- For the χ^2 distribution:
  - dchisq(x, df)
  - pchisq(q, df, lower.tail = TRUE)
  - qchisq(p, df, lower.tail = TRUE)
  - rchisq(n, df)

3.3 Example: Confidence Intervals in the *food* Model

Let us calculate a 95% confidence interval for the coefficient on *income* in the food expenditure model. Besides coalculating confidence intervals, the following lines of code demonstratre how to retrieve information such as standard errors of coefficients from the `summary()` output. The function `summary` summarizes the results of a linear regression, some of which are not available directly from running the model itself.

```r
library(PoEdata)
data("food")
alpha <- 0.05 # chosen significance level
mod1 <- lm(food_exp~income, data=food)
b2 <- coef(mod1)[[2]]
df <- df.residual(mod1) # degrees of freedom
smod1 <- summary(mod1)
seb2 <- coef(smod1)[2,2] # se(b2)
tc <- qt(1-alpha/2, df)
lowb <- b2-tc*seb2  # lower bound
upb <- b2+tc*seb2   # upper bound
```

The resulting confidence interval for the coefficient b_2 in the *food* simple regression model is $(5.97, 14.45)$.

R has a special function, `confint(model)`, that can calculate confidence intervals taking as its argument the name of a regression model. The result of applying this function is a $K \times 2$ matrix with a confidence interval (two values: lower and upper bound) on each row and a number of lines equal to the number of parametrs in the model (equal to 2 in the simple linear regression model). Compare the values from the next code to the ones from the previous to check that they are equal.

```
ci <- confint(mod1)
print(ci)
```

```
##                  2.5 %   97.5 %
## (Intercept) -4.46328 171.2953
## income       5.97205  14.4472
```

```
lowb_b2 <- ci[2, 1] # lower bound
upb_b2 <- ci[2, 2]  # upper bound.
```

3.4 Confidence Intervals in Repeated Samples

```
data("table2_2")
alpha <- 0.05
mod1 <- lm(y1~x, data=table2_2) # just to determine df
tc <- qt(1-alpha/2, df) # critical t

# Initiate four vectors that will store the results:
lowb1 <- rep(0, 10) # 'repeat 0 ten times'
upb1 <- rep(0, 10)  # (alternatively, 'numeric(10)')
lowb2 <- rep(0, 10)
upb2 <-rep(0, 10)

# One loop for each set of income:
for(i in 2:11){  # The curly bracket begins the loop
  dat <- data.frame(cbind(table2_2[,1], table2_2[,i]))
  names(dat) <- c("x", "y")
  mod1 <- lm(y~x, data=dat)
  smod1 <- summary(mod1)
  b1 <- coef(mod1)[[1]]
  b2 <- coef(mod1)[[2]]
  seb1 <- coef(smod1)[1,2]
```

Table 3.1: Confidence intervals for b_1 and b_2

lowb1	upb1	lowb2	upb2
49.542182	213.846	2.51843	10.4413
-9.831097	124.323	7.64838	14.1174
28.556681	179.264	4.50553	11.7727
-20.959444	113.968	8.64817	15.1545
0.931168	167.534	5.27120	13.3049
-66.044847	119.302	9.08188	18.0194
-0.629753	129.046	7.80618	14.0592
19.194721	140.129	6.84889	12.6804
38.315701	156.287	5.20631	10.8950
20.691744	171.232	4.13968	11.3988

```
  seb2 <- coef(smod1)[2,2]
  lowb1[i-1] <- b1-tc*seb1
  upb1[i-1] <- b1+tc*seb1
  lowb2[i-1] <- b2-tc*seb2
  upb2[i-1] <- b2+tc*seb2
} # This curly bracket ends the loop

table <- data.frame(lowb1, upb1, lowb2, upb2)
kable(table,
  caption="Confidence intervals for $b_{1}$ and $b_{2}$",
  align="c")
```

Table 3.1 shows the lower and upper bounds of the confidence intervals of β_1 and β_2.

3.5 Hypothesis Tests

Hypothesis testing seeks to establish whether the data sample at hand provides sufficient evidence to support a certain conjecture (hypothesis) about a population parameter such as the intercept in a regresion model, the slope, or some combination of them. The procedure requires three elements: the hypotheses (the *null* and the *alternative*), a test statistic, which in the case of the simple linear regression parameters is the t-ratio, and a significance level, α.

Suppose we believe that there is a significant relationship between a household's income and its expenditure on food, a conjecture which has led us to formulate the food expenditure model in the first place. Thus, we believe that β_2, the (population)

Table 3.2: Regression output showing the coefficients

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	83.4160	43.41016	1.92158	0.062182
income	10.2096	2.09326	4.87738	0.000019

parameter, is different from zero. Equation 3.5 shows the null and alternative hypotheses for such a test.

$$H_0 : \beta_2 = 0, \quad H_A : \beta_2 \neq 0 \tag{3.5}$$

In general, if a null hypothesis $H_0 : \beta_k = c$ is true, the t statistic (the t-ratio) is given by Equation 3.6 and has a t distribution with $N - 2$ degrees of freedom.

$$t = \frac{b_k - c}{se(b_k)} \sim t_{(N-2)} \tag{3.6}$$

Let us test the hypothesis in Equation 3.5, which makes $c = 0$ in Equation 3.6. Let $\alpha = 0.05$. Table 3.2 shows the regression output.

```
alpha <- 0.05
library(PoEdata); library(xtable); library(knitr)
data("food")
mod1 <- lm(food_exp~income, data=food)
smod1 <- summary(mod1)
table <- data.frame(xtable(mod1))
kable(table,
  caption="Regression output showing the coefficients")
```

```
b2 <- coef(mod1)[["income"]] #coefficient on income
# or:
b2 <- coef(mod1)[[2]] # the coefficient on income
seb2 <- sqrt(vcov(mod1)[2,2]) #standard error of b2
df <- df.residual(mod1) # degrees of freedom
t <- b2/seb2
tcr <- qt(1-alpha/2, df)
```

The results $t = 4.88$ and $t_{cr} = 2.02$ show that $t > t_{cr}$, and therefore t falls in the rejection region (see Figure 3.1).

```
# Plot the density function and the values of t:
curve(dt(x, df), -2.5*seb2, 2.5*seb2, ylab=" ", xlab="t")
abline(v=c(-tcr, tcr, t), col=c("red", "red", "blue"),
```

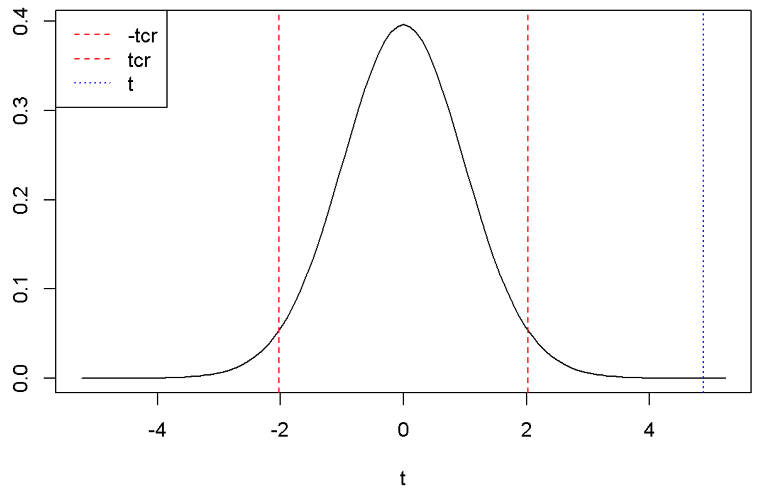


Figure 3.1: A two-tail hypothesis testing for b_2 in the $food$ example

```
       lty=c(2,2,3))
legend("topleft", legend=c("-tcr", "tcr", "t"), col=
         c("red", "red", "blue"), lty=c(2, 2, 3))
```

Suppose we are interested to determine if β_2 is greater than 5.5. This conjecture will go into the alternative hypothesis: $H_0 \leq 5.5, \quad H_A > 5.5$. The procedure is the same as for the two-tail test, but now the whole rejection region is to the right of the critical value t_{cr}.

```
c <- 5.5
alpha <- 0.05
t <- (b2-c)/seb2
tcr <- qt(1-alpha, df) # note: alpha is not divided by 2
curve(dt(x, df), -2.5*seb2, 2.5*seb2, ylab=" ", xlab="t")
abline(v=c(tcr, t), col=c("red", "blue"), lty=c(2, 3))
legend("topleft", legend=c("tcr", "t"),
       col=c("red", "blue"), lty=c(2, 3))
```

Figure 3.2 shows $t_{cr} = 1.685954$, $t = 2.249904$. Since t falls again in the rejection region, we can reject the null hypothesis $H_0 : \beta_2 \leq 0$.

A left-tail test is not different from the right-tail one, but of course the rejection region is to the left of t_{cr}. For example, if we are interested to determine if β_2 is less than

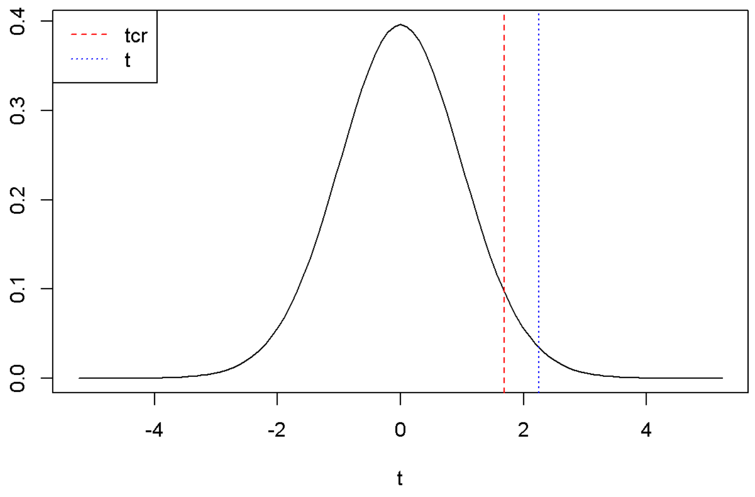


Figure 3.2: Right-tail test: the rejection region is to the right of t_{cr}

15, we place this conjecture in the alternative hypothesis: $H_0 \geq 15, \quad H_A < 15$. The novelty here is how we use the `qt()` function to calculate t_{cr}: instead of `qt(1-alpha, ...)`, we need to use `qt(alpha, ...)`. Figure 3.3 illustrates this example, where the rejection region is, remember, to the left of t_{cr}.

```
c <- 15
alpha <- 0.05
t <- (b2-c)/seb2
tcr <- qt(alpha, df) # note: alpha is not divided by 2
curve(dt(x, df), -2.5*seb2, 2.5*seb2, ylab=" ", xlab="t")
abline(v=c(tcr, t), col=c("red", "blue"), lty=c(2, 3))
legend("topleft", legend=c("tcr", "t"),
       col=c("red", "blue"), lty=c(2, 3))
```

R does automatically a *test of significance*, which is indeed testing the hypothesis $H_0 : \beta_2 = 0, \quad H_A : \beta_2 \neq 0$. The regression output shows the values of the t-ratio for all the regression coefficients.

```
library(PoEdata)
data("food")
mod1 <- lm(food_exp ~ income, data = food)
table <- data.frame(round(xtable(summary(mod1)), 3))
kable(table, caption = "Regression output for the 'food' model")
```

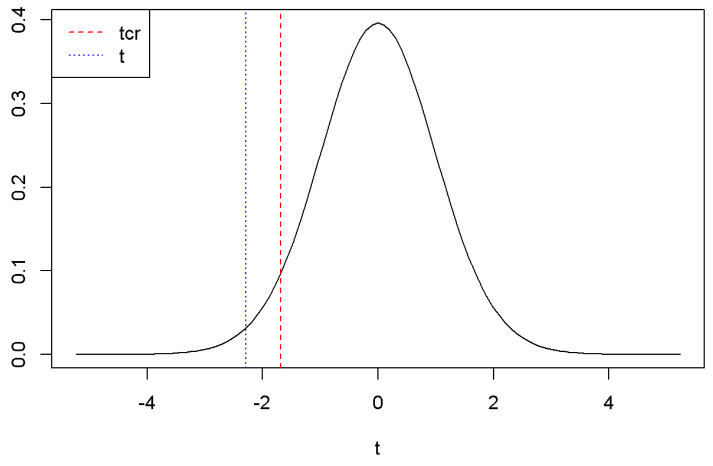


Figure 3.3: Left-tail test: the rejection region is to the left of $t_{cr} =$

Table 3.3: Regression output for the 'food' model

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	83.416	43.410	1.922	0.062
income	10.210	2.093	4.877	0.000

Table 3.3 shows the regression output where the t-statistics of the coefficients can be observed.

3.6 The p-Value

In the context of a hypothesis test, the p-value is the area outside the calculated t-statistic; it is the probability that the t-ratio takes a value that is more extreme than the calculated one, under the assumption that the null hypothesis is true. We reject the null hypothesis if the p-value is less than a chosen significance level. For a right-tail test, the p-value is the area to the right of the calculated t; for a left-tail test it is the area to the left of the calculated t; for a two-tail test the p-value is split in two equal amounts: $p/2$ to the left and $p/2$ to the right. p-values are calculated in R by the function `pt(t, df)`, where t is the calculated t-ratio and df is the number of degrees of freedom in the estimated model.

Right-tail test, $H_0 : \beta_2 \leq c, \quad H_A : \beta_2 > c.$

```r
# Calculating the p-value for a right-tail test
c <- 5.5
t <- (b2-c)/seb2
p <- 1-pt(t, df) # pt() returns p-values;
```

The right-tail test shown in Figure 3.2 gives the p-value $p = 0.01516$.

Left-tail test, $H_0 : \beta_2 \geq c, \quad H_A : \beta_2 < c.$

```r
# Calculating the p-value for a left-tail test
c <- 15
t <- (b2-c)/seb2
p <- pt(t, df)
```

The left-tail test shown in Figure 3.3 gives the p-value $p = 0.01388$.

Two-tail test, $H_0 : \beta_2 = c, \quad H_A : \beta_2 \neq c.$

```r
# Calculating the p-value for a two-tail test
c <- 0
t <- (b2-c)/seb2
p <- 2*(1-pt(abs(t), df))
```

The two-tail test shown in Figure 3.4 gives the p-value $p = 2 \times 10^{-5}$, for a t-ratio $t = 4.88$.

```r
curve(dt(x, df), from=-2.5*seb2, to=2.5*seb2)
abline(v=c(-t, t), col=c("blue", "blue"), lty=c(2, 2))
```

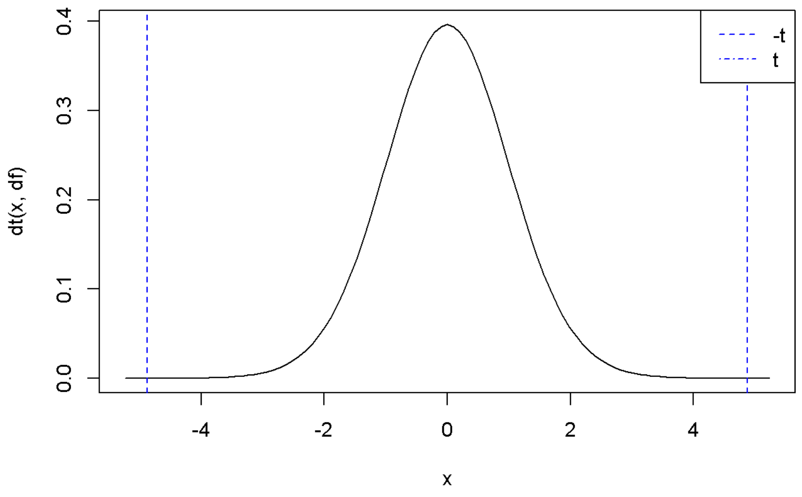


Figure 3.4: The p-value in two-tail hypothesis testing

Table 3.4: Regression output showing p-values

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	83.4160	43.41016	1.92158	0.062182
income	10.2096	2.09326	4.87738	0.000019

```r
legend("topright", legend=c("-t", "t"),
       col=c("blue", "blue"), lty=c(2, 4))
```

R gives the p-values in the standard regression output, which we can retrieve using the `summary(model)` function. Table 3.4 shows the output of the regression model, where the p-values can be observed.

```r
table <- data.frame(xtable(smod1))
knitr::kable(table, caption=
   "Regression output showing p-values")
```

3.7 Testing Linear Combinations of Parameters

Sometimes we wish to estimate the expected value of the dependent variable, y, for a given value of x. For example, according to our *food* model, what is the average expenditure of a household having income of \$2000? We need to estimate the linear combination of the regression coefficients β_1 and β_2 given in Equation 3.7 (let's denote the linear combination by L).

$$L = E(food\_exp|income = 20) = \beta_1 + 20\beta_2 \tag{3.7}$$

Finding confidence intervals and testing hypotheses about the linear combination in Equation 3.7 requires calculating a t-statistic similar to the one for the regression coefficients we calculated before. However, estimating the standard error of the linear combination is not as straightforward. In general, if X and Y are two random variables and a and b two constants, the variance of the linear combination $aX + bY$ is

$$var(aX + bY) = a^2var(X) + b^2var(Y) + 2abcov(X, Y). \tag{3.8}$$

Now, let us apply the formula in Equation 3.8 to the linear combination of $\beta 1$ and $\beta 2$ given by Equation 3.7, we obtain Equation 3.9.

$$var(b_1 + 20b_2) = var(b_1) + 20^2var(b_2) + 2 \times 20cov(b_1b_2) \tag{3.9}$$

The following sequence of code determines an interval estimate for the expected value of food expenditure in a household earning \$2000 a week.

```r
library(PoEdata)
data("food")
alpha <- 0.05
x <- 20 # income is in 100s, remember?
m1 <- lm(food_exp~income, data=food)
tcr <- qt(1-alpha/2, df) # rejection region right of tcr.
df <- df.residual(m1)
b1 <- m1$coef[1]
b2 <- m1$coef[2]
varb1 <- vcov(m1)[1, 1]
varb2 <- vcov(m1)[2, 2]
covb1b2 <- vcov(m1)[1, 2]
L <- b1+b2*x  # estimated L
varL = varb1 + x^2 * varb2 + 2*x*covb1b2 # var(L)
seL <- sqrt(varL) # standard error of L
lowbL <- L-tcr*seL
upbL <- L+tcr*seL
```

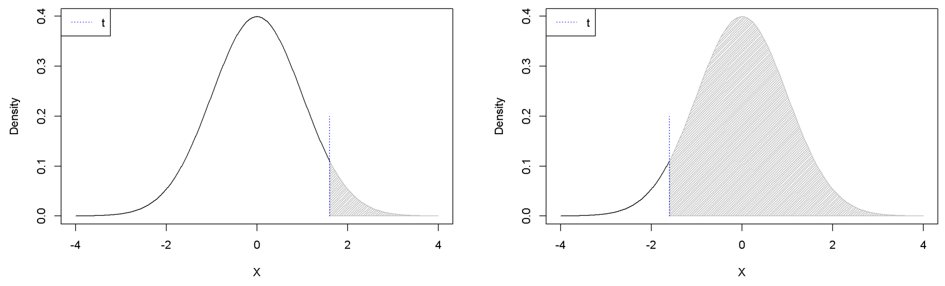


Figure 3.5: $p-$Values for positive and negative t as calculated using the formula $1 - pt(t, df)$

The result is the confidence interval (258.91, 316.31). Next, we test hypotheses about the linear combination L defined in Equation 3.7, looking at the three types of hypotheses: two-tail, left-tail, and right-tail. Equations 3.10 − 3.12 show the test setups for a hypothesized value of food expenditure c.

$$H_0 : L = c, \quad H_A : L \neq c \tag{3.10}$$

$$H_0 : L \geq c, \quad H_A : L < c \tag{3.11}$$

$$H_0 : L \leq c, \quad H_A : L > c \tag{3.12}$$

One should use the function `pt(t, df)` carefully, because it gives wrong results when testing hypotheses using the p-value metod and the calculated t is negative. Therefore, the absolute value of t should be used. Figure 3.5 shows the p-values calculated with the formula `1-pt(t, df)`. When t is positive and the test is two-tail, doubling the p-value `1-pt(t, df)` is correct; but when t is negative, the correct p-value is `2*p(t, df)`.

```
.shadenorm(above=1.6, justabove=TRUE)
segments(1.6,0,1.6,0.2,col="blue", lty=3)
legend("topleft", legend="t", col="blue", lty=3)

.shadenorm(above=-1.6, justabove=TRUE)
segments(-1.6,0,-1.6,0.2,col="blue", lty=3)
legend("topleft", legend="t", col="blue", lty=3)
```

The next sequence uses the values already calculated before, a hypothesized level of food expenditure c=\$250, and an income of \$2000; it tests the two-tail hypothesis in Equation 3.10 first using the "critical t" method, then using the p-value method.

```r
c <- 250
alpha <- 0.05
t <- (L-c)/seL  # t < tcr --> Reject Ho.
tcr <- qt(1-alpha/2, df)

# Or, we can calculate the p-value, as follows:
p_value <- 2*(1-pt(abs(t), df)) #p<alpha -> Reject Ho
```

The results are: $t = 2.65$, $t_{cr} = 2.02$, and $p = 0.0116$. Since $t > t_{cr}$, we reject the null hypothesis. The same result is given by the p-value method, where the p-value is twice the probability area determined by the calculated t.

Chapter 4

Prediction, R-squared, and Modeling

```r
rm(list=ls()) # Caution: this clears the Environment
```

A **prediction** is an estimate of the value of y for a given value of x, based on a regression model of the form shown in Equation 4.1. **Goodness-of-fit** is a measure of how well an estimated regression line approximates the data in a given sample. One such measure is the correlation coefficient between the predicted values of y for all x-s in the data file and the actual y-s. Goodness-of-fit, along with other diagnostic tests help determining the most suitable **functional form** of our regression equation, i.e., the most suitable mathematical relationship between y and x.

$$y_i = \beta_1 + \beta_2 x_i + e_i \tag{4.1}$$

4.1 Forecasting (Predicting a Particular Value)

Assuming that the expected values of the error term in Equation 4.1 is zero, Equation 4.2 gives $\hat{y}_i$, the predicted value of the expectation of y_i given x_i, where b_1 and b_2 are the (least squares) estimates of the regression parameters β_1 and β_2.

$$\hat{y}_i = b_1 + b_2 x_i \tag{4.2}$$

The predicted value $\hat{y}_i$ is a random variable, since it depends on the sample; therefore, we can calculate a confidence interval and test hypothesis about it, provided we can determine its distribution and variance. The prediction has a normal distribution, being a linear combination of two normally distributed random variables b_1 and b_2, and its variance is given by Equation 4.3. Please note that the variance in Equation 4.3 is not the same as the one in Equation 3.9; the former is the variance of the estimated *expectation* of y, while the latter is the variance of a particular occurrence of y. Let us call the latter the variance of the forecast error. Not surprisingly, the variance of the forecast error is greater than the variance of the predicted $E(y|x)$.

As before, since we need to use an estimated variance, we use a t-distribution instead of a normal one. Equation 4.3 applies to any given x, say x_0, not only to those x-s in the dataset.

$$\widehat{var(f_i)} = \hat{\sigma}^{\,2}\left[1 + \frac{1}{N} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{N}(x_j - \bar{x})^2}\right], \tag{4.3}$$

which can be reduced to

$$\widehat{var(f_i)} = \hat{\sigma}^{\,2} + \frac{\hat{\sigma}^{\,2}}{N} + (x_i - \bar{x})^2\widehat{var(b_2)} \tag{4.4}$$

Let's determine a standard error for the *food* equation for a household earning \$2000 a week, i.e., at $x = x_0 = 20$, using Equation 4.4; to do so, we need to retrieve $var(b_2)$ and $\hat{\sigma}$, the standard error of regression from the regression output.

```r
library(PoEdata)
data("food")
alpha <- 0.05
x <- 20
xbar <- mean(food$income)
m1 <- lm(food_exp~income, data=food)
b1 <- coef(m1)[[1]]
b2 <- coef(m1)[[2]]
yhatx <- b1+b2*x
sm1 <- summary(m1)
df <- df.residual(m1)
tcr <- qt(1-alpha/2, df)
N <- nobs(m1)   #number of observations, N
N <- NROW(food) #just another way of finding N
varb2 <- vcov(m1)[2, 2]
sighat2 <- sm1$sigma^2 # estimated variance
varf <- sighat2+sighat2/N+(x-xbar)^2*varb2 #forecast variance
sef <- sqrt(varf) #standard error of forecast
lb <- yhatx-tcr*sef
ub <- yhatx+tcr*sef
```

The result is the confidence interval for the forecast $(104.13, 471.09)$, which is, as expected, larger than the confidence interval of the estimated expected value of y based on Equation 3.9.

Let us calculate confidence intervals of the forecast for all the observations in the sample and draw the upper and lower limits together with the regression line. Figure 4.1 shows the confidence interval band about the regression line.

```r
sef <- sqrt(sighat2+sighat2/N+(food$income-xbar)^2*varb2)
yhatv <- fitted.values(m1)
lbv <- yhatv-tcr*sef
ubv <- yhatv+tcr*sef
xincome <- food$income
dplot <- data.frame(xincome, yhatv, lbv, ubv)
dplotord <- dplot[order(xincome), ]
xmax <- max(dplotord$xincome)
xmin <- min(dplotord$xincome)
ymax <- max(dplotord$ubv)
ymin <- min(dplotord$lbv)
plot(dplotord$xincome, dplotord$yhatv,
     xlim=c(xmin, xmax),
     ylim=c(ymin, ymax),
     xlab="income", ylab="food expenditure",
     type="l")
lines(dplotord$ubv~dplotord$xincome, lty=2)
lines(dplotord$lbv~dplotord$xincome, lty=2)
```

A different way of finding point and interval estimates for the *predicted* $E(y|x)$ and *forecasted* y (please see the distinction I mentioned above) is to use the `predict()` function in R. This function requires that the values of the independent variable where the prediction (or forecast) is intended have a data frame structure. The next example shows in parallel point and interval estimates of *predicted* and *forecasted* food expenditures for income is \$2000. As I have pointed out before, the point estimate is the same for both prediction and forecast, but the interval estimates are very different.

```r
incomex=data.frame(income=20)
predict(m1, newdata=incomex, interval="confidence",level=0.95)
```

fit	lwr	upr
287.609	258.907	316.311

```r
predict(m1, newdata=incomex, interval="prediction",level=0.95)
```

fit	lwr	upr
287.609	104.132	471.085

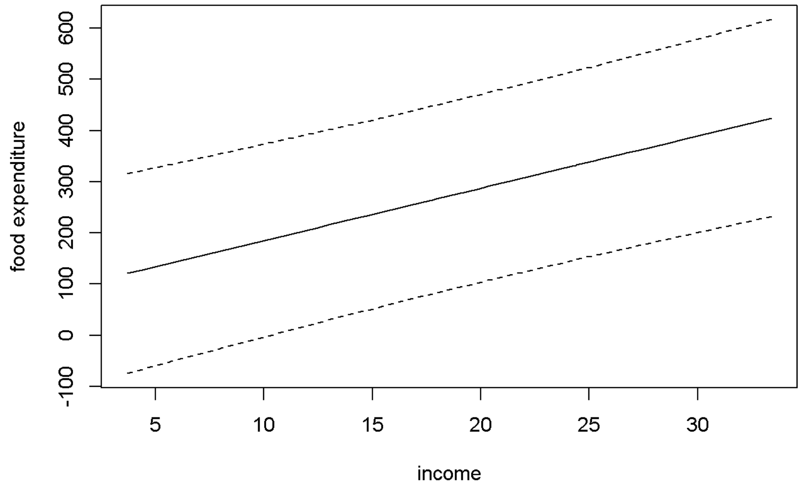


Figure 4.1: Forecast confidence intervals for the *food* simple regression

Let us now use the `predict()` function to replicate Figure 4.1. The result is Figure 4.2, which shows, besides the interval estimation band, the points in the dataset. (I will create new values for `income` just for the purpose of plotting.)

```
xmin <- min(food$income)
xmax <- max(food$income)
income <- seq(from=xmin, to=xmax)
ypredict <- predict(m1, newdata=data.frame(income),
                  interval="confidence")
yforecast <- predict(m1, newdata=data.frame(income),
                  interval="predict")
matplot(income, cbind(ypredict[,1], ypredict[,2], ypredict[,3],
                      yforecast[,2], yforecast[,3]),
        type ="l", lty=c(1, 2, 2, 3, 3),
        col=c("black", "red", "red", "blue", "blue"),
        ylab="food expenditure", xlab="income")
points(food$income, food$food_exp)
legend("topleft",
        legend=c("E[y|x]", "lwr_pred", "upr_pred",
                "lwr_forcst","upr_forcst"),
        lty=c(1, 2, 2, 3, 3),
        col=c("black", "red", "red", "blue", "blue")
        )
```

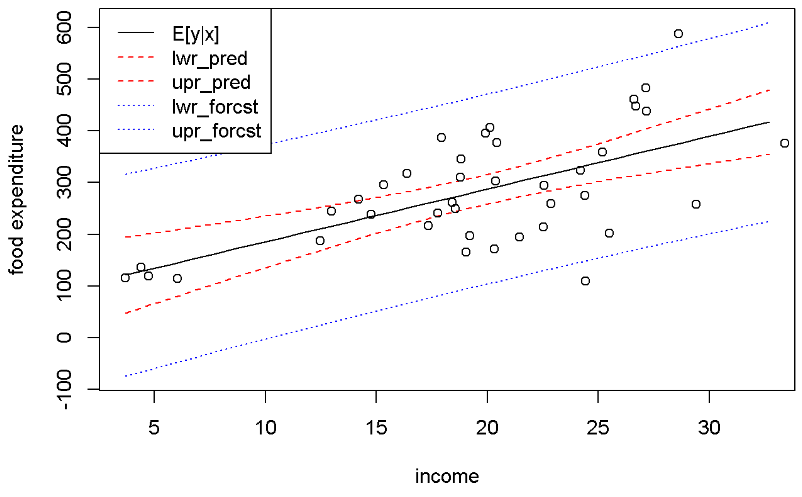


Figure 4.2: Predicted and forecasted bands for the *food* dataset

Figure 4.2 presents the predicted and forecasted bands on the same graph, to show that they have the same point estimates (the black, solid line) and that the forecasted band is much larger than the predicted one. Put another way, you may think about the distinction between the two types of intervals that we called *prediction* and *forecast* as follows: the prediction interval is not supposed to include, say, 95 percent of the points, but to include the regression line, $E(y|x)$, with a probability of 95 percent; the forecasted interval, on the other hand, should include any true point with a 95 percent probability.

4.2 Goodness-of-Fit

The total variation of y about its sample mean, SST, can be decomposed in variation about the regression line, SSE, and variation of the regression line about the mean of y, SSR, as Equation 4.5 shows.

$$SST = SSR + SSE \tag{4.5}$$

The **coefficient of determination**, R^2, is defined as the proportion of the variance in y that is explained by the regression, SSR, in the total variation in y, SST. Dividing both sides of the Equation 4.5 by SST and re-arranging terms gives a formula to calculate R^2, as shown in Equation 4.6.

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{4.6}$$

R^2 takes values between 0 and 1, with higher values showing a closer fit of the regression line to the data. In R, the value of R^2 can be retrieved from the summary of the regression model under the name `r.squared`; for instance, in our *food* example, $R^2 = 0.385$. R^2 is also printed as part of the summary of a regression model, as the following code sequence shows. (The parentheses around a command tells R to print the result.)

```
(rsq <- sm1$r.squared) #or
```

```
## [1] 0.385002
```

```
sm1 #prints the summary of regression model m1
```

```
##
## Call:
## lm(formula = food_exp ~ income, data = food)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -223.03  -50.82   -6.32   67.88  212.04
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    83.42      43.41    1.92    0.062 .
## income         10.21       2.09    4.88 0.000019 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 89.5 on 38 degrees of freedom
## Multiple R-squared:  0.385,  Adjusted R-squared:  0.369
## F-statistic: 23.8 on 1 and 38 DF,  p-value: 0.0000195
```

If you need the sum of squared errors, SSE, or the sum of squares due to regression, SSR, use the `anova` function, which has the structure shown in Table 4.1.

```
anov <- anova(m1)
dfr <- data.frame(anov)
kable(dfr,
  caption="Output generated by the `anova` function")
```

Table 4.1 indicates that $SSE =$ `anov[2,2]` $= 3.045052 \times 10^5$, $SSR =$ `anov[1,2]` $= 1.90627 \times 10^5$, and $SST =$ `anov[1,2]+anov[2,2]` $= 4.951322 \times 10^5$. In our

Table 4.1: Output generated by the ‘anova‘ function

	Df	Sum.Sq	Mean.Sq	F.value	Pr..F.
income	1	190627	190626.98	23.7888	0.000019
Residuals	38	304505	8013.29	NA	NA

simple regression model, the sum of squares due to regression only includes the variable `income`. In multiple regression models, which are models with more than one independent variable, the sum of squares due to regression is equal to the sum of squares due to all independent variables. The `anova` results in Table 4.1 include other useful information: the number of degrees of freedom, `anov[2,1]` and the estimated variance $\hat{\sigma}^2$ =`anov[2,3]`.

4.3 Linear-Log Models

Non-linear functional forms of regression models are useful when the relationship between two variables seems to be more complex than the linear one. One can decide to use a non-linear functional form based on a mathematical model, reasoning, or simply inspecting a scatter plot of the data. In the *food expenditure* model, for example, it is reasonable to believe that the amount spent on food increases faster at lower incomes than at higher incomes. In other words, it increases at a decreasing rate, which makes the regression curve flatten out at higher incomes.

What function could one use to model such a relationship? The logarithmic function fits this profile and, as it turns out, it is relatively easy to interpret, which makes it very popular in econometric models. The general form of a *linear-log* econometric model is provided in Equation 4.7.

$$y_i = \beta_1 + \beta_2 log(x_i) + e_i \tag{4.7}$$

The *marginal effect* of a change in x on y is the slope of the regression curve and is given by Equation 4.8; unlike in the linear form, it depends on x and it is, therefore, only valid for small changes in x.

$$\frac{dy}{dx} = \frac{\beta_2}{x} \tag{4.8}$$

Related to the linear-log model, another measure of interest in economics is the *semi-elasticity* of y with respect to x, which is given by Equation 4.9. Semi-elasticity suggests that a change in x of 1% changes y by $\beta_2/100$ units of y. Since semi-elasticity also changes when x changes, it should only be determined for small changes in x.

Table 4.2: Linear-log model output for the *food* example

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	-97.1864	84.2374	-1.15372	0.25582
log(income)	132.1658	28.8046	4.58836	0.00005

$$dy = \frac{\beta_2}{100}(\%\Delta x) \tag{4.9}$$

Another quantity that might be of interest is the *elasticity* of y with respect to x, which is given by Equation 4.10 and indicates that a one percent increase in x produces a (β_2/y) percent change in y.

$$\%\Delta y = \frac{\beta_2}{y}(\%\Delta x) \tag{4.10}$$

Let us estimate a linear-log model for the *food* dataset, draw the regression curve, and calculate the marginal effects for some given values of the dependent variable.

```r
mod2 <- lm(food_exp~log(income), data=food)
tbl <- data.frame(xtable(mod2))
kable(tbl, digits=5,
      caption="Linear-log model output for the *food* example")
```

```r
b1 <- coef(mod2)[[1]]
b2 <- coef(mod2)[[2]]
pmod2 <- predict(mod2, newdata=data.frame(income),
                 interval="confidence")
plot(food$income, food$food_exp, xlab="income",
     ylab="food expenditure")
lines(pmod2[,1]~income, lty=1, col="black")
lines(pmod2[,2]~income, lty=2, col="red")
lines(pmod2[,3]~income, lty=2, col="red")
```

```r
x <- 10 #for a household earning #1000 per week
y <- b1+b2*log(x)
DyDx <- b2/x    #marginal effect
DyPDx <- b2/100 #semi-elasticity
PDyPDx <- b2/y  #elasticity
```

The results for an income of \$1000 are as follows: $dy/dx = 13.217$, which indicates that an increase in income of \$100 (i.e., one unit of x) increases expenditure by \$

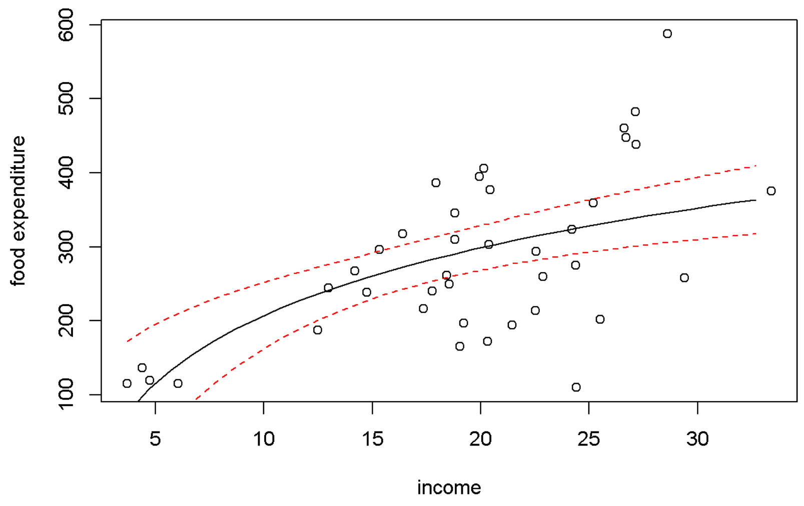


Figure 4.3: Linear-log representation for the *food* data

13.217; for a 1% increase in income, that is, an increase of \$10, expenditure increases by \$ 1.322; and, finally, for a 1% increase in income expenditure incrases by 0.638%.

4.4 Residuals and Diagnostics

Regression results are reliable only to the extent to which the underlying assumptions are met. Plotting the residuals and calculating certain test statistics help deciding whether assumptions such as homoskedasticity, serial correlation, and normality of the errors are not violated. In R, the residuals are stored in the vector `residuals` of the regression output.

```
ehat <- mod2$residuals
plot(food$income, ehat, xlab="income", ylab="residuals")
```

Figure 4.4 shows the residuals of the of the linear-log equation of the *food expenditure* example. One can notice that the spread of the residuals seems to be higher at higher incomes, which may indicate that the heteroskedasticity assumption is violated.

Let us draw a residual plot generated with a simulated model that satisfies the regression assumptions. The data generating process is given by Equation 4.11, where x is a number between 0 and 10, randomly drawn from a uniform distribution, and the error term is randomly drawn from a standard normal distribution. Figure

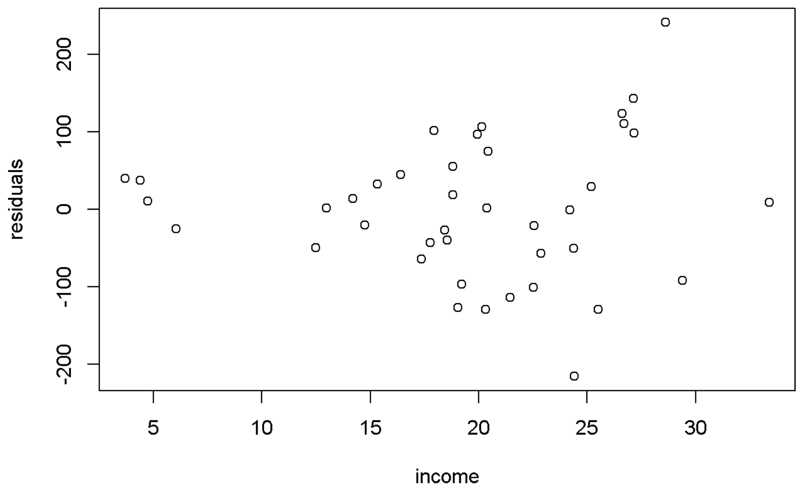


Figure 4.4: Residual plot for the $food$ linear-log model

4.5 illustrates this simulated example.

$$y_i = 1 + x_i + e_i, \quad i = 1, ..., N \tag{4.11}$$

```
set.seed(12345)    #sets the seed for the random number generator
x <- runif(300, 0, 10)
e <- rnorm(300, 0, 1)
y <- 1+x+e
mod3 <- lm(y~x)
ehat <- resid(mod3)
plot(x,ehat, xlab="x", ylab="residuals")
```

The next example illustrates how the residuals look like when a linear functional form is used when the true relationship is, in fact, quadratic. The data generating equation is given in Equation 4.12, where x is the same uniformly distributed between -2.5 and 2.5), and $e \sim N(0, 4)$. Figure 4.6 shows the residuals from estimating an incorrectly specified, linear econometric model when the correct specification should be quadratic.

$$y_i = 15 - 4x_i^2 + e_i, \quad i = 1, ..., N \tag{4.12}$$

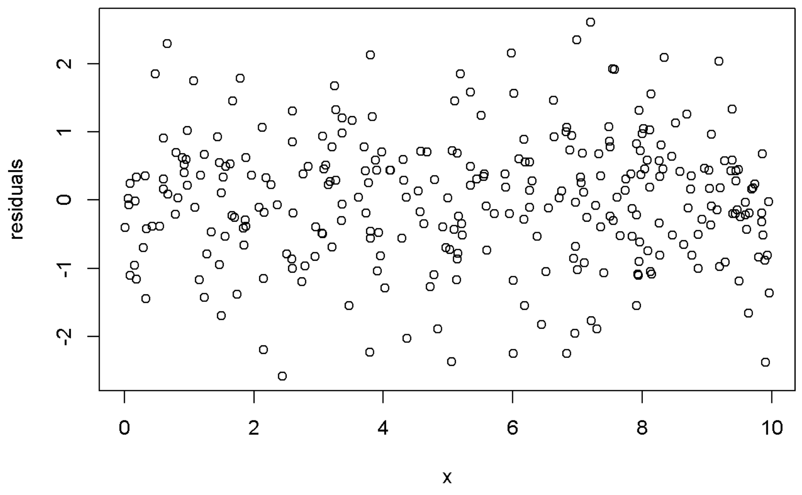


Figure 4.5: Residuals generated by a simulated regression model that satisfies the regression assumptions

```
set.seed(12345)
x <- runif(1000, -2.5, 2.5)
e <- rnorm(1000, 0, 4)
y <- 15-4*x^2+e
mod3 <- lm(y~x)
ehat <- resid(mod3)
ymi <- min(ehat)
yma <- max(ehat)
plot(x, ehat, ylim=c(ymi, yma),
     xlab="x", ylab="residuals",col="grey")
```

Another assumption that we would like to test is the normality of the residuals, which assures reliable hypothesis testing and confidence intervals even in small samples. This assumption can be assessed by inspecting a histogram of the residuals, as well as performing a Jarque-Bera test, for which the null hypothesis is “Series is normally distributed”. Thus, a small p-value rejects the null hypothesis, which means the series fails the normality test. The Jarque-Bera test requires installing and loading the package `tseries` in R. Figure 4.7 shows a histogram and a superimposed normal distribution for the linear *food expenditure* model.

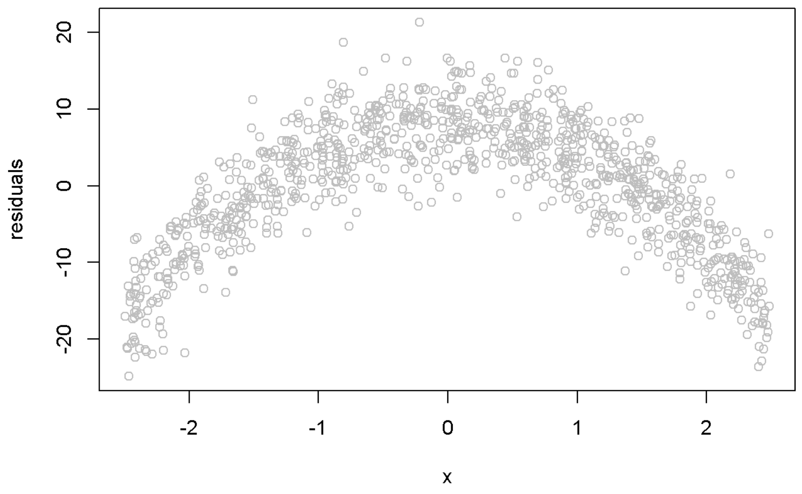


Figure 4.6: Simulated quadratic residuals from an incorrectly specified econometric model

```r
library(tseries)
mod1 <- lm(food_exp~income, data=food)
ehat <- resid(m1)
ebar <- mean(ehat)
sde <- sd(ehat)
hist(ehat, col="grey", freq=FALSE, main="",
     ylab="density", xlab="ehat")
curve(dnorm(x, ebar, sde), col=2, add=TRUE,
      ylab="density", xlab="ehat")
```

```r
jarque.bera.test(ehat) #(in package 'tseries')
```

```
##
##  Jarque Bera Test
##
## data:  ehat
## X-squared = 0.06334, df = 2, p-value = 0.969
```

While the histogram in Figure 4.7 may not strongly support one conclusion or another about the normlity of `ehat`, the Jarque-Bera test is unambiguous: there is no evidence against the normality hypothesis.

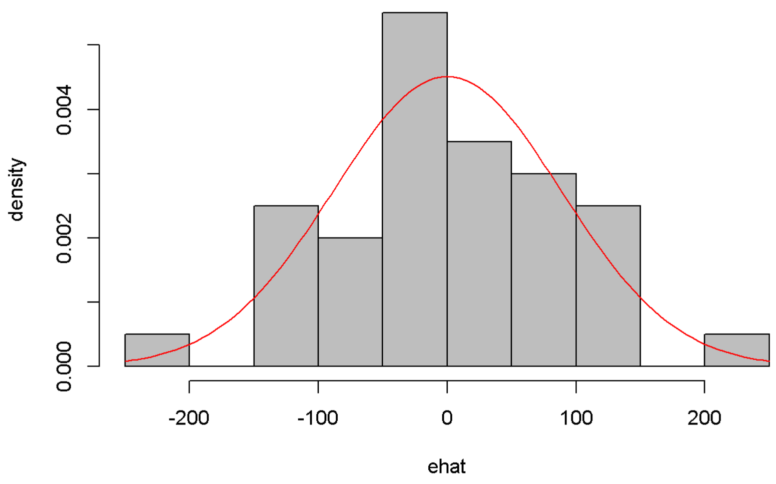


Figure 4.7: Histogram of residuals from the $food$ linear model

4.5 Polynomial Models

Regression models may include quadratic or cubic terms to better describe the nature of the dadta. The following is an example of quadratic and cubic model for the `wa_wheat` dataset, which gives annual wheat yield in tonnes per hectare in Greenough Shire in Western Australia over a period of 48 years. The *linear* model is given in Equation 4.13, where the subscript t indicates the observation period.

$$yield_t = \beta_1 + \beta_2 time_t + e_t \tag{4.13}$$

```
library(PoEdata)
data("wa_wheat")
mod1 <- lm(greenough~time, data=wa_wheat)
ehat <- resid(mod1)
plot(wa_wheat$time, ehat, xlab="time", ylab="residuals")
```

Figure 4.8 shows a pattern in the residuals generated by the linear model, which may inspire us to think of a more appropriate functional form, such as the one in Equation 4.14.

$$yield_t = \beta_1 + \beta_2 time_t^3 + e_t \tag{4.14}$$

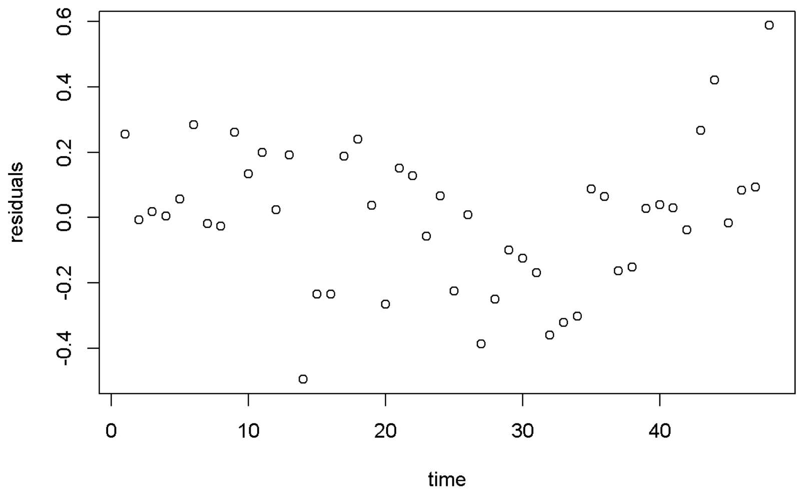


Figure 4.8: Residuals from the linear *wheatyield* model

Please note in the following code sequence the use of the function `I()`, which is needed in R when an independent variable is transformed by mathematical operators. You do not need the operator `I()` when an independent variable is transformed through a function such as $log(x)$. In our example, the transformation requiring the use of `I()` is raising `time` to the power of 3. Of course, you can create a new variable, `x3=x^3` if you wish to avoid the use of `I()` in a regression equation.

```r
mod2 <- lm(wa_wheat$greenough~I(time^3), data=wa_wheat)
ehat <- resid(mod2)
plot(wa_wheat$time, ehat, xlab="time", ylab="residuals")
```

Figure 4.9 displays a much better image of the residuals than Figure 4.8, since the residuals are more evenly spread about the zero line.

4.6 Log-Linear Models

Transforming the dependent variable with the $log()$ function is useful when the variable has a skewed distribution, which is in general the case with amounts that cannot be negative. The $log()$ transformation often makes the distribution closer to normal. The general log-linear model is given in Equation 4.15.

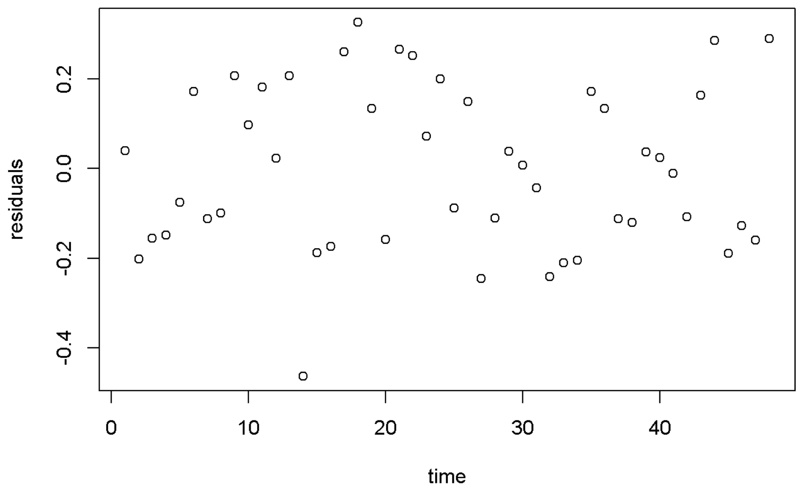


Figure 4.9: Residuals from the cubic *wheatyield* model

$$log(y_i) = \beta_1 + \beta_2 x_i + e_i \tag{4.15}$$

The following formulas are easily derived from the log-linear Equation 4.15. The semi-elasticity has here a different interpretation than the one in the linear-log model: here, an increase in x by one unit (of x) produces a change of $100b_2$ percent in y. For small changes in x, the amount $100b_2$ in the log-linear model can also be interpreted as the growth rate in y (corresponding to a unit increase in x). For instance, if x is time, then $100b_2$ is the growth rate in y per unit of time.

- Prediction: $\hat{y}_n = exp(b_1 + b_2 x)$, or $\hat{y}_c = exp(b_1 + b_2 x + \frac{\hat{\sigma}^2}{2})$, with the "natural" predictor $\hat{y}_n$ to be used in small samples and the "corrected" predictor, $\hat{y}_c$, in large samples
- Marginal effect (slope): $\frac{dy}{dx} = b_2 y$
- Semi-elasticity: $\%\Delta y = 100 b_2 \Delta x$

Let us do these calculations first for the *yield* equation using the *wa_wheat* dataset.

```
mod4 <- lm(log(greenough)~time, data=wa_wheat)
smod4 <- summary(mod4)
tbl <- data.frame(xtable(smod4))
kable(tbl, caption="Log-linear model for the *yield* equation")
```

Table 4.3 gives $b_2 = 0.017844$, which indicates that the rate of growth in wheat

Table 4.3: Log-linear model for the *yield* equation

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	-0.343366	0.058404	-5.87914	0
time	0.017844	0.002075	8.59911	0

Table 4.4: Log-linear 'wage' regression output

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	1.609444	0.086423	18.6229	0
educ	0.090408	0.006146	14.7110	0

production has increased at an average rate of approximately 1.78 percent per year.

The *wage* log-linear equation provides another example of calculating a growth rate, but this time the independent variable is not *time*, but *education*. The predictions and the slope are calculated for $educ = 12$ years.

```
data("cps4_small", package="PoEdata")
xeduc <- 12
mod5 <- lm(log(wage)~educ, data=cps4_small)
smod5 <- summary(mod5)
tabl <- data.frame(xtable(smod5))
kable(tabl, caption="Log-linear 'wage' regression output")
```

```
b1 <- coef(smod5)[[1]]
b2 <- coef(smod5)[[2]]
sighat2 <- smod5$sigma^2
g <- 100*b2                 #growth rate
yhatn <- exp(b1+b2*xeduc) #"natural" predictiction
yhatc <- exp(b1+b2*xeduc+sighat2/2) #corrected prediction
DyDx <- b2*yhatn            #marginal effect
```

Here are the results of these calculations: "natural" prediction $\hat{y}_n = 14.796$; corrected prediction, $\hat{y}_c = 16.996$; growth rate $g = 9.041$; and marginal effect $\frac{dy}{dx} = 1.34$. The growth rate indicates that an increase in education by one unit (see the data description using `?cps4_small`) increases hourly wage by 9.041 percent.

Figure 4.10 presents the "natural" and the "corrected" regression lines for the *wage* equation, together with the actual data points.

```
education=seq(0,22,2)
yn <- exp(b1+b2*education)
yc <- exp(b1+b2*education+sighat2/2)
```

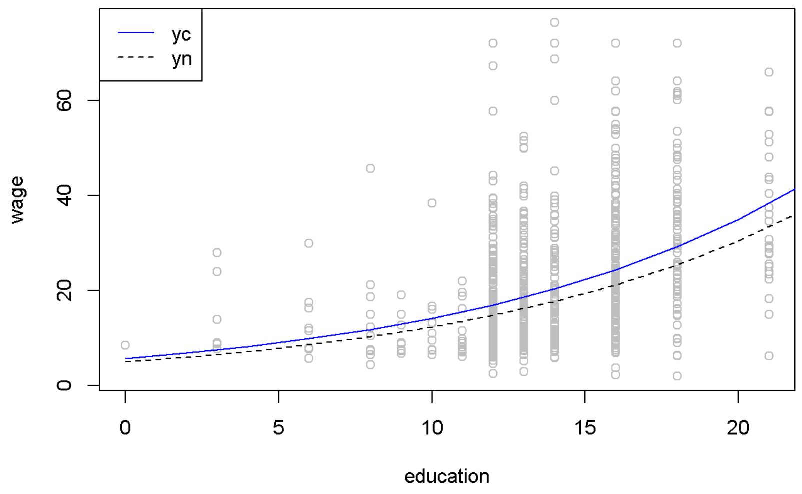


Figure 4.10: The 'normal' and 'corrected' regression lines in the log-linear *wage* equation

```
plot(cps4_small$educ, cps4_small$wage,
     xlab="education", ylab="wage", col="grey")
lines(yn~education, lty=2, col="black")
lines(yc~education, lty=1, col="blue")
legend("topleft", legend=c("yc","yn"),
       lty=c(1,2), col=c("blue","black"))
```

The regular R^2 cannot be used to compare two regression models having different dependent variables such as a linear-log and a log-linear models; when such a comparison is needed, one can use the general R^2, which is $R_g^2 = [corr(y, \hat{y}]^2$. Let us calculate the generalized R^2 for the quadratic and the log-linear *wage* models.

```
mod4 <- lm(wage~I(educ^2), data=cps4_small)
yhat4 <- predict(mod4)
mod5 <- lm(log(wage)~educ, data=cps4_small)
smod5 <- summary(mod5)
b1 <- coef(smod5)[[1]]
b2 <- coef(smod5)[[2]]
sighat2 <- smod5$sigma^2
yhat5 <- exp(b1+b2*cps4_small$educ+sighat2/2)
rg4 <- cor(cps4_small$wage, yhat4)^2
```

```r
rg5 <- cor(cps4_small$wage,yhat5)^2
```

The quadratic model yields $R_g^2 = 0.188$, and the log-linear model yields $R_g^2 = 0.186$; since the former is higher, we conclude that the quadratic model is a better fit to the data than the log-linear one. (However, other tests of how the two models meet the assumptions of linear refgression may reach a different conclusion; R^2 is only one of the model selection criteria.)

To determne a forecast interval estimate in the log-linear model, we first construct the interval in logs using the natural predictor $\hat{y}_n$, then take antilogs of the interval limits. The forecasting error is the same as before, given in Equation 4.4. The following calculations use an education level equal to 12 and $\alpha = 0.05$.

```r
# The *wage* log-linear model
# Prediction interval for educ = 12
alpha <- 0.05
xeduc <- 12
xedbar <- mean(cps4_small$educ)
mod5 <- lm(log(wage)~educ, data=cps4_small)
b1 <- coef(mod5)[[1]]
b2 <- coef(mod5)[[2]]
df5 <- mod5$df.residual
N <- nobs(mod5)
tcr <- qt(1-alpha/2, df=df5)
smod5 <- summary(mod5)
varb2 <- vcov(mod5)[2,2]
sighat2 <- smod5$sigma^2
varf <- sighat2+sighat2/N+(xeduc-xedbar)^2*varb2
sef <- sqrt(varf)
lnyhat <- b1+b2*xeduc
lowb <- exp(lnyhat-tcr*sef)
upb <- exp(lnyhat+tcr*sef)
```

The result is the confidence interval $(5.26, \ 41.62)$. Figure 4.11 shows a 95% confidence band for the log-linear *wage* model.

```r
# Drawing a confidence band for the log-linear
# *wage* equation

xmin <- min(cps4_small$educ)
xmax <- max(cps4_small$educ)+2
education <- seq(xmin, xmax, 2)
lnyhat <- b1+b2*education
yhat <- exp(lnyhat)
```

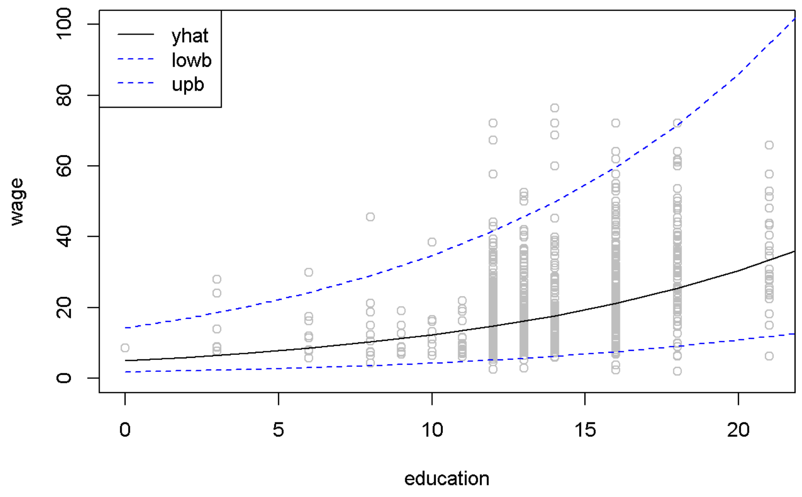


Figure 4.11: Confidence band for the log-linear *wage* equation

```
varf <- sighat2+sighat2/N+(education-xedbar)^2*varb2
sef <- sqrt(varf)
lowb <- exp(lnyhat-tcr*sef)
upb <- exp(lnyhat+tcr*sef)
plot(cps4_small$educ, cps4_small$wage, col="grey",
     xlab="education", ylab="wage", ylim=c(0,100))
lines(yhat~education, lty=1, col="black")
lines(lowb~education, lty=2, col="blue")
lines(upb~education, lty=2, col="blue")
legend("topleft", legend=c("yhat", "lowb", "upb"),
       lty=c(1, 2, 2), col=c("black", "blue", "blue"))
```

4.7 The Log-Log Model

The log-log model has the desirable property that the coefficient of the independent variable is equal to the (constant) elasticity of y with respect to x. Therefore, this model is often used to estimate supply and demand equations. Its standard form is given in Equation 4.16, where y, x, and e are $N \times 1$ vectors.

$$log(y) = \beta_1 + \beta_2 log(x) + e \tag{4.16}$$

Table 4.5: The log-log poultry regression equation

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	3.71694	0.022359	166.2362	0
log(p)	-1.12136	0.048756	-22.9992	0

```r
# Calculating log-log demand for chicken
data("newbroiler", package="PoEdata")
mod6 <- lm(log(q)~log(p), data=newbroiler)
b1 <- coef(mod6)[[1]]
b2 <- coef(mod6)[[2]]
smod6 <- summary(mod6)
tbl <- data.frame(xtable(smod6))
kable(tbl, caption="The log-log poultry regression equation")
```

Table 4.5 gives the log-log regression output. The coefficient on p indicates that an increase in price by 1% changes the quantity demanded by -1.121%.

```r
# Drawing the fitted values of the log-log equation
ngrid <- 20 # number of drawing points
xmin <- min(newbroiler$p)
xmax <- max(newbroiler$p)
step <- (xmax-xmin)/ngrid # grid dimension
xp <- seq(xmin, xmax, step)
sighat2 <- smod6$sigma^2
yhatc <- exp(b1+b2*log(newbroiler$p)+sighat2/2)
yc <- exp(b1+b2*log(xp)+sighat2/2) #corrected q
plot(newbroiler$p, newbroiler$q, ylim=c(10,60),
     xlab="price", ylab="quantity")
lines(yc~xp, lty=1, col="black")
```

```r
# The generalized R-squared:
rgsq <- cor(newbroiler$q, yhatc)^2
```

The generalized R^2, wich uses the corrected fitted values, is equal to 0.8818.

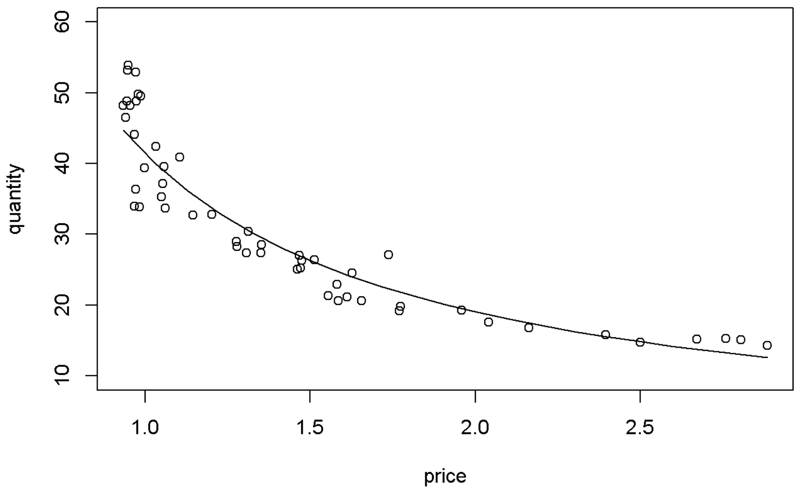


Figure 4.12: Log-log demand for chicken

Chapter 5

The Multiple Regression Model

```r
rm(list=ls())
library(PoEdata)
library(knitr)
library(xtable)
library(printr)
library(effects)
library(car)
library(AER)
library(broom)
```

This chapter uses a few new packages: `effects` (Fox et al. 2016), `car` (Fox and Weisberg 2016), and `AER`, (Kleiber and Zeileis 2015).

5.1 The General Model

A multiple regression model is very similar to the simple regression model, but includes more independent variables. Thus, the interpretation of a slope parameter has to take into account possible changes in other independent variables: a slope parameter, say β_k, gives the change in the dependent variable, y, when the independent variable x_k increases by one unit **while all the other independent variables remain constant**. Equation 5.1 gives the general form of a multiple regression model, where y, x_k, and e are $N \times 1$ vectors, N is the number of observations in the sample, and $k = 1, ..., K$ indicates the k-th independent variable. The notation used in Equation 5.1 implies that the first independent variable, x_1, is an $N \times 1$ vector of 1s.

$$y = \beta_1 + \beta_2 x_2 + ... + \beta_K x_K + e \tag{5.1}$$

Table 5.1: Summary statistics for dataset *andy*

column	n	mean	sd	median	min	max
sales	75	77.3747	6.488537	76.50	62.40	91.20
price	75	5.6872	0.518432	5.69	4.83	6.49
advert	75	1.8440	0.831677	1.80	0.50	3.10

The model assumptions remain the same, with the additional requirement that no independent variable is a linear combination of the others.

5.2 Example: Big Andy's Hamburger Sales

The *andy* dataset includes variables *sales*, which is monthly revenue to the company in \$1000s, *price*, which is a price index of all products sold by Big Andy's, and *advert*, the advertising expenditure in a given month, in \$1000s. Summary statistics for the *andy* dataset is shown in Table 5.1. The basic *andy* model is presented in Equation 5.2.

$$sales = \beta_1 + \beta_2 price + \beta_3 advert + e \tag{5.2}$$

```r
# Summary statistics
data(andy)
s=tidy(andy)[,c(1:5,8,9)]
kable(s,caption="Summary statistics for dataset $andy$")
```

```r
# The basic *andy* model
data("andy",package="PoEdata")
mod1 <- lm(sales~price+advert, data=andy)
smod1 <- data.frame(xtable(summary(mod1)))
kable(smod1,
caption="The basic multiple regression model",
col.names=c("coefficient", "Std. Error", "t-value", "p-value"),
align="c", digits=3)
```

Table 5.2 shows that, for any given (but fixed) value of advertising expenditure, an increase in price by \$1 decreases sales by \$7908. On the other hand, for any given price, sales increase by \$1863 when advertising expenditures increase by \$1000.

```r
effprice <- effect("price", mod1)
plot(effprice)
```

Table 5.2: The basic multiple regression model

	coefficient	Std. Error	t-value	p-value
(Intercept)	118.914	6.352	18.722	0.000
price	-7.908	1.096	-7.215	0.000
advert	1.863	0.683	2.726	0.008

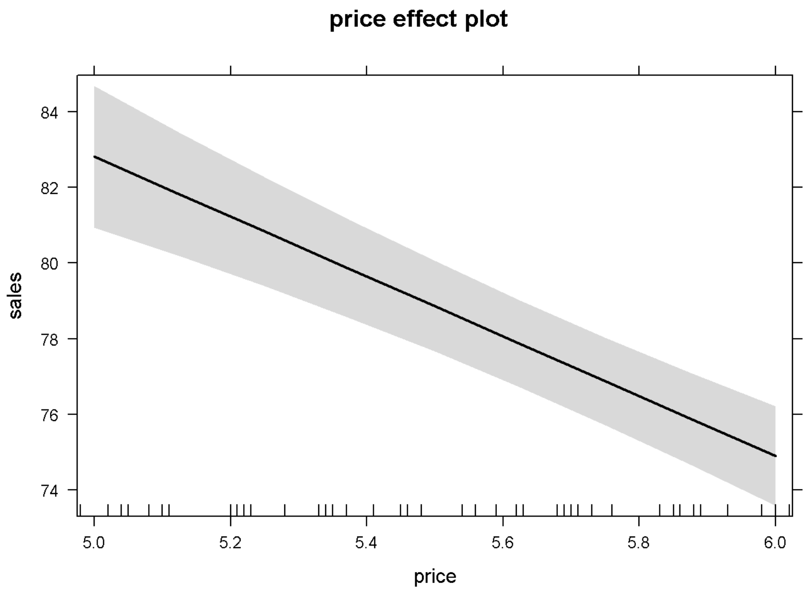


Figure 5.1: The partial effect of *price* in the basic *andy* regression

```
summary(effprice)
```

```
##
##  price effect
## price
##       5     5.5       6
## 82.8089 78.8550 74.9011
##
##  Lower 95 Percent Confidence Limits
## price
##       5     5.5       6
## 80.9330 77.6582 73.5850
##
##  Upper 95 Percent Confidence Limits
## price
##       5     5.5       6
## 84.6849 80.0518 76.2172
```

Figure 5.1 shows the predicted levels of the dependent variable *sales* and its 95% confidence band for the sample values of the variable *price*. In more complex functional forms, the *R* function `effect()` plots the partial effect of a variable for given levels of the other independent variables in the model. The simplest possible call of this function requires, as arguments, the name of the term for which we wish the partial effect (in our example *price*), and the object (model) in question. If not otherwise specified, the confidence intervals (band) are determined for a 95% confidence level and the other variables at their means. A simple use of the `effects` package is presented in Figure 5.2, which plots the partial effects of all variables in the basic *andy* model. (Function `effect()` plots only one graph for one variable, while `allEffects()` plots all variables in the model.)

```
alleffandy <- allEffects(mod1)
plot(alleffandy)
```

```
# Another example of using the function effect()
mod2 <- lm(sales~price+advert+I(advert^2), data=andy)
summary(mod2)
```

```
##
## Call:
## lm(formula = sales ~ price + advert + I(advert^2), data = andy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

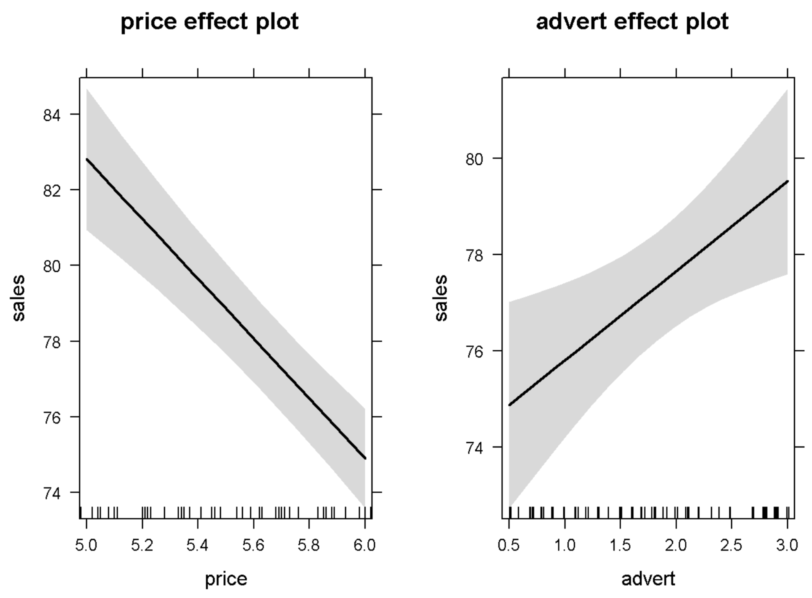


Figure 5.2: Using the function 'allEffects()' in the basic *andy* model

```
## -12.255  -3.143  -0.012   2.851  11.805
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  109.719      6.799   16.14  < 2e-16 ***
## price         -7.640      1.046   -7.30  3.2e-10 ***
## advert        12.151      3.556    3.42   0.0011 **
## I(advert^2)   -2.768      0.941   -2.94   0.0044 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.65 on 71 degrees of freedom
## Multiple R-squared:  0.508,  Adjusted R-squared:  0.487
## F-statistic: 24.5 on 3 and 71 DF,  p-value: 5.6e-11
```

```
plot(effect("I(advert^2)", mod2))
```

Figure 5.3 is an example of using the `effect()` function to plot the partial effect of a quadratic independent variable. I chose to insert the `I(advert^2)` term to indicate that the variable of interest needs to be specified exactly as it appears in the model.

All the methods available in *R* for simple linear regression models are available for multiple models as well. Thus, to extract the information generated by the `lm()`

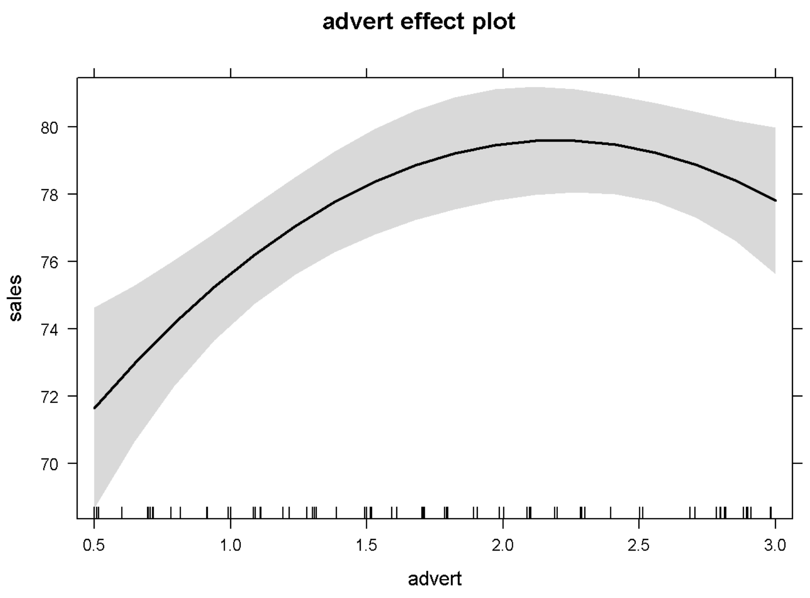


Figure 5.3: An example of using the function 'effect' in a quadratic model

function, we use the similar code as before. As we have already learned, the command `?mod1` gives a list with all the information stored in the `mod1` object. The next code sequence illustrates how to access various regression output items for the basic *andy* equation.

```
mod1 <- lm(sales~price+advert, data=andy)
smod1 <- summary(mod1)
df <- mod1$df.residual
N <- nobs(mod1)
b1 <- coef(mod1)[[1]] #or
b2 <- coef(mod1)[["price"]]
b3 <- coef(mod1)[["advert"]]
sighat2 <- smod1$sigma^2
anov <- anova(mod1)
SSE <- anov[3,2]
SST <- sum(anov[,2]) #sum of column 2 in anova
SSR <- SST-SSE
kable(data.frame(vcov(mod1)), align='c', digits=3,
      caption="The coefficient covariance matrix",
      col.names=c("(Intercept)", "price", "advert"))
```

The *covariance matrix*, or the *variance-covariance* matrix shown in Table 5.3 contains all the estimated variances and covariances of the regression coefficients. These are

Table 5.3: The coefficient covariance matrix

	(Intercept)	price	advert
(Intercept)	40.343	-6.795	-0.748
price	-6.795	1.201	-0.020
advert	-0.748	-0.020	0.467

useful when testing hypotheses about individual coefficients or combinations of those.

5.3 Interval Estimation in Multiple Regression

Interval estimation is similar with the one we have studied in the simple regression model, except the number of degrees of freedom is now $N - K$, where K is the number of regression coefficients to be estimated. In the *andy* example, $df = 72$ and the variances and covariances of the estimates are given in the following code sequence.

```
varb1 <- vcov(mod1)[1,1]
varb2 <- vcov(mod1)[2,2]
varb3 <- vcov(mod1)[3,3]
covb1b2 <- vcov(mod1)[1,2]
covb1b3 <- vcov(mod1)[1,3]
covb2b3 <- vcov(mod1)[2,3]
seb2 <- sqrt(varb2) #standard error of b2
seb3 <- sqrt(varb3)
```

With the calculated standard error of b_2, $se(b_2) = 1.096$, we can now determine a 95% confidence interval, as shown in the following code lines. The code also shows using the *R* function `confint(model, parm, level)` to check our results.

```
alpha <- 0.05
tcr <- qt(1-alpha/2, df)
lowb2 <- b2-tcr*seb2
upb2 <- b2+tcr*seb2
lowb3 <- b3-tcr*seb3
upb3 <- b3+tcr*seb3
confints <- confint(mod1, parm=c("price", "advert"), level=0.95)
kable(data.frame(confints),
  caption="Confidence intervals for 'price' and 'advert'",
  align="c", col.names=c("lowb", "upb"), digits=4)
```

Finding an interval estimate for a linear combination of the parameters is often

Table 5.4: Confidence intervals for 'price' and 'advert'

	lowb	upb
price	-10.0927	-5.7230
advert	0.5007	3.2245

needed. Suppose one is interested in determining an interval estimate for *sales* when the price decreases by 40 cents and the advertising expenditure increases by $800 (see Equation 5.3) in the *andy* basic equation.

$$\lambda = 0\beta_1 - 0.4\beta_2 + 0.8\beta_3 \tag{5.3}$$

```
alpha <- 0.1
tcr <- qt(1-alpha/2, df)
a1 <- 0
a2 <- -0.4
a3 <- 0.8
L <- a1*b1+a2*b2+a3*b3
varL <- a1^2*varb1+a2^2*varb2+a3^2*varb3+
  2*a1*a2*covb1b2+2*a1*a3*covb1b3+2*a2*a3*covb2b3
seL <- sqrt(varL)
lowbL <- L-tcr*seL
upbL <- L+tcr*seL
```

The calculated confidence interval is $(3.471, 5.836)$. Let us calculate the variance of a linear combination of regression parameters in a more general way to take advantage of *R*'s excellent capabilities of working with complex data structures such as lists, vectors, and matrices. This code sequence introduces a few new elements, such as matrix transposition and multiplication, as well as turning a list into a vector. The matrix multiplication operator in *R* is `%*%` and the transposition operator is `t()`.

```
a <- c(0, -0.4, 0.8) # vector
b <- as.numeric(coef(mod1))# vector of coefficients
L <- L <- sum(a*b) # sum of elementwise products
V <- vcov(mod1) # the variance-covariance matrix
A <- as.vector(a) # (indeed not necessary)
varL <- as.numeric(t(A) %*% V %*% A)
```

5.4 Hypothesis Testing in Multiple Regression

The process of testing hypotheses about a single parameter is similar to the one we've seen in simple regression, the only difference consisting in the number of degrees of freedom. As an example, let us test the significance of β_2 of the basic *andy* equation. The hypotheses are given in Equation 5.4.

$$H_0 : \beta_2 = 0, \quad H_A : \beta_2 \neq 0 \tag{5.4}$$

```
alpha <- 0.05
df <- mod1$df.residual
tcr <- qt(1-alpha/2, df)
b2 <- coef(mod1)[["price"]]
seb2 <- sqrt(vcov(mod1)[2,2])
t <- b2/seb2
```

The calculated t is equal to -7.215, which is less than $-t_{cr} = -1.993$, indicating that the null hypothesis is rejected and that β_2 is significantly different from zero. As usual, we can perform the same test using the p-value instead of t_{cr}.

```
t <- b2/seb2
pval <- 2*(1-pt(abs(t), df)) #two-tail test
```

The calculated p-value is 4.423999×10^{-10}. Since this is less than α we reject, again, the null hypothesis $H_0 : \beta_2 = 0$. Let us do the same for β_3:

```
alpha <- 0.05
df <- mod1$df.residual
tcr <- qt(1-alpha/2, df)
b3 <- coef(mod1)[[3]]
seb3 <- sqrt(vcov(mod1)[3,3])
tval <- b3/seb3
```

Calculated $t = 2.726$, and $t_{cr} = 1.993$. Since $t > t_{cr}$ we reject the null hypothesis and conclude that there is a statistically significant relationship between *advert* and *sales*. The same result can be obtained using the p-value method:

```
pval <- 2*(1-pt(abs(tval), df))
```

Result: p-value $= 0.008038 < \alpha$. *R* shows the two-tail t and p values for coefficients in regression output (see Table 5.2).

A one-tail hypothesis testing can give us information about price elasitcity of demand in the basic *andy* equation in Table 5.2. Let us test the hypothesis described in

Equation 5.5 at a 5% significance level. The calculated value of t is the same, but t_{cr} corresponds now not to $\frac{\alpha}{2}$ but to α.

$$H_0 : \beta_2 \geq 0, \quad H_A : \beta_2 < 0 \tag{5.5}$$

```
alpha <- 0.05
tval <- b2/seb2
tcr <- -qt(1-alpha, df) #left-tail test
pval <- pt(tval, df)
```

The results $t = -7.215$, $t_{cr} = -1.666$ show that the calculated t falls in the (left-tail) rejection region. The p-value, which is equal to 2.211999×10^{-10}, is less than α, also rejects the null hypothesis.

Here is how the p-values should be calculated, depending on the type of test:

- Two-tail test ($H_A : \beta_2 \neq 0$), `p-value <- 2*(1-pt(abs(t), df))`
- Left-tail test ($H_A : \beta_2 < 0$), `p-value <- pt(t, df)`
- Right-tail test ($H_A : \beta_2 > 0$), `p-value <- 1-pt(t, df)`

Another example of a one-tail hypothesis testing is to test whether an increase of \$1 in advertising expenditure increases revenue by more than \$1. In the hypothesis testing language, we need to test the hypothesis presented in Equation 5.6.

$$H_0 : \beta_3 \leq 1, \quad H_A : \beta_3 > 1 \tag{5.6}$$

```
tval <- (b3-1)/seb3
pval <- 1-pt(tval,df)
```

The calculated p-value is 0.105408, which is greater than α, showing that we cannot reject the null hypothesis $H_0 : \beta_3 \leq 1$ at $\alpha = 0.05$. In other words, increasing advertising expenditure by \$1 may or may not increase sales by more than \$1.

Testing hypotheses for linear combinations of coefficients resambles the interval estimation procedure. Suppose we wish to determine if lowering the price by 20 cents increases sales more than would an increase in advertising expenditure by \$500. The hypothesis to test this conjecture is given in Equation 5.7.

$$H_0 : 0\beta_1 - 0.2\beta_2 - 0.5\beta_3 \leq 0, \quad H_A : 0\beta_1 - 0.2\beta_2 - 0.5\beta_3 > 0 \tag{5.7}$$

Let us practice the matrix form of testing this hypotesis. R functions having names that start with `as.` coerce a certain structure into another, such as a named list into a vector (names are removed and only numbers remain).

```r
A <- as.vector(c(0, -0.2, -0.5))
V <- vcov(mod1)
L <- as.numeric(t(A) %*% coef(mod1))
seL <- as.numeric(sqrt(t(A) %*% V %*% A))
tval <- L/seL
pval <- 1-pt(tval, df) # the result (p-value)
```

The answer is p-value= 0.054619, which barely fails to reject the null hypothesis. Thus, the conjecture that a decrease in price by 20 cents is more effective than an increase in advertising by $800 is not supported by the data.

For two-tail linear hypotheses, R has a built-in function, `linearHypothesis(model, hypothesis)`, in the package `car`. This function tests hypotheses based not on t-statistics as we have done so far, but based on an F-statistic. However, the p-value criterion to reject or not the null hypothesis is the same: reject if p-value is less than α. Let us use this function to test the two-tail hypothesis similar to the one given in Equation 5.7. (Note that the `linearHypothesis()` function can test not only one hypothesis, but a set of simultaneous hypotheses.)

```r
hypothesis <- "-0.2*price = 0.5*advert"
test <- linearHypothesis (mod1, hypothesis)
Fstat <- test$F[2]
pval <- 1-pf(Fstat, 1, df)
```

The calculated p-value is 0.109238, which shows that the null hypothesis cannot be rejected. There are a few new elements, besides the use of the `linearHypothesis` function in this code sequence. First, the `linearHypothesis()` function creates an R object that contains several items, one of which is the F-statistic we are looking for. This object is shown in Table 5.5 and it is named `test` in our code. The code element `test$F[2]` extracts the F-statistic from the `linearHypothesis()` object.

Second, please note that the F-statistic has two 'degrees of freedom' parameters, not only one as the t-statistic does. The first degree of freedom is equal to the number of simultaneous hypotheses to be tested (in our case only one); the second is the number of degrees of freedom in the model, $df = N - K$.

Last, the function that calculates the p-value is `pf(Fval, df1, df2)`, where `Fval` is the calculated value of the F-statistic, and `df1` and `df2` are the two degrees of freedom parameters of the F-statistic.

Table 5.5: The ‘linearHypothesis()‘ object

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
73	1781.73	NA	NA	NA	NA
72	1718.94	1	62.7874	2.62993	0.109238

Table 5.6: The quadratic version of the *andy* model

	Estimate	Std. Error	t	p-Value
(Intercept)	109.719	6.799	16.137	0.000
price	-7.640	1.046	-7.304	0.000
advert	12.151	3.556	3.417	0.001
I(advert^2)	-2.768	0.941	-2.943	0.004

```r
kable(test, caption="The `linearHypothesis()` object")
```

5.5 Polynomial Regression Models

A polynomial multivariate regression model may include several independent variables at various powers. In such models, the partial (or marginal) effect of a regressor x_k on the response y is determined by the partial derivative $\frac{\partial y}{\partial x_k}$. Let us consider again the basic *andy* model with the added *advert* quadratic term as presented in Equation 5.8.

$$sales_i = \beta_1 + \beta_2 price_i + \beta_3 advert_i + \beta_4 advert_i^2 + e_i \tag{5.8}$$

As we have noticed before, the quadratic term is introduced into the model using the function `I()` to indicate the presence of a mathematical operator. Table 5.6 presents a summary of the regression output from the model described in Equation 5.8.

```r
mod2 <- lm(sales~price+advert+I(advert^2),data=andy)
smod2 <- summary(mod2)
tabl <- data.frame(xtable(smod2))
names(tabl) <- c("Estimate",
                 "Std. Error", "t", "p-Value")
kable(tabl, digits=3, align='c',
  caption="The quadratic version of the $andy$ model")
```

Let us calculate the marginal effect of *advert* on *sales* for two levels of *advert*; the relevant partial derivative is the one in Equation 5.9.

$$\frac{\partial sales}{\partial advert} = \beta_3 + 2\beta_4 advert \tag{5.9}$$

```
advlevels <- c(0.5, 2)
b3 <- coef(mod2)[[3]]
b4 <- coef(mod2)[[4]]
DsDa <- b3+b4*advlevels
```

The calculated marginal effects for the two levels of advertising expenditure are, respectively, 10.767254 and 6.615309, which shows, as expected, diminishing returns to advertising at any given price.

Often, marginal effects or other quantities are non-linear functions of the parameters of a regression model. The *delta* method allows calculating the variance of such quantities using a Taylor series approximation. This method is, howver, valid only in a vicinity of a point, as any mathematical object involving derivatives.

Suppose we wish to test a hypothesis such as the one in Equation 5.10, where c is a constant.

$$H_0 : g(\beta_1, \beta_2) = c \tag{5.10}$$

The *delta* method consists in estimating the variance of the function $g(\beta_1, \beta_2)$ around some given data point, using the formula in Equation 5.11, where g_i stands for $\frac{\partial g}{\partial \beta_i}$ calculated at the point $\beta = b$.

$$var(g) = g_1^2 var(b_1) + g_2^2 var(b_2) + 2 g_1 g_2 cov(b_1, b_2) \tag{5.11}$$

Let us apply this method to find a confidence interval for the optimal level of advertising in the *andy* quadratic equation, g, which is given by Equation 5.12.

$$\hat{g} = \frac{1 - b_3}{2b_4} \tag{5.12}$$

Equation 5.13 shows the derivatives of function g.

$$\hat{g}_3 = -\frac{1}{2b_4}, \quad \hat{g}_4 = -\frac{1 - b_3}{2b_4^2}. \tag{5.13}$$

```
alpha <- 0.05
df <- mod2$df.residual
tcr <- qt(1-alpha/2, df)
g <- (1-b3)/(2*b4)
g3 <- -1/(2*b4)
g4 <- -(1-b3)/(2*b4^2)
varb3 <- vcov(mod2)[3,3]
varb4 <- vcov(mod2)[4,4]
covb3b4 <- vcov(mod2)[3,4]
varg <- g3^2*varb3+g4^2*varb4+2*g3*g4*covb3b4
seg <- sqrt(varg)
lowbg <- g-tcr*seg
upbg <- g+tcr*seg
```

The point estimate of the optimal advertising level is 2.014, with its confidence interval (1.758, 2.271).

5.6 Interaction Terms in Linear Regression

Interaction terms in a regression model combine two or more variables and model interdependences among the variables. In such models, the slope of one variable may depend on another variable. Partial effects are calculated as partial derivatives, similar to the polynomial equations we have already studied, considering the interaction term a a product of the variables involved.

The data file *pizza4* includes 40 observations on an individual's age and his or her spending on pizza. It is natural to believe that there is some connection between a person's age and her pizza purchases, but also that this effect may depend on the person's income. Equation 5.14 combines *age* and *income* in an interaction term, while Equation 5.15 shows the marginal effect of *age* on the expected pizza purchases.

$$pizza = \beta_1 + \beta_2 age + \beta_3 income + \beta_4 (age \times income) + e \tag{5.14}$$

$$\frac{\widehat{\partial pizza}}{\partial age} = b_2 + b_4 income \tag{5.15}$$

Let us calculate the marginal effect of *age* on *pizza* for two levels of income, \$25 000 and \$90 000. There are two ways to introduce interaction terms in *R*; first, with a : (colon) symbol, when only the interaction term is created; second, with the * (star) symbol when all the terms involving the variables in the interaction term should be present in the regression. For instance, the code $x * z$ creates three terms: x, z, and $x : z$ (this last one is x 'interacted' with z, which is a peculiarity of the R system).

```
data("pizza4",package="PoEdata")
mod3 <- lm(pizza~age*income, data=pizza4)
summary(mod3)
```

```
##
## Call:
## lm(formula = pizza ~ age * income, data = pizza4)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -200.9  -83.8   20.7   85.0  254.2
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 161.4654   120.6634    1.34    0.189
## age          -2.9774     3.3521   -0.89    0.380
## income        6.9799     2.8228    2.47    0.018 *
## age:income   -0.1232     0.0667   -1.85    0.073 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 127 on 36 degrees of freedom
## Multiple R-squared:  0.387,  Adjusted R-squared:  0.336
## F-statistic: 7.59 on 3 and 36 DF,  p-value: 0.000468
```

```
inc <- c(25, 90)
b2 <- coef(mod3)[[2]]
b4 <- coef(mod3)[[4]]
(DpDa <- b2+b4*inc)
```

```
## [1]  -6.05841 -14.06896
```

The marginal effect of *age* is a reduction in pizza expenditure by \$6.058407 for an income of \$25 000 and by \$14.068965 for an income of \$90 000.

Equation 5.16 is another example of a model involving an interaction term: the *wage* equation. The equation involves, besides the interaction term, logs of the response (dependent) variable is in logs and a quadratic term.

$$log(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 (educ \times exper) + \beta_5 exper^2 + e \qquad (5.16)$$

Table 5.7: Wage equation with interaction and quadratic terms

	Estimate	Std..Error	t.value	Pr...t..
(Intercept)	0.529677	0.226741	2.33604	0.019687
educ	0.127195	0.014719	8.64167	0.000000
exper	0.062981	0.009536	6.60447	0.000000
I(exper^2)	-0.000714	0.000088	-8.10911	0.000000
educ:exper	-0.001322	0.000495	-2.67222	0.007658

The marginal effect of an extra year of experience is determined, again, by the partial derivative of $log(wage)$ with respect to $exper$, as Equation 5.17 shows. Here, however, the marginal effect is easier expressed in percentage change, rather than in units of $wage$.

$$\%\Delta wage \cong 100\,(\beta_3 + \beta_4 educ + 2\beta_5 exper)\Delta exper \tag{5.17}$$

Please note that the marginal effect of $exper$ depends on both education and experience. Let us calculate this marginal effect for the average values of $educ$ and $exper$.

```
data(cps4_small)
meduc <- mean(cps4_small$educ)
mexper <- mean(cps4_small$exper)
mod4 <- lm(log(wage)~educ*exper+I(exper^2), data=cps4_small)
smod4 <- data.frame(xtable(summary(mod4)))
b3 <- coef(mod4)[[3]]
b4 <- coef(mod4)[[4]]
b5 <- coef(mod4)[[5]]
pDwDex <- 100*(b3+b4*meduc+2*b5*mexper)
```

The result of this calculation is $\%\Delta wage = -1.698$, which has, apparently, the wrong sign. What could be the cause? Let us look at a summary of the regression output and identify the terms (see Table 5.7).

```
kable(smod4, caption="Wage equation with interaction and quadratic terms")
```

The output table shows that the order of the terms in the regression equation is not the same as in Equation 5.16, which is the effect of using the compact term $educ*exper$ in the R code. This places the proper interaction term, $educ : exper$ in the last position of all terms containing any of the variables involved in the combined term $educ*exper$. There are a few ways to solve this issue. One is to write the equation code in full, without the shortcut $educ * exper$; another is to use the names of the terms when

Table 5.8: Anova table for the basic *andy* model

	Df	Sum.Sq	Mean.Sq	F.value	Pr..F.
price	1	1219.091	1219.0910	51.06310	0.000000
advert	1	177.448	177.4479	7.43262	0.008038
Residuals	72	1718.943	23.8742	NA	NA

retrieving the coefficients, such as `b4 <- coef(mod1)[["I(exper^2)"]]`; another to change the position of the terms in Equation 5.16 according to Table 5.7 and re-calculate the derivative in Equation 5.17. I am going to use the names of the variables, as they appear in Table 5.7, but I also change the names of the parameters to avoid confusion.

```
bexper <- coef(mod4)[["exper"]]
bint <- coef(mod4)[["educ:exper"]]
bsqr <- coef(mod4)[["I(exper^2)"]]
pDwDex <- 100*(bexper+bint*meduc+2*bsqr*mexper)
```

The new result indicates that the expected wage increases by 0.68829 percent when experience increases by one year, which, at least, has the “right” sign.

5.7 Goodness-of-Fit in Multiple Regression

The coefficient of determination, R^2, has the same formula (Equation 5.18)and interpretation as in the case of the simple regression model.

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{5.18}$$

As we have already seen, the *R* function `anova()` provides a decomposition of the total sum of squares, showing by how much each predictor contributes to the reduction in the residual sum of squares. Let us look at the basic *andy* model described in Equation 5.2.

```
mod1 <- lm(sales~price+advert, data=andy)
smod1 <- summary(mod1)
Rsq <- smod1$r.squared
anov <- anova(mod1)
dfr <- data.frame(anov)
kable(dfr, caption="Anova table for the basic *andy* model")
```

Table 5.8 shows that *price* has the largest contribution to the reduction of variablity in the residuals; since total sum of squares is $SST = 3115.482$ (equal to the sum of column `Sum.Sq` in Table 5.8), the portion explained by *price* is 1219.091, the one explained by *advert* is 177.448, and the portion still remaining in the residuals is $SSE = 1718.943$. The portion of variability explained by regression is equal to the sum of the first two items in the `Sum.Sq` column of the `anova` table, those corresponding to *price* and *advert*: $SSR = 1396.539$.

RStudio displays the values of all named variables in the `Environment` window (the NE panel of your screen). Please check, for instance, that R^2 (named `Rsq` in the previous code sequence) is equal to 0.448.

Chapter 6

Further Inference in Multiple Regression

```r
rm(list=ls())
library(PoEdata) #for PoE datasets
library(knitr) #for referenced tables with kable()
library(xtable) #makes data frame for kable
library(printr) #automatically prints output nicely
library(effects)
library(car)
library(AER)
library(broom) #for tidy lm output and function glance()
library(stats)
```

New package: `broom` (Robinson 2016).

6.1 Joint Hypotheses and the F-statistic

A joint hypothesis is a set of relationships among regression parameters, relationships that need to be simultaneously true according to the null hypothesis. Joint hypotheses can be tested using the F-statistic that we have already met. Its formula is given by Equation 6.1. The F-statistic has an F distribution with the degrees of freedom J and $N - K$.

$$F = \frac{(SSE_R - SSE_U)\,/\,J}{SSE_U\,/\,(N-K)} \sim F_{(J,\,N-K)} \tag{6.1}$$

In Equation 6.1 the subscript U stands for "unrestricted," that is, the initial regression equation; the "restricted" equation is a new equation, obtained from the initial one, with the relationships in the null hypothesis assumed to hold. For example, if the initial equation is Equation 6.2 and the null hypothesis is Equation 6.3, then the restricted equation is Equation 6.4.

$$y = \beta_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + e \tag{6.2}$$

$$H_0 : \beta_2 = 0 \;\; AND \;\; \beta_3 = 0; \quad H_A : \beta_2 \neq 0 \;\; OR \;\; \beta_3 \neq 0 \tag{6.3}$$

$$y = \beta_1 + \beta_4 x_4 + e \tag{6.4}$$

The symbol J in the F formula (Equation 6.1) is the first (*numerator*) degrees of freedom of the F statistic and is equal to the number of simultaneous restrictions in the null hypothesis (Equation 6.3); the second (the *denominator*) degrees of freedom of the F-statistic is $N - K$, which is the usual degrees of freedom of the unrestricted regression model (Equation 6.2). The practical procedure to test a joint hypothesis like the one in Equation 6.3 is to estimate the two regressions (unrestricted and restricted) and to calculate the F-statistic.

6.2 Testing Simultaneous Hypotheses

Let's look, again, at the quadratic form of the *andy* equation (Equation 6.5).

$$sales = \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \tag{6.5}$$

Equation 6.5 has two terms that involve the regressor *advert*, of which at least one needs to be significant for a relationship between *advert* and *sales* to be established. To test if such a relationship exists, we can formulate the following test:

$$H_0 : \beta_3 = 0 \;\; AND \;\; \beta_4 = 0; \quad H_A : \beta_3 \neq 0 \;\; OR \;\; \beta_4 \neq 0 \tag{6.6}$$

I have already mentioned that R can do an F test quite easily (remember the function `linearHypothesis`?), but for learning purposes let us calculate the F-statistic in steps. The next code sequence uses information in the `anova`-type object, which, remember, can be visualized simply by typing the name of the object in the RStudio's `Console` window.

```r
alpha <- 0.05
data("andy", package="PoEdata")
N <- NROW(andy) #Number of observations in dataset
K <- 4 #Four Betas in the unrestricted model
J <- 2 #Because Ho has two restrictions
fcr <- qf(1-alpha, J, N-K)
mod1 <- lm(sales~price+advert+I(advert^2), data=andy)
anov <- anova(mod1)
anov # prints 'anova' table for the unrestricted model
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
price	1	1219.091	1219.0910	56.49523	0.000000
advert	1	177.448	177.4479	8.22331	0.005441
I(advert^2)	1	186.858	186.8585	8.65941	0.004393
Residuals	71	1532.084	21.5787	NA	NA

```r
SSEu <- anov[4, 2]
mod2 <- lm(sales~price, data=andy) # restricted
anov <- anova(mod2)
anov # prints the 'anova' table for the restrictred model
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
price	1	1219.09	1219.091	46.9279	0
Residuals	73	1896.39	25.978	NA	NA

```r
SSEr <- anov[2,2]
fval <- ((SSEr-SSEu)/J) / (SSEu/(N-K))
pval <- 1-pf(fval, J, N-K)
```

The calculated F-statistic is $fval = 8.441$ and the critical value corresponding to a significance level $\alpha = 0.05$ is 3.126, which rejects the null hypothesis that both β_3 and β_4 are zero. The p-value of the test is $p = 0.0005$.

Using the `linearHypothesis()` function should produce the same result:

```r
Hnull <- c("advert=0", "I(advert^2)=0")
linearHypothesis(mod1,Hnull)
```

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
73	1896.39	NA	NA	NA	NA
71	1532.08	2	364.306	8.44136	0.000514

The table generated by the `linearHypothesis()` function shows the same values of the F-statistic and p-value that we have calculated before, as well as the residual sum of squares for the restricted and unrestricted models. Please note how I formulate the joint hypothesis as a vector of character values in which the names of the variables perfectly match those in the unrestricted model.

Testing the overall significance of a model amounts to testing the joint hypothesis that all the slope coefficients are zero. *R* does automatically this test and the resulting F-statistic and p-value are reported in the regression output.

```
summary(mod1)
```

```
##
## Call:
## lm(formula = sales ~ price + advert + I(advert^2), data = andy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.255  -3.143  -0.012   2.851  11.805
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  109.719      6.799   16.14  < 2e-16 ***
## price         -7.640      1.046   -7.30  3.2e-10 ***
## advert        12.151      3.556    3.42   0.0011 **
## I(advert^2)   -2.768      0.941   -2.94   0.0044 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.65 on 71 degrees of freedom
## Multiple R-squared:  0.508,  Adjusted R-squared:  0.487
## F-statistic: 24.5 on 3 and 71 DF,  p-value: 5.6e-11
```

The F-statistic can be retrieved from `summary(mod1)` or by using the function `glance(modelname)` in package `broom`, as shown in the following code lines. The function `tidy`, also from package `broom` organizes regression output (mainly the coefficients and their statistics) in a neat table. Both `glance` and `tidy` create output in the form of `data.frame`, which makes it suitable for use by other functions such as `kable` and `ggplot2`. Please also note that `tidy(mod1)` and `tidy(summary(mod1))` produce the same result, as shown in Tables 6.1 and 6.2. As always, we can use the function `names` to obtain a list of the quantities available in the output of the `glance` function.

```
smod1 <- summary(mod1)
kable(tidy(smod1), caption="Tidy 'summary(mod1)' output")
```

```
fval <- smod1$fstatistic
```

Table 6.1: Tidy 'summary(mod1)' output

term	estimate	std.error	statistic	p.value
(Intercept)	109.71904	6.799045	16.13742	0.000000
price	-7.64000	1.045939	-7.30444	0.000000
advert	12.15124	3.556164	3.41695	0.001052
I(advert^2)	-2.76796	0.940624	-2.94269	0.004393

Table 6.2: 'Tidy(mod1)' output

term	estimate	std.error	statistic	p.value
(Intercept)	109.71904	6.799045	16.13742	0.000000
price	-7.64000	1.045939	-7.30444	0.000000
advert	12.15124	3.556164	3.41695	0.001052
I(advert^2)	-2.76796	0.940624	-2.94269	0.004393

```r
library(broom)
kable(tidy(mod1), caption="'Tidy(mod1)' output")
```

```r
glance(mod1)$statistic #Retrieves the F-statistic
```

```
## [1] 24.4593
```

```r
names(glance(mod1)) #Shows what is available in 'glance'
```

```
##  [1] "r.squared"     "adj.r.squared" "sigma"         "statistic"
##  [5] "p.value"       "df"            "logLik"        "AIC"
##  [9] "BIC"           "deviance"      "df.residual"
```

```r
kable(glance(mod1),
 caption="Function 'glance(mod1)' output", digits=2,
 col.names=(c("Rsq","AdjRsq","sig","F",
 "pF","K","logL","AIC","BIC","dev","df.res")))
```

Table 6.3 shows a summary of the quadratic *andy* model (`mod1`), where I have changed the names of various items so that the table fits the width of the page.

Table 6.3: Function 'glance(mod1)' output

Rsq	AdjRsq	sig	F	pF	K	logL	AIC	BIC	dev	df.res
0.51	0.49	4.65	24.46	0	4	-219.55	449.11	460.7	1532.08	71

Table 6.4: Joint hypotheses with the 'linearHypothesis' function

res.df	rss	df	sumsq	statistic	p.value
73	1779.86	NA	NA	NA	NA
71	1532.08	2	247.776	5.74123	0.004885

When retrieving these variables, make sure you use the original names as indicated by the `names(glance(mod1))` command.

When testing a two-tail single (not joint) null hypothesis, the t and F tests are equivalent. However, one-tail tests, single or joint cannot be easily performed by an F test.

Let us solve one more exercise involving a joint hypothesis with linear combinations of regression coefficients. Suppose we want to test the simultaneous hypotheses that the monthly advertising expenditure $advert_0 = \$1900$ in the quadratic *andy* model (Equation 6.5) satisfies the profit-maximizing condition $\beta_3 + 2\beta_4 advert_0 = 1$, and that, when $price = \$6$ and $advert = \$1900$ sales revenue is \$80 000.

```
hyp <- c("advert+3.8*I(advert^2)=1",
"(Intercept)+6*price+1.9*advert+3.61*I(advert^2)=80")
lhout <- tidy(linearHypothesis(mod1,hyp))
kable(lhout,
 caption="Joint hypotheses with the 'linearHypothesis' function")
```

Table 6.4 includes the F-statistic of the test, $F = 5.741229$ and its p-value $p = 0.004885$. Please be aware that the function `tidy` changes the names in the output of `linearHypothesis`. A useful exercise is to compare the raw output of `linearHypothesis` with the output generated by `tidy(linearHypothesis(mod1))`. There are several other possibilities to compare two regression models, such as a restricted and unrestricted ones in R, such as the `anova()` function or Wald tests. These are going to be mentioned in later chapters.

6.3 Omitted Variable Bias

Consider the general model with two regressors in Equation 6.7, a model that I will call the *true* model.

$$y = \beta_1 + \beta_2 x_2 + \beta_3 x_3 + e \tag{6.7}$$

Suppose we are only interested in estimating β_2, but there is no data available for x_3, or for other reasons x_3 is omitted from the model in Equation 6.7. What is the

Table 6.5: The correct model

term	estimate	std.error	statistic	p.value
(Intercept)	-5533.63	11229.533	-0.492775	0.622426
he	3131.51	802.908	3.900209	0.000112
we	4522.64	1066.327	4.241328	0.000027

error in the estimate of β_2 introduced by omitting x_3? Equation 6.8 shows what is left of the *true* model after omitting x_3.

$$y = \beta_1 + \beta_2 x_2 + u \tag{6.8}$$

Let b_2^* be the estimate of β_2 when x_3 is omitted. Equation 6.9 gives the bias in this simple, two-regressor case. The formula shows that bias depends on the direct relationship between the omitted regressor and response through β_3, as well as the correlation between the omitted and the included regressors. When β_3 and $cov(x_2, x_3)$ are both positive or both negative the bias is positive (the incorrect model overestimates the true β_2), and when they are of opposite signs the bias is negative (β_2 is underestimated)

$$bias(b_2^*) = E(b_2)^* - \beta_2 = \beta_3 \frac{\widehat{cov(x_2, x_3)}}{var(x_2)} \tag{6.9}$$

The example in this section uses the dataset *edu_inc*, and two models: one where the response variable *family income* ($faminc$) is explained by the regressors *husband's education* (he) and *wife's education* (we), and another model, where the we regressor is omitted. The purpose is to compare the estimates coming from the two models and see if there is a significant difference between them.

```r
data("edu_inc", package="PoEdata")
mod1 <- lm(faminc~he+we, data=edu_inc)
mod2 <- lm(faminc~he, data=edu_inc)
kable(tidy(mod1), caption="The correct model")
```

```r
kable(tidy(mod2),
  caption="The incorrect model ('we' omitted)")
```

The marginal effect of husband's education is much lower in the incorrect model. Let us apply the logic of Equation 6.9 to the *edu_inc* model. The direct effect of the omitted regressor (we) on response ($faminc$) is likely to be positive in theory (higher education generates higher income); the correlation between husband's and wife's education is also likely to be positive if we believe that people generally marry

Table 6.6: The incorrect model ('we' omitted)

term	estimate	std.error	statistic	p.value
(Intercept)	26191.27	8541.108	3.06650	0.002304
he	5155.48	658.457	7.82964	0.000000

Table 6.7: Correct 'faminc' model

term	estimate	std.error	statistic	p.value
(Intercept)	-7755.33	11162.935	-0.694739	0.487599
he	3211.53	796.703	4.031022	0.000066
we	4776.91	1061.164	4.501574	0.000009
kl6	-14310.92	5003.928	-2.859937	0.004447

persons within their entourage. Thus, we should expect that omitting the regressor *we* should produce an overestimated marginal effect of *he*. Our data happen to confirm this supposition, though there is some chance that they might not.

Understanding the problem of omitted variable is very important because it can justify the choice of a particular model. If one is not interested in the effect of variable x_3 on y and can convince that x_3 is uncorrelated with x_2, one can argue with criticism about omitting the important regressor x_3.

6.4 Irrelevant Variables

We have seen the effect of omitting a relevant regressor (the effect is biased estimates and lower variances of the included regressors). But what happens if irrelevant variables are incorrectly included? Not surprisingly, this increases the variances (lowers the precision) of the other variables in the model. The next example uses the same (*edu_inc*) dataset as above, but includes two artificially generated variables, $xtra\_x5$ and $xtra\_x6$ that are correlated with *he* and *we* but, obviously, have no role in determining y. Let us compare two models, of which one includes these irrelevant variables.

```r
mod3 <- lm(faminc~he+we+kl6, data=edu_inc)
mod4 <- lm(faminc~he+we+kl6+xtra_x5+xtra_x6, data=edu_inc)
kable(tidy(mod3), caption="Correct 'faminc' model")
```

```r
kable(tidy(mod4),
 caption="Incorrect 'faminc' with irrelevant variables")
```

Table 6.8: Incorrect 'faminc' with irrelevant variables

term	estimate	std.error	statistic	p.value
(Intercept)	-7558.613	11195.41	-0.675153	0.499948
he	3339.792	1250.04	2.671750	0.007838
we	5868.677	2278.07	2.576165	0.010329
kl6	-14200.184	5043.72	-2.815419	0.005100
xtra_x5	888.843	2242.49	0.396364	0.692037
xtra_x6	-1067.186	1981.69	-0.538524	0.590499

A comparison of the two models shown in Tables 6.7 and 6.8 indicates that the inclusion of the two irrelevant variables has increased the marginal effects, standard errors, and the p-values of he and we. Thus, including irrelevant variables may incorrectly diminish the significance of the "true" regressors.

6.5 Model Selection Criteria

The main tools of building a model should be economic theory, sound reasoning based on economic principles, and making sure that the model satisfies the Gauss-Markov assumptions. One should also consider the possibility of omitted variable bias and the exclusion of irrelevant variables that may increase variablility in the estimates. After all these aspects have been considered and a model established, there are a few quantities that help comparing different models. These are R^2, adjusted R^2 ($\bar{R}^2$), the Akaike information criterion (AIC), and the Schwarz (or Bayesian information) criterion (SC or BIC).

We have already seen how to calculate the coefficient of determination, R^2 and how it measures the distance between the regression line and the observation points. A major disadvantage of R^2 is that it increases every time a new regressor is included in the model, whether the regressor is relevant or not. The idea of counterbalancing this property of R^2 has lead to a new measure, adjusted R^2, denoted by $\bar{R}^2$, given by Equation 6.10.

$$\bar{R}^2 = 1 - \frac{SSE\,/\,(N-K)}{SST\,/\,(N-1)} \tag{6.10}$$

Adjusted R^2, while addressing the problem with R^2, introduces other problems. In general, no single goodness of fit measure is perfect. The Akaike information criterion (AIC) and the Schwarz criterion use the same idea of penalizing the introduction of extra regressors. Their formulas are given in Equations 6.11 and 6.12.

$$AIC = ln\left(\frac{SSE}{N}\right) + \frac{2K}{N} \tag{6.11}$$

$$SC = ln\left(\frac{SSE}{N}\right) + \frac{K\,ln(N)}{N} \tag{6.12}$$

Among several models, the best fit is the one that maximizes R^2 or $\bar{R}^2$. On the contrary, the best model must **minimize** AIC or BIC. Some computer packages, R included, calculate AIC and BIC differentlly than Equations 6.11 and 6.12 indicate. However, the ranking of the various models is the same.

The following code sequence needs some explanation. Function `as.numeric` extracts only the numbers from an object such as `glance(mod1)`, which also contains row and column names. The purpose is to put together a table with information comming from several models. Function `rbind` gathers several rows in a matrix, which then is made into a `data.frame` and given the name `tab`. The part of code `[,c(1,2,8,9)]` at the end of `rbind` instructs R to pick all rows, but only columns `1, 2, 8, and 9` from the `glance` table. Function `row.names` assigns or changes the row names in a data frame; finally, `kable`, which we have already encountered several times, prints the table, assigns column names, and gives a caption to the table. While there are many ways to create a table in R, I use `kable` from package `knitr` because it allows me to cross-reference tables within this book. `kable` only works with data frames.

```
mod1 <- lm(faminc~he, data=edu_inc)
mod2 <- lm(faminc~he+we, data=edu_inc)
mod3 <- lm(faminc~he+we+kl6, data=edu_inc)
mod4 <- lm(faminc~he+we+kl6+xtra_x5+xtra_x6, data=edu_inc)
r1 <- as.numeric(glance(mod1))
r2 <- as.numeric(glance(mod2))
r3 <- as.numeric(glance(mod3))
r4 <- as.numeric(glance(mod4))
tab <- data.frame(rbind(r1, r2, r3, r4))[,c(1,2,8,9)]
row.names(tab) <- c("he","he, we","he, we, kl6",
                    "he, we, kl6, xtra_x5, xtra_x6")
kable(tab,
 caption="Model comparison, 'faminc' ", digits=4,
 col.names=c("Rsq","AdjRsq","AIC","BIC"))
```

Tabla 6.9 shows the four model selection criteria for four different models based on the *edu_inc* dataset, with the first column showing the variables included in each model. It is noticeable that three of the criteria indicate the third model as the best fit, while one, namely R^2 prefers the model that includes the irrelevant variables $xtra\_x5$ and $xtra\_x6$.

Table 6.9: Model comparison, 'faminc'

	Rsq	AdjRsq	AIC	BIC
he	0.1258	0.1237	10316.7	10328.8
he, we	0.1613	0.1574	10300.9	10317.1
he, we, kl6	0.1772	0.1714	10294.7	10315.0
he, we, kl6, xtra_x5, xtra_x6	0.1778	0.1681	10298.4	10326.8

As a side note, a quick way of extracting the information criteria from an `lm()` object is illustrated in the following code fragment.

```
library(stats)
smod1 <- summary(mod1)
Rsq <- smod1$r.squared
AdjRsq <- smod1$adj.r.squared
aic <- AIC(mod1)
bic <- BIC(mod1)
c(Rsq, AdjRsq, aic, bic)
```

```
## [1]     0.125801     0.123749 10316.651535 10328.828905
```

Another potentially useful tool for building an appropriate model is the Ramsey specification test, RESET. This method automatically adds higher-order polynomial terms to your model and tests the joint hypothesis that their coefficients are all zero. Thus, the null hypothesis of the test is H_0: "No higher-order polynomial terms are necessary"; if we reject the null hypothesis we need to consider including such terms.

The *R* function that performs a RESET test is `resettest`, which requires the following argumets: `formula`, the formula of the model to be tested or the name of an already calculated `lm` object; `power`, a set of integers indicating the powers of the polynomial terms to be included; `type`, which could be one of "fitted", "regressor", or "princomp", indicating whether the aditional terms should be powers of the regressors, fitted values, or the first principal component of the regressor matrix; and, finally, `data`, which specifies the dataset to be used if a formula has been provided and not a model object. The following code applies the test to the complete *faminc* model, first using only quadratic terms of the fitted values, then using both quadratic and cubic terms.

```
mod3 <- lm(faminc~he+we+kl6, data=edu_inc)
resettest(mod3, power=2, type="fitted")
```

```
##
##  RESET test
##
```

Table 6.10: A simple linear 'mpg' model

term	estimate	std.error	statistic	p.value
(Intercept)	42.91551	0.834867	51.4040	0
cyl	-3.55808	0.145676	-24.4247	0

```
## data:  mod3
## RESET = 5.984, df1 = 1, df2 = 423, p-value = 0.0148
```

```r
resettest(mod3, power=2:3, type="fitted")
```

```
##
##  RESET test
##
## data:  mod3
## RESET = 3.123, df1 = 2, df2 = 422, p-value = 0.0451
```

The number labeled as RESET in the output is the F-statistic of the test under the null hypothesis followed by the two types of degrees of freedom of the F distribution and the p-value. In our case both p-values are slightly lower than 0.05, indicating that the model marginally fails the specification test and some higher order terms may be necessary.

6.6 Collinearity

There is collinearity among regressors when two or more regressors move closely with each other or display little variability. A consequence of collinearity is large variance in the estimated parameters, which increases the chances of not finding them significantly different from zero. The estimates are, however, unbiased since (imperfect) collinearity does not technically violate the Gauss-Markov assumptions. Collinearity tends to show insignificant coefficients even when measures of goodness-of-fit such as R^2 or overall significance (the F-statistic) may be quite large.

Let us consider the example of the dataset *cars*, where *mpg* is miles per gallon, *cyl* is number of cylinders, *eng* is engine displacement in cubic inches, and *wgt* is the weight of the vehicle in pounds.

```r
data("cars", package="PoEdata")
mod1 <- lm(mpg~cyl, data=cars)
kable(tidy(mod1), caption="A simple linear 'mpg' model")
```

This naive model suggests a strong effect of the number of cylinders on fuel economy, but if we introduce more terms in the equation this result changes substantially.

Table 6.11: Multivariate 'mpg' model

term	estimate	std.error	statistic	p.value
(Intercept)	44.370962	1.480685	29.966509	0.000000
cyl	-0.267797	0.413067	-0.648313	0.517166
eng	-0.012674	0.008250	-1.536225	0.125298
wgt	-0.005708	0.000714	-7.995143	0.000000

Table 6.12: Variance inflation factors for the 'mpg' regression model

regressor	VIF
cyl	10.51551
eng	15.78646
wgt	7.78872

```r
mod2 <- lm(mpg~cyl+eng+wgt, data=cars)
kable(tidy(mod2), caption="Multivariate 'mpg' model")
```

In the model summarized in Table 6.11 the number of cylinders becomes insignificant alltogether, a sharp change with respect to the previous specification. This high sensitivity of the estimates when other variables are introduced is also a sign of collinearity. Indeed, it is reasonable to believe that the characteristics of the vehicles vary together: heavier vehicles have more cylinders and bigger engines.

A test that may be useful in detecting collinearity is to calculate the *variance inflation factor*, VIF, for each regressor. The rule of thumb is that a regressor produces collinearity if its VIF is greater than 10. Equation 6.13 shows the formula for the variance inflation factor, where R_k^2 is the R^2 from regressing the variable x_k on all the remaining regressors.

$$VIF_k = \frac{1}{1-R_k^2} \tag{6.13}$$

```r
mod2 <- lm(mpg~cyl+eng+wgt, data=cars)
tab <- tidy(vif(mod2))
kable(tab,
caption="Variance inflation factors for the 'mpg' regression model",
  col.names=c("regressor","VIF"))
```

The results in Table 6.12 show that the regressors `cyl` and `eng` fail the collinearity test, having VIFs greater than 10.

Table 6.13: Forecasting in the quadratic 'andy' model

fit	lwr	upr
76.974	67.5326	86.4155

6.7 Prediction and Forecasting

We have previously discussed the semantic difference between prediction and forecasting, with prediction meaning the estimation of an expected value of the response and forecasting meaning an estimate of a particular value of the response. We mentioned that, for the same vector of regressors, prediction has a narrower confidence interval than forecasting because forecasting includes, besides the uncertainty of the expected value of the response, the variablility of a particular observation about its mean. I essentially repeat the same procedure here for the quadratic *andy* model, which regresses *sales* on *price*, *advert*, and $advert^2$. The key R function to calculate both predictions and forecasts is the function `predict`, with the following arguments: `model`, which is the name of a model object; `newdata`, which contains the new data points where prediction is desired; if `newdata` is missing, predictions are calculated for all observations in the dataset; `interval`, which can be "none", "confidence", or "prediction", and tells R whether we want only a point estimate of the response, a prediction with its confidence interval, or a forecast with its confidence interval; `level`, which is the confidence level we want; if missing, `level` is 95%; other arguments (see `?predict()` for more details).

```
predpoint <- data.frame(price=6, advert=1.9)
mod3 <- lm(sales~price+advert+I(advert^2), data=andy)
kable(tidy(predict(mod3, newdata=predpoint,
    interval="prediction")),
    caption="Forecasting in the quadratic 'andy' model")
```

Table 6.13 displays the point estimate and forecast interval estimate for the data point $price = \$6$ and $advert = 1.9$, which, remember, stands for advertising expenditure of $1900.

Chapter 7

Using Indicator Variables

```
rm(list=ls())
library(bookdown)
library(PoEdata)#for PoE datasets
library(knitr)  #for referenced tables with kable()
library(xtable) #makes data frame for kable
library(printr) #automatically prints output nicely
library(effects)
library(car)
library(AER)
library(broom) #for tidy lm output and function glance()
library(stats)
library(lmtest)#for coeftest() and other test functions
library(stargazer) #nice and informative tables
```

This chapter introduces the package `lmtest` (Hothorn et al. 2015).

7.1 Factor Variables

Indicator variables show the presence of a certain non-quantifiable attribute, such as whether an individual is male or female, or whether a house has a swimming pool. In R, an indicator variable is called *factor*, *category*, or *ennumerated type* and there is no distinction between binary (such as yes-no, or male-female) or multiple-category factors.

Factors can be either numerical or character variables, ordered or not. R automatically creates dummy variables for each category within a factor variable and excludes a

Table 7.1: Summary for 'utown' dataset

	price	sqft	age	utown	pool	fplace
	Min. :134	Min. :20.0	Min. : 0.00	0:481	0:796	0:482
	1st Qu.:216	1st Qu.:22.8	1st Qu.: 3.00	1:519	1:204	1:518
	Median :246	Median :25.4	Median : 6.00	NA	NA	NA
	Mean :248	Mean :25.2	Mean : 9.39	NA	NA	NA
	3rd Qu.:278	3rd Qu.:27.8	3rd Qu.:13.00	NA	NA	NA
	Max. :345	Max. :30.0	Max. :60.00	NA	NA	NA

(baseline) category from a model. The choice of the baseline category, as well as which categories should be included can be changed using the function `contrasts()`.

The following code fragment loads the `utown` data, declares the indicator variables `utown`, `pool`, and `fplace` as factors, and displays a summary statistics table (Table 7.1). Please notice that the factor variables are represented in the summary table as count data, i.e., how many observations are in each category.

```
library(stargazer); library(ggplot2)
data("utown", package="PoEdata")
utown$utown <- as.factor(utown$utown)
utown$pool <- as.factor(utown$pool)
utown$fplace <- as.factor(utown$fplace)
kable(summary.data.frame(utown),
          caption="Summary for 'utown' dataset")
```

This example uses the `utown` dataset, where prices are in thousands, `sqft` is the living area is hundreds of square feet, and `utown` marks houses located near university.

7.2 Examples

```
mod4 <- lm(price~utown*sqft+age+pool+fplace, data=utown)
kable(tidy(mod4), caption="The 'house prices' model")
```

```
bsqft <- 1000*coef(mod4)[["sqft"]]
bsqft1 <- 1000*(coef(mod4)[["sqft"]]+coef(mod4)[["utown1:sqft"]])
```

Notice how the model in the above code sequence has been specified: the term `utown*sqft` created three terms: `utown`, `sqft`, and the interaction term `utown:sqft`. Table 7.2 shows the estimated coefficients and their statistics; please notice how the factor variables have been market at the end of their names with `1` to show which

Table 7.2: The 'house prices' model

term	estimate	std.error	statistic	p.value
(Intercept)	24.499985	6.191721	3.95689	0.000081
utown1	27.452952	8.422582	3.25945	0.001154
sqft	7.612177	0.245176	31.04775	0.000000
age	-0.190086	0.051205	-3.71229	0.000217
pool1	4.377163	1.196692	3.65772	0.000268
fplace1	1.649176	0.971957	1.69676	0.090056
utown1:sqft	1.299405	0.332048	3.91331	0.000097

category they represent. For instance, `utown1` is the equivalent of a dummy variable equal to 1 when `utown` is equal to 1. When a factor has n categories the regression output will show $n-1$ dummies marked to identify each category.

According to the results in Table 7.2, an extra hundred square feet increases the price by `bsqft`=\$7612.18 if the house is not near university and by `bsqft1` = \$8911.58 if the house is near university. This example shows how interaction terms allow distinguishing the marginal effect of a continuous variable (`sqft`) for one category (`utown=1`) from the marginal effect of the same continuous variable within another category (`utown=0`).

In general, the marginal effect of a regressor on the response is, as we have seen before, the partial derivative of the response with respect to that regressor. For example, the marginal effect of `sqft` in the *house prices* model is as shown in Equation 7.1.

$$\frac{\partial \widehat{price}}{\partial sqft} = b[sqft] + utown \times b[utown : sqft] \tag{7.1}$$

Let us look at another example, the *wage* equation using the dataset *cps4_small*.

```r
data("cps4_small", package="PoEdata")
names(cps4_small)
```

```
##  [1] "wage"    "educ"    "exper"   "hrswk"   "married" "female"  "metro"
##  [8] "midwest" "south"   "west"    "black"   "asian"
```

This dataset already includes dummy variables for gender, race, and region, so that we do not need *R*'s capability of building these dummies for us. Although usually it is useful to declare the categorical variables as factors, I will not do so this time. Let us consider the model in Equation 7.2.

$$wage = \beta_1 + \beta_2 educ + \delta_1 black + \delta_2 female + \gamma(black \times female) \tag{7.2}$$

Table 7.3: A wage-discrimination model

term	estimate	std.error	statistic	p.value
(Intercept)	-5.28116	1.900468	-2.77887	0.005557
educ	2.07039	0.134878	15.35009	0.000000
black	-4.16908	1.774714	-2.34916	0.019011
female	-4.78461	0.773414	-6.18635	0.000000
black:female	3.84429	2.327653	1.65158	0.098937

```
mod5 <- lm(wage~educ+black*female, data=cps4_small)
delta1 <- coef(mod5)[["black"]]
delta2 <- coef(mod5)[["female"]]
gamma <- coef(mod5)[["black:female"]]
blfm <- delta1+delta2+gamma
kable(tidy(mod5), caption="A wage-discrimination model")
```

What are the expected wages for different categories, according to Equation 7.2 and Table 7.3?

- white male: $\beta_1 + \beta_2 educ$ (the baseline category)
- black male: $\beta_1 + \beta_2 educ + \delta_1$
- white female: $\beta_1 + \beta_2 educ + \delta_2$
- black female: $\beta_1 + \beta_2 educ + \delta_1 + \delta_2 + \gamma$

These formulas help evaluating the differences in wage expectations between different categories. For instance, given the same education, the difference between black female and white male is $\delta_1 + \delta_2 + \gamma = -5.11$, which means that the average black female is paid less by \$5.11 than the average white man.

To test the hypothesis that neither race nor gender affects wage is to test the joint hypothesis $H_0 : \delta_1 = 0, \quad \delta_2 = 0, \quad \gamma = 0$ against the alterntive that at least one of these coefficients is different from zero. We have learned already how to perform an F-test the hard way and how to retrieve the quantities needed for the F-statistic (SSE_R, SSE_U, J, and $N - K$). Let us this time use R's `linearHypothesis` function.

```
hyp <- c("black=0", "female=0", "black:female=0")
tab <- tidy(linearHypothesis(mod5, hyp))
kable(tab,
caption="Testing a joint hypothesis for the 'wage' equation")
```

The results in Table 7.4 indicates that the null hypothesis of no discrimination can be rejected.

Table 7.4: Testing a joint hypothesis for the 'wage' equation

res.df	rss	df	sumsq	statistic	p.value
998	135771	NA	NA	NA	NA
995	130195	3	5576.47	14.2059	0

7.3 Comparing Two Regressions: the Chow Test

By interacting a binary indicator variable with all the terms in a regression and testing that the new terms are insignificant we can determine if a certain category is significantly different than the other categories. Starting with the *wage* regression in Equation 7.2, let us include the indicator variable "south".

```
dnosouth <- cps4_small[which(cps4_small$south==0),]#no south
dsouth <- cps4_small[which(cps4_small$south==1),] #south
mod5ns <- lm(wage~educ+black*female, data=dnosouth)
mod5s <- lm(wage~educ+black*female, data=dsouth)
mod6 <- lm(wage~educ+black*female+south/(educ+black*female),
                      data=cps4_small)
stargazer(mod6, mod5ns, mod5s, header=FALSE,
  type=.stargazertype,
  title="Model comparison, 'wage' equation",
  keep.stat="n",digits=2, single.row=TRUE,
  intercept.bottom=FALSE)
```

The table titled "Model comparison, 'wage' equation" presents the results of three equations: equation (1) is the full *wage* model with all terms interacted with variable *south*; equation (2) is the basic 'wage' model shown in Equation 7.2 with the sample restricted to non-south regions, and, finally, equation (3) is the same as (2) but with the sample restricted only to the south region. (This table is constructed with the function `stargazer` from the package by the same name (Hlavac 2015).)

However, in *R* it is not necessary to split the data manually as I did in the above code sequence; instead, we just write two equations like `mod5` and `mod6` and ask *R* to do a Chow test to see if the two equations are statistically identical. The Chow test is performed in *R* by the function `anova`, with the results presented in Table 7.6, where the F statistic is equal to 0.320278 and the corresponding p-value of 0.900945.

```
kable(anova(mod5, mod6),
      caption="Chow test for the 'wage' equation")
```

Table 7.6 indicates that the null hypothesis that the equations are equivalent cannot be rejected. In other words, our test provides no evidence that wages in the south

Table 7.5: Model comparison, 'wage' equation

	Dependent variable:		
	wage		
	(1)	(2)	(3)
Constant	-6.61^{***} (2.34)	-6.61^{***} (2.30)	-2.66 (3.42)
educ	2.17^{***} (0.17)	2.17^{***} (0.16)	1.86^{***} (0.24)
black	-5.09^{*} (2.64)	-5.09^{*} (2.60)	-3.38 (2.58)
female	-5.01^{***} (0.90)	-5.01^{***} (0.89)	-4.10^{***} (1.58)
south	3.94 (4.05)		
black:female	5.31 (3.50)	5.31 (3.45)	2.37 (3.38)
educ:south	-0.31 (0.29)		
black:south	1.70 (3.63)		
female:south	0.90 (1.77)		
black:female:south	-2.94 (4.79)		
Observations	1,000	704	296
Note:			*p<0.1; **p<0.05; ***p<0.01

Table 7.6: Chow test for the 'wage' equation

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
995	130195	NA	NA	NA	NA
990	129984	5	210.258	0.320278	0.900945

region are statistically different from the rest of the regions.

7.4 Indicator Variables in Log-Linear Models

An indicator regressor is essentially no different from a continuous one and its interpretation is very similar to the one we have studied in the context of log-linear models. In a model like $log(y) = \beta_1 + \beta_2 x + \delta D$, where D is an indicator variable, the percentage difference between the two categories represented by D can be calculated in two ways, both using the coefficient δ:

- approximate: $\%\Delta y \cong 100\delta$
- exact: $\%\Delta y = 100(e^{\delta} - 1)$

Let us calculate these two effets in a log-linear wage equation based on the dataset *cps4_small*.

```
data("cps4_small", package="PoEdata")
mod1 <- lm(log(wage)~educ+female, data=cps4_small)
approx <- 100*coef(mod1)[["female"]]
exact <- 100*(exp(coef(mod1)[["female"]])-1)
```

The results indicate a percentage difference in expected wage between females and males as follows: $\%\Delta wage = -24.32\%$ (approximate), or $\%\Delta wage = -21.59\%$ (exact).

7.5 The Linear Probability Model

Linear probability models are regression models in which the response, rather than a regressor is a binary indicator variable. However, since the regressors can be either continuous or factor variables, the fitted values will be continuous. Equation 7.3, where $y \in \{0, 1\}$ shows a general linear probability model.

$$y = \beta_1 + \beta_2 x_2 + ... + \beta_k x_k + e \tag{7.3}$$

How can a continous fitted variable be interpreted when the actual response is binary? As it turns out, the fitted value represents the *probability* that the response takes the value 1, $\hat{y} = Pr(y = 1)$. There are two major problems with this model. First, the model is heteroskedastic, with the variance being larger in the middle range of the fitted values; second, the fitted variable being continuous, it can take values, unlike a probability function, outside the interval $[0, 1]$.

Table 7.7: Linear probability model, the 'coke' example

term	estimate	std.error	statistic	p.value
(Intercept)	0.890215	0.065485	13.59421	0.000000
pratio	-0.400861	0.061349	-6.53407	0.000000
disp_coke	0.077174	0.034392	2.24397	0.025026
disp_pepsi	-0.165664	0.035600	-4.65352	0.000004

The next example, based on the dataset *coke* estimates the probability that a customer chooses *coke* when *coke* or *pepsi* are displayed. Besides the two indicator variables *disp_coke* and *disp_pepsi*, the model includes the price ratio between *coke* and *pepsi*.

```
# Linear probability example
data("coke", package="PoEdata")
mod2 <- lm(coke~pratio+disp_coke+disp_pepsi, data=coke)
kable(tidy(mod2),
 caption="Linear probability model, the 'coke' example")
```

```
# Graph for the linear probability model
b00 <- coef(mod2)[[1]]
b10 <- b00+coef(mod2)[["disp_coke"]]
b11 <- b10+coef(mod2)[["disp_pepsi"]]
b01 <-b11-coef(mod2)[["disp_coke"]]
b2 <- coef(mod2)[["pratio"]]
plot(coke$pratio, coke$coke,
     ylab="Pr[coke]", xlab="price ratio")
abline(b00, b2, lty=2, col=2)
abline(b10,b2, lty=3, col=3)
abline(b11,b2, lty=4, col=4)
abline(b01,b2, lty=5, col=5)
legend("topright", c("00","10","11","01"),
       lty=c(2,3,4,5), col=c(2,3,4,5))
```

Figure 7.1 plots the probalility of choosing *coke* with respect to the price ratio for the four possible combinations of the indicator variables *disp_coke* and *disp_pepsi*. In addition, the graph shows the observation points, all located at a height of either 0 or 1. In the legend, the first digit is 0 if *coke* is not displayed and 1 otherwise; the second digit does the same for *pepsi*. Thus, the highest probability of choosing *coke* for any given *pratio* happens when *coke* is displayed and *pepsi* is not (the line marked 10).

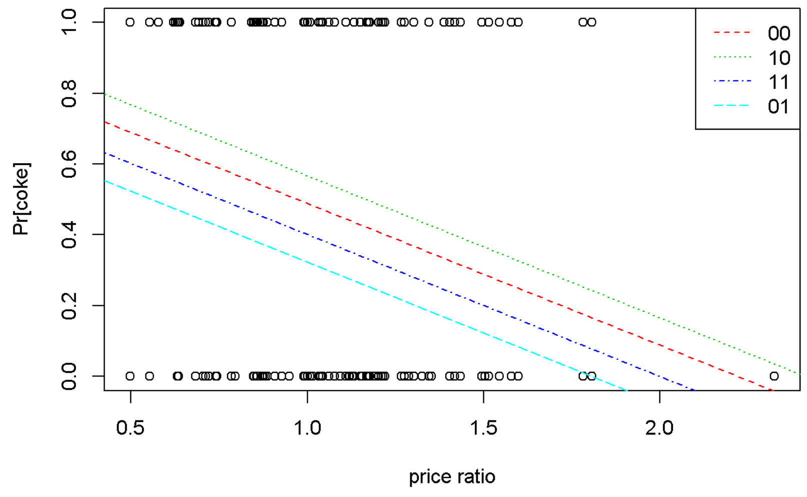


Figure 7.1: Linear probability: the 'coke' example

7.6 Treatment Effects

Treatment effects models aim at measuring the differences between a *treatment* group and a *control* group. The **difference estimator** is the simplest such a model and consists in constructing an indicator variable to distinguish between the two groups. Consider, for example, Equation 7.4, where $d_i = 1$ if observation i belongs to the treatment group and $d_i = 0$ otherwise.

$$y_i = \beta_1 + \beta_2 d_i + e_i \tag{7.4}$$

Thus, the expected value of the response is β_1 for the control group and $\beta_1 + \beta_2$ for the treatment group. Put otherwise, the coefficient on the dummy variable represents the difference between the treatment and the control group. Moreover, it turns out that b_2, the estimator of β_2 is equal to the difference between the sample averages of the two groups:

$$b_2 = \bar{y}_1 - \bar{y}_0 \tag{7.5}$$

The dataset *star* contains information on students from kindergarten to the third grade and was built to identify the determinants of student performance. Let us use this dataset for an example of a difference estimator, as well as an example of selecting just part of the variables and observations in a dataset.

The next piece of code presents a few new elements, that are used to select only a set of all the variables in the dataset, then to split the dataset in two parts: one for small classes, and one for regular classes. Lets look at these elements in the order in which they appear in the code. The function `attach(datasetname)` allows subsequent use of variable names without specifying from which database they come. While in general doing this is not advisable, I used it to simplify the next lines of code, which lists the variables I want to extract and assigns them to variable `vars`. As soon as I am done with splitting my dataset, I `detach()` the dataset to avoid subsequent confusion.

The line `starregular` is the one that actually retrieves the wanted variables and observations from the dataset. In essence, it picks from the dataset `star` the lines for `which small==0` (see, no need for `star$small` as long as `star` is still `attached`). The same about the variable `starsmall`. Another novelty in the following code fragment is the use of the *R* function `stargazer()` to vizualize a summary table instead of the usual function `summary`; `stargazer` shows a more familiar version of the summary statistics table.

```r
# Project STAR, an application of the simple difference estimator
data("star", package="PoEdata")
attach(star)
vars <- c("totalscore","small","tchexper","boy",
          "freelunch","white_asian","tchwhite","tchmasters",
            "schurban","schrural")
starregular <- star[which(small==0),vars]
starsmall <- star[which(small==1),vars]
detach(star)
stargazer(starregular, type=.stargazertype, header=FALSE,
title="Dataset 'star' for regular classes")
```

```r
stargazer(starsmall, type=.stargazertype, header=FALSE,
title="Dataset 'star' for small classes")
```

The two tables titled "Dataset'star'..." display summary statistics for a number of variables in dataset 'star'. The difference in the means of `totalscore` is equal to 13.740553, which should be equal to the coefficient of the dummy variable *small* in Equation 7.6. (Note: the results for this dataset seem to be slightly different from *PoE*4 because the datasets used are of different sizes.)

$$totalscore = \beta_1 + \beta_2 small + e \tag{7.6}$$

Table 7.8: Dataset 'star' for regular classes

Statistic	N	Mean	St. Dev.	Min	Max
totalscore	4,048	918.201	72.214	635	1,253
small	4,048	0.000	0.000	0	0
tchexper	4,028	9.441	5.779	0	27
boy	4,048	0.513	0.500	0	1
freelunch	4,048	0.486	0.500	0	1
white_asian	4,048	0.673	0.469	0	1
tchwhite	4,048	0.824	0.381	0	1
tchmasters	4,048	0.366	0.482	0	1
schurban	4,048	0.316	0.465	0	1
schrural	4,048	0.475	0.499	0	1

Table 7.9: Dataset 'star' for small classes

Statistic	N	Mean	St. Dev.	Min	Max
totalscore	1,738	931.942	76.359	747	1,253
small	1,738	1.000	0.000	1	1
tchexper	1,738	8.995	5.732	0	27
boy	1,738	0.515	0.500	0	1
freelunch	1,738	0.472	0.499	0	1
white_asian	1,738	0.685	0.465	0	1
tchwhite	1,738	0.862	0.344	0	1
tchmasters	1,738	0.318	0.466	0	1
schurban	1,738	0.306	0.461	0	1
schrural	1,738	0.463	0.499	0	1

```
mod3 <- lm(totalscore~small, data=star)
b2 <- coef(mod3)[["small"]]
```

The difference estimated based on Equation 7.6 is $b_2 = 13.740553$, which coincides with the difference in means we calculated above.

If the students were randomly assigned to small or regular classes and the number of observations is large there is no need for additional regressors (there is no omitted variable bias if other regressors are not correlated with the variable `small`). In some instances, however, including other regressors may improve the difference estimator. Here is a model that includes a 'teacher experience' variable and some school characteristics (the students were randomized within schools, but schools were not randomized).

```
school <- as.factor(star$schid)#creates dummies for schools
mod4 <- lm(totalscore~small+tchexper, data=star)
mod5 <- lm(totalscore~small+tchexper+school, data=star)
b2n <- coef(mod4)[["small"]]
b2s <- coef(mod5)[["small"]]
anova(mod4, mod5)
```

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
5763	30777461	NA	NA	NA	NA
5685	24072033	78	6705428	20.3025	0

By the introduction of school *fixed effects* the difference estimate have increased from 14.306725 to 15.320602. The `anova()` function, which tests the equivalence of the two regressions yields a very low p-value indicating that the difference between the two models is significant.

I have mentioned already, in the context of collinearity that a way to check if there is a relationship between an independent variable and the others is to regress that variable on the others and check the overall significance of the regression. The same method allows checking if the assignments to the treated and control groups are random. If we regress `small` on the other regressors and the assignment was random we should find no significant relationship.

```
mod6 <- lm(small~boy+white_asian+tchexper+freelunch, data=star)
kable(tidy(mod6),
  caption="Checking random assignment in the 'star' dataset")
```

```
fstat <- glance(mod6)$statistic
pf <- glance(mod6)$p.value
```

The F-statistic of the model in Table 7.10 is $F = 2.322664$ and its corresponding p-value 0.054362, which shows that the model is overall insignificant at the 5% level

Table 7.10: Checking random assignment in the 'star' dataset

term	estimate	std.error	statistic	p.value
(Intercept)	0.325167	0.018835	17.264354	0.000000
boy	-0.000254	0.012098	-0.021014	0.983235
white_asian	0.012360	0.014489	0.853110	0.393634
tchexper	-0.002979	0.001055	-2.825242	0.004741
freelunch	-0.008800	0.013526	-0.650570	0.515350

or lower. The coefficients of the independent variables are insignificant or extremely small (the coefficient on *tchexper* is statistically significant, but its marginal effect is a change in the probability that $small = 1$ of about 0.3 percent). Again, these results provide fair evidence that the students' assignment to small or regular classes was random.

7.7 The Difference-in-Differences Estimator

In many economic problems, which do not benefit from the luxury of random samples, the selection into one of the two groups is by choice, thus introducing a selection bias. The difference estimator is biased to the extent to which selection bias is present, therefore it is inadequate when selection bias may be present. The more complex **difference-in-differences** estimator is more appropriate in such cases. While the simple *difference* estimator assumes that before the treatment all units are identical or, at least, that the assignment to the treatment and control group is random, the *difference-in-differences* estimator takes into account any initial heterogeneity between the two groups.

The two 'differences' in the *diference-in-differences* estimator are: (i) the difference in the means of the *treatment* and *control* groups in the response variable **after** the treatment, and (ii) the difference in the means of the *treatment* and *control* groups in the response variable **before** the treatment. The 'difference' is between *after* and *before*.

Let us denote four averages of the response, as follows:

- $\bar{y}_{T,A}$ Treatment, After
- $\bar{y}_{C,A}$ Control, After
- $\bar{y}_{T,B}$ Treatment, Before
- $\bar{y}_{C,B}$ Control, Before

The *difference-in-differences* estimator $\hat{\delta}$ is defined as in Equation 7.7.

$$\hat{\delta} = (\bar{y}_{T,\,A} - \bar{y}_{C,\,A}) - (\bar{y}_{T,\,B} - \bar{y}_{C,\,B}) \tag{7.7}$$

Instead of manually calculating the four means and their difference-in-differences, it is possible to estimate the *difference-in-differences* estimator and its statistical properties by running a regression that includes indicator variables for *treatment* and *after* and their interaction term. The advantage of a regression over simply using Equation 7.7 is that the regression allows taking into account other factors that might influence the treatment effect. The simplest *difference-in-differences* regression model is presented in Equation 7.8, where y_{it} is the response for unit i in period t. In the typical *difference-in-differences* model there are only two periods, *before* and *after*.

$$y_{it} = \beta_1 + \beta_2 T + \beta_3 A + \delta T \times A + e_{it} \tag{7.8}$$

With a litle algebra it can be seen that the coefficinet δ on the interaction term in Equation 7.8 is exactly the *difference-in-differences* estimator defined in Equation 7.7. The following example calculates this estimator for the dataset $njmin3$, where the response is fte, the full-time equivalent employment, d is the *after* dummy, with $d = 1$ for the *after* period and $d = 0$ for the *before* period, and nj is the dummy that marks the treatment group ($nj_i = 1$ if unit i is in New Jersey where the minimum wage law has been changed, and $nj_i = 0$ if unit i in Pennsylvania, where the minimum wage law has not changed). In other words, units (fast-food restaurants) located in New Jersey form the treatment group, and units located in Pennsylvania form the control group.

```r
data("njmin3", package="PoEdata")
mod1 <- lm(fte~nj*d, data=njmin3)
mod2 <- lm(fte~nj*d+
             kfc+roys+wendys+co_owned, data=njmin3)
mod3 <- lm(fte~nj*d+
             kfc+roys+wendys+co_owned+
             southj+centralj+pa1, data=njmin3)
stargazer(mod1,mod2,mod3,
          type=.stargazertype,
          title="Difference in Differences example",
          header=FALSE, keep.stat="n",digits=2
#         single.row=TRUE, intercept.bottom=FALSE
          )
```

```r
# t-ratio for delta, the D-in-D estimator:
tdelta <- summary(mod1)$coefficients[4,3]
```

Table 7.11: Difference in Differences example

	Dependent variable:		
	fte		
	(1)	(2)	(3)
nj	-2.89^{**}	-2.38^{**}	-0.91
	(1.19)	(1.08)	(1.27)
d	-2.17	-2.22	-2.21
	(1.52)	(1.37)	(1.35)
kfc		-10.45^{***}	-10.06^{***}
		(0.85)	(0.84)
roys		-1.62^{*}	-1.69^{**}
		(0.86)	(0.86)
wendys		-1.06	-1.06
		(0.93)	(0.92)
co_owned		-1.17	-0.72
		(0.72)	(0.72)
southj			-3.70^{***}
			(0.78)
centralj			0.01
			(0.90)
pa1			0.92
			(1.38)
nj:d	2.75	2.85^{*}	2.81^{*}
	(1.69)	(1.52)	(1.50)
Constant	23.33^{***}	25.95^{***}	25.32^{***}
	(1.07)	(1.04)	(1.21)
Observations	794	794	794
Note:	*p<0.1; **p<0.05; ***p<0.01		

The coefficient on the term $nj:d$ in the Table titled "Difference-in-Differences example" is δ, our difference-in-differences estimator. If we want to test the null hypothesis $H_0 : \delta \geq 0$, the rejection region is at the left tail; since the calculated t, which is equal to 1.630888 is positive, we cannot reject the null hypothesis. In other words, there is no evidence that an increased minimum wage reduces employment at fast-food restaurants.

Figure 7.2 displays the change of fte from the period before ($d = 0$) to the period after the change in minimum wage ($d = 1$) for both the treatment and the control groups. The line labeled "counterfactual" shows how the treatment group would have changed in the absence of the treatment, assuming its change would mirror the change in the control group. The graph is plotted using Equation 7.8.

```r
b1 <- coef(mod1)[[1]]
b2 <- coef(mod1)[["nj"]]
b3 <- coef(mod1)[["d"]]
delta <- coef(mod1)[["nj:d"]]
C <- b1+b2+b3+delta
E <- b1+b3
B <- b1+b2
A <- b1
D <- E+(B-A)
# This creates an empty plot:
plot(1, type="n", xlab="period", ylab="fte", xaxt="n",
     xlim=c(-0.01, 1.01), ylim=c(18, 24))
segments(x0=0, y0=A, x1=1, y1=E, lty=1, col=2)#control
segments(x0=0, y0=B, x1=1, y1=C, lty=3, col=3)#treated
segments(x0=0, y0=B, x1=1, y1=D,      #counterfactual
         lty=4, col=4)
legend("topright", legend=c("control", "treated",
    "counterfactual"), lty=c(1,3,4), col=c(2,3,4))
axis(side=1, at=c(0,1), labels=NULL)
```

7.8 Using Panel Data

The *difference-in-differences* method does not require that the same units be observed both periods, since it works with averages before, and after. If we observe a number of units within the same time period, we construct a *cross-section*; if we construct different cross-sections in different periods we obtain a data structure named *repeated cross-sections*. Sometimes, however, we have information about *the same* units over several periods. A data structure in which the same (cross-sectional) units are observed over two or more periods is called a **panel data** and contains more

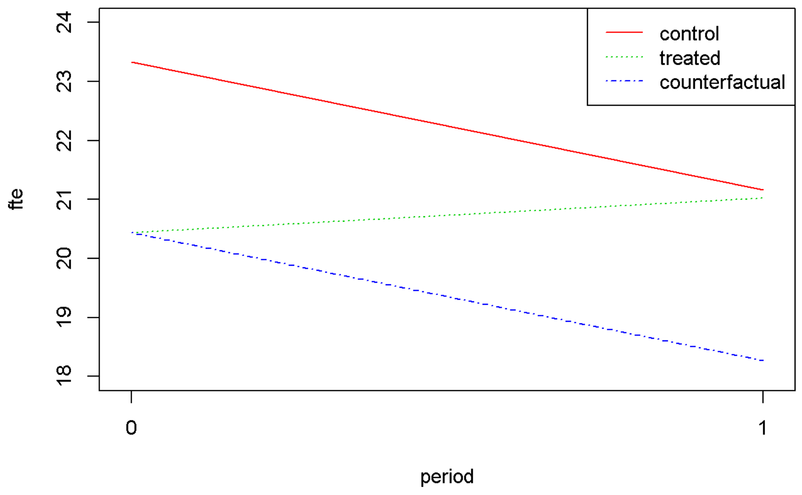


Figure 7.2: Difference-in-Differences for 'njmin3'

information than a repeated cross-section. Let us re-consider the simplest $njmin3$ equation (Equation 7.9), with the unit and time subscripts reflecting the panel data structure of the dataset. The time-invariant term c_i has been added to reflect unobserved, individual-specific attributes.

$$fte_{it} = \beta_1 + \beta_2 nj_i + \beta_3 d_t + \delta(nj_i \times d_t) + c_i + e_{it} \tag{7.9}$$

In the dataset $njmin3$, some restaurants, belonging to either group have been observed both periods. If we restrict the dataset to only those restaurants we obtain a short (two period) panel data. Let us re-calculate the difference-in-differences estimator using this panel data. To do so, we notice that, if we write Equation 7.9 twice, once for each period and subtract the *before* from the *after* equation we obtain Equation 7.10, where the response, $dfte$, is the after- minus- before difference in employment and the only regressor that remains is the *treatment* dummy, nj, whose coefficient is δ, the very *difference- in- differences* estimator we are trying to estimate.

$$dfte_i = \alpha + \delta nj_i + u_i \tag{7.10}$$

```r
mod3 <- lm(demp~nj, data=njmin3)
kable(tidy(summary(mod3)),
```

Table 7.12: Difference in differences with panel data

term	estimate	std.error	statistic	p.value
(Intercept)	-2.28333	0.731258	-3.12247	0.001861
nj	2.75000	0.815186	3.37346	0.000780

```
  caption="Difference in differences with panel data")
```

```
(smod3 <- summary(mod3))
```

```
##
## Call:
## lm(formula = demp ~ nj, data = njmin3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -39.22  -3.97   0.53   4.53  33.53
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.283      0.731   -3.12  0.00186 **
## nj             2.750      0.815    3.37  0.00078 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.96 on 766 degrees of freedom
##   (52 observations deleted due to missingness)
## Multiple R-squared:  0.0146, Adjusted R-squared:  0.0134
## F-statistic: 11.4 on 1 and 766 DF,  p-value: 0.00078
```

$R^2 = 0.014639, \quad F = 11.380248, \quad p = 0.00078$

Table 7.12 shows a value of the estimated *difference-in-differences* coefficient very close to the one we estimated before. Its t-statistic is still positive, indicating that the null hypothesis H_0: "an increse in minimum wage increases employment" cannot be rejected.

7.9 R Practicum

7.9.1 Extracting Various Information

Here is a reminder on how to extract various results after fitting a linear model. The function `names(lm.object)` returns a list of the names of different items contained in the object. Suppose we run an `lm()` model and name it `mod5`. Then, `mod5$name` returns the item identified by the name. Like about everything in *R*, there are many ways to extract values from an `lm()` object. I will present three objects that contain about everything we will need. These are the `lm()` object itself, `summary(lm())`, and the function `glance(lm())` in package `broom`. The next code shows the name lists for all three objects and a few examples of extracting various statistics.

```
library(broom)
mod5 <- mod5 <- lm(wage~educ+black*female, data=cps4_small)
smod5 <- summary(mod5)
gmod5 <- glance(mod5) #from package 'broom'
names(mod5)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
names(smod5)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
names(gmod5)
```

```
##  [1] "r.squared"     "adj.r.squared" "sigma"         "statistic"
##  [5] "p.value"       "df"            "logLik"        "AIC"
##  [9] "BIC"           "deviance"      "df.residual"
```

```
# Examples:
head(mod5$fitted.values)
```

```
##       1       2       3       4       5       6
## 23.0605 19.5635 23.6760 18.5949 19.5635 19.5635
```

```
head(mod5$residuals)#head of residual vector
```

```
##         1         2         3         4         5         6
## -4.360484 -8.063529 -8.636014  7.355080  4.466471  0.436471
```

```
smod5$r.squared
```

```
## [1] 0.208858
```

```
smod5$fstatistic # F-stat. and its degrees of freedom
```

```
##    value    numdf    dendf
##  65.6688   4.0000 995.0000
```

```
gmod5$statistic #the F-statistic of the model
```

```
## [1] 65.6688
```

```
mod5$df.residual
```

```
## [1] 995
```

For some often used statistics, such as coefficients and their statistics, fitted values, or residuals there are specialized functions to extract them from regression results.

```
N <- nobs(mod5)
yhat <- fitted(mod5) # fitted values
ehat <- resid(mod5)  # estimated residuals
allcoeffs <- coef(mod5) # only coefficients, no statistics
coef(mod5)[[2]] #or:
```

```
## [1] 2.07039
```

```
coef(mod5)[["educ"]]
```

```
## [1] 2.07039
```

```
coeftest(mod5)# all coefficients and their statistics
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.2812     1.9005  -2.779  0.00556 **
## educ           2.0704     0.1349  15.350  < 2e-16 ***
## black         -4.1691     1.7747  -2.349  0.01901 *
## female        -4.7846     0.7734  -6.186 8.98e-10 ***
## black:female   3.8443     2.3277   1.652  0.09894 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# The 'tidy()' function is from the package 'broom':
tabl <- tidy(mod5) #this gives the same result as coeftest()
```

Table 7.13: Example of using the function 'tidy'

term	estimate	std.error	statistic	p.value
(Intercept)	-5.28116	1.900468	-2.77887	0.005557
educ	2.07039	0.134878	15.35009	0.000000
black	-4.16908	1.774714	-2.34916	0.019011
female	-4.78461	0.773414	-6.18635	0.000000
black:female	3.84429	2.327653	1.65158	0.098937

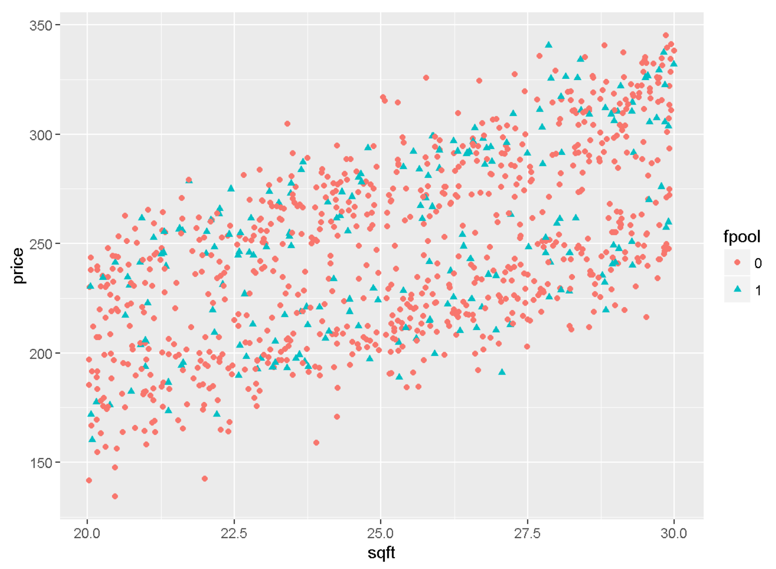
```
kable(tabl, caption=" Example of using the function 'tidy'")
```

7.9.2 `ggplot2`, An Excellent Data Visualising Tool

The function `ggplot` in package `ggplot2` is a very flexible plotting tool. For instance, it can assign different colours to different levels of an indicator variable. The following example uses the file *utown* to plot *price* against *sqft* using different colours for houses with swimming pool or for houses close to university.

```
library(ggplot2)
data("utown", package="PoEdata")
fpool <- as.factor(utown$pool)
futown <- as.factor(utown$utown)
ggplot(data = utown) +
  geom_point(mapping = aes(x = sqft, y = price,
  color = fpool, shape=fpool))
ggplot(data = utown) +
  geom_point(mapping = aes(x = sqft, y = price,
  color = futown, shape=futown))
```

A brilliant, yet concise presentation of the `ggplot()` system can be found in (Grolemund and Wickham 2016), which I strongly recommend.



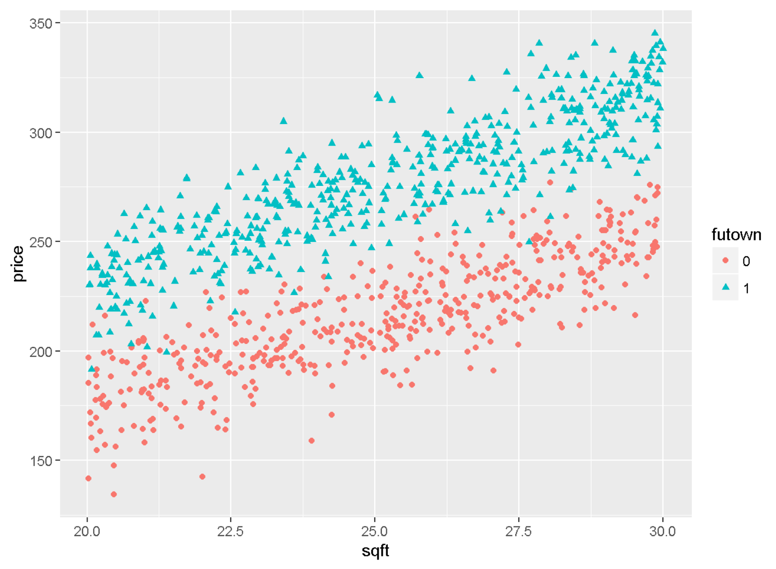


Figure 7.3: Graphs of dataset 'utown' using the 'ggplot' function

Chapter 8

Heteroskedasticity

```
rm(list=ls()) #Removes all items in Environment!
library(lmtest) #for coeftest() and bptest().
library(broom) #for glance() and tidy()
library(PoEdata) #for PoE4 datasets
library(car) #for hccm() robust standard errors
library(sandwich)
library(knitr)
library(stargazer)
```

Reference for the package `sandwich` (Lumley and Zeileis 2015).

One of the assumptions of the Gauss-Markov theorem is **homoskedasticity**, which requires that all observations of the response (dependent) variable come from distributions with the same variance σ^2. In many economic applications, however, the spread of y tends to depend on one or more of the regressors x. For example, in the *food* simple regression model (Equation 8.1) expenditure on food stays closer to its mean (regression line) at lower incomes and to be more spread about its mean at higher incomes. Think just that people have more choices at higher income whether to spend their extra income on food or something else.

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \tag{8.1}$$

In the presence of heteroskedasticity, the coefficient estimators are still unbiased, but their variance is incorrectly calculated by the usual OLS method, which makes confidence intervals and hypothesis testing incorrect as well. Thus, new methods need to be applied to correct the variances.

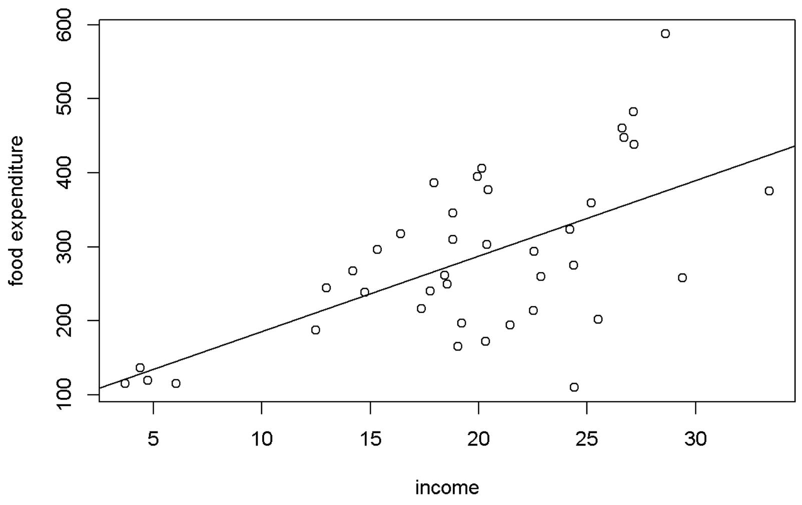


Figure 8.1: Heteroskedasticity in the 'food' data

8.1 Spotting Heteroskedasticity in Scatter Plots

When the variance of y, or of e, which is the same thing, is not constant, we say that the response or the residuals are **heteroskedastic**. Figure 8.1 shows, again, a scatter diagram of the *food* dataset with the regression line to show how the observations tend to be more spread at higher income.

```
data("food",package="PoEdata")
mod1 <- lm(food_exp~income, data=food)
plot(food$income,food$food_exp, type="p",
     xlab="income", ylab="food expenditure")
abline(mod1)
```

Another useful method to visualize possible heteroskedasticity is to plot the residuals against the regressors suspected of creating heteroskedasticity, or, more generally, against the fitted values of the regression. Figure 8.2 shows both these options for the simple *food_exp* model.

```
res <- residuals(mod1)
yhat <- fitted(mod1)
plot(food$income,res, xlab="income", ylab="residuals")
plot(yhat,res, xlab="fitted values", ylab="residuals")
```

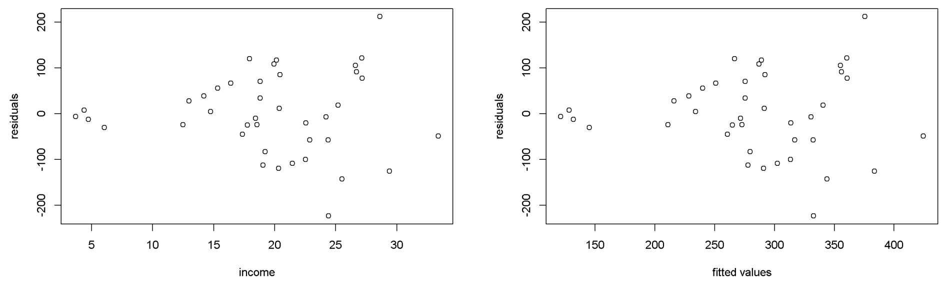


Figure 8.2: Residual plots in the 'food' model

8.2 Heteroskedasticity Tests

Suppose the regression model we want to test for heteroskedasticity is the one in Equation 8.2.

$$y_i = \beta_1 + \beta_2 x_{i2} + ... + \beta_K x_{iK} + e_i \tag{8.2}$$

The test we are construction assumes that the variance of the errors is a function h of a number of regressors z_s, which may or may not be present in the initial regression model that we want to test. Equation 8.3 shows the general form of the variance function.

$$var(y_i) = E(e_i^2) = h(\alpha_1 + \alpha_2 z_{i2} + ... + \alpha_S z_{iS}) \tag{8.3}$$

The variance $var(y_i)$ is constant only if all the coefficients of the regressors z in Equation 8.3 are zero, which provides the null hypothesis of our heteroskedasticity test shown in Equation 8.4.

$$H_0 : \alpha_2 = \alpha_3 = \; ... \alpha_S = 0 \tag{8.4}$$

Since we do not know the true variances of the response variable y_i, all we can do is to estimate the residuals from the initial model in Equation 8.2 and replace e_i^2 in Equation 8.3 with the estimated residuals. For a linear function $h()$, the test equation is, finally, Equation 8.5.

$$\hat{e}_i^2 = \alpha_1 + \alpha_2 z_{i2} + ... + \alpha_S z_{iS} + \nu_i \tag{8.5}$$

Te relevant test statistic is χ^2, given by Equation 8.6, where R^2 is the one resulted from Equation 8.5.

$$\chi^2 = N \times R^2 \sim \chi^2_{(S-1)} \tag{8.6}$$

The **Breusch-Pagan** heteroskedasticiy test uses the method we have just described, where the regressors z_s are the variables x_k in the initial model. Let us apply this test to the *food* model. The function to determine a critical value of the χ^2 distribution for a significance level α and $S - 1$ degrees of freedom is `qchisq(1-alpha, S-1)`.

```r
alpha <- 0.05
mod1 <- lm(food_exp~income, data=food)
ressq <- resid(mod1)^2
#The test equation:
modres <- lm(ressq~income, data=food)
N <- nobs(modres)
gmodres <- glance(modres)
S <- gmodres$df #Number of Betas in model
#Chi-square is always a right-tail test
chisqcr <- qchisq(1-alpha, S-1)
Rsqres <- gmodres$r.squared
chisq <- N*Rsqres
pval <- 1-pchisq(chisq,S-1)
```

Our test yields a value of the test statistic χ^2 of 7.38, which is to be compared to the critical χ^2_{cr} having $S - 1 = 1$ degrees of freedom and $\alpha = 0.05$. This critical value is $\chi^2_{cr} = 3.84$. Since the calculated χ^2 exceeds the critical value, we reject the null hypothesis of homoskedasticity, which means there is heteroskedasticity in our data and model. Alternatively, we can find the p-value corresponding to the calculated χ^2, $p = 0.007$.

Let us now do the same test, but using a **White** version of the residuals equation, in its quadratic form.

```r
modres <- lm(ressq~income+I(income^2), data=food)
gmodres <- glance(modres)
Rsq <- gmodres$r.squared
S <- gmodres$df #Number of Betas in model
chisq <- N*Rsq
pval <- 1-pchisq(chisq, S-1)
```

The calculated p-value in this version is $p = 0.023$, which also implies rejection of the null hypothesis of homoskedasticity.

Table 8.1: Breusch-Pagan heteroskedasticity test

statistic	p.value	parameter	method
7.38442	0.006579	1	studentized Breusch-Pagan test

The function `bptest()` in package `lmtest` does (the robust version of) the Breusch-Pagan test in *R*. The following code applies this function to the basic *food* equation, showing the results in Table 8.1, where 'statistic' is the calculated χ^2.

```r
mod1 <- lm(food_exp~income, data=food)
kable(tidy(bptest(mod1)),
caption="Breusch-Pagan heteroskedasticity test")
```

The **Goldfeld-Quandt** heteroskedasticity test is useful when the regression model to be tested includes an indicator variable among its regressors. The test compares the variance of one group of the indicator variable (say group 1) to the variance of the benchmark group (say group 0), as the null hypothesis in Equation8.7 shows.

$$H_0 : \hat{\sigma}_1^2 = \hat{\sigma}_0^2, \quad H_A : \hat{\sigma}_1^2 \neq \hat{\sigma}_0^2 \tag{8.7}$$

The test statistic when the null hyppthesis is true, given in Equation 8.8, has an F distribution with its two degrees of freedom equal to the degrees of freedom of the two subsamples, respectively $N_1 - K$ and $N_0 - K$.

$$F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_0^2} \tag{8.8}$$

Let us apply this test to a *wage* equation based on the dataset *cps*2, where *metro* is an indicator variable equal to 1 if the individual lives in a metropolitan area and 0 for rural area. I will split the dataset in two based on the indicator variable *metro* and apply the regression model (Equation 8.9) separately to each group.

$$wage = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 metro + e \tag{8.9}$$

```r
alpha <- 0.05 #two tail, will take alpha/2
data("cps2", package="PoEdata")
#Create the two groups, m (metro) and r (rural)
m <- cps2[which(cps2$metro==1),]
r <- cps2[which(cps2$metro==0),]
wg1 <- lm(wage~educ+exper, data=m)
wg0 <- lm(wage~educ+exper, data=r)
```

```
df1 <- wg1$df.residual #Numerator degrees of freedom
df0 <- wg0$df.residual #Denominatot df
sig1squared <- glance(wg1)$sigma^2
sig0squared <- glance(wg0)$sigma^2
fstat <- sig1squared/sig0squared
Flc <- qf(alpha/2, df1, df0)#Left (lower) critical F
Fuc <- qf(1-alpha/2, df1, df0) #Right (upper) critical F
```

The results of these calculations are as follows: calculated F statistic $F = 2.09$, the lower tail critical value $F_{lc} = 0.81$, and the upper tail critical value $F_{uc} = 1.26$. Since the calculated amount is greater than the upper critical value, we reject the hypothesis that the two variances are equal, facing, thus, a heteroskedasticity problem. If one expects the variance in the metropolitan area to be higher and wants to test the (alternative) hypothesis $H_0 : \sigma_1^2 \leq \sigma_0^2, \quad H_A : \sigma_1^2 > \sigma_0^2$, one needs to re-calcuate the critical value for $\alpha = 0.05$ as follows:

```
Fc <- qf(1-alpha, df1, df0) #Right-tail test
```

The critical value for the right tail test is $F_c = 1.22$, which still implies rejecting the null hypothesis.

The Goldfeld-Quant test can be used even when there is no indicator variable in the model or in the dataset. One can split the dataset in two using an arbitrary rule. Let us apply the method to the basic $food$ equation, with the data split in low-income (li) and high-income (hi) halves. The cutoff point is, in this case, the median income, and the hypothesis to be tested

$$H_0 : \sigma_{hi}^2 \leq \sigma_{li}^2, \quad H_A : \sigma_{hi}^2 > \sigma_{li}^2$$

```
alpha <- 0.05
data("food", package="PoEdata")
medianincome <- median(food$income)
li <- food[which(food$income<=medianincome),]
hi <- food[which(food$income>=medianincome),]
eqli <- lm(food_exp~income, data=li)
eqhi <- lm(food_exp~income, data=hi)
dfli <- eqli$df.residual
dfhi <- eqhi$df.residual
sigsqli <- glance(eqli)$sigma^2
sigsqhi <- glance(eqhi)$sigma^2
fstat <- sigsqhi/sigsqli #The larger var in numerator
Fc <- qf(1-alpha, dfhi, dfli)
pval <- 1-pf(fstat, dfhi, dfli)
```

Table 8.2: R function ‘gqtest()‘ with the ’food’ equation

df1	df2	statistic	p.value	method
18	18	3.61476	0.004596	Goldfeld-Quandt test

The resulting F statistic in the $food$ example is $F = 3.61$, which is greater than the critical value $F_{cr} = 2.22$, rejecting the null hypothesis in favour of the alternative hypothesis that variance is higher at higher incomes. The p-value of the test is $p = 0.0046$.

In the package `lmtest`, R has a specialized function to perform Goldfeld-Quandt tests, the function `gqtest` which takes, among other arguments, the `formula` describing the model to be tested, a break `point` specifying how the data should be split (percentage of the number of observations), what is the `alternative` hypothesis (“greater”, “two.sided”, or “less”), how the data should be ordered (`order.by=`), and `data=`. Let us apply `gqtest()` to the $food$ example with the same partition as we have just did before.

```r
foodeq <- lm(food_exp~income, data=food)
tst <- gqtest(foodeq, point=0.5, alternative="greater",
       order.by=food$income)
kable(tidy(tst),
  caption="R function `gqtest()` with the 'food' equation")
```

Please note that the results from applying `gqtest()` (Table 8.2 are the same as those we have already calculated.

8.3 Heteroskedasticity-Consistent Standard Errors

Since the presence of heteroskedasticity makes the lest-squares standard errors incorrect, there is a need for another method to calculate them. White robust standard errors is such a method.

The R function that does this job is `hccm()`, which is part of the `car` package and yields a heteroskedasticity-robust coefficient covariance matrix. This matrix can then be used with other functions, such as `coeftest()` (instead of `summary`), `waldtest()` (instead of `anova`), or `linearHypothesis()` to perform hypothesis testing. The function `hccm()` takes several arguments, among which is the `model` for which we want the robust standard errors and the `type` of standard errors we wish to calculate. `type` can be “constant” (the regular homoskedastic errors), “hc0”, “hc1”, “hc2”, “hc3”, or “hc4”; “hc1” is the default type in some statistical software packages. Let

Table 8.3: Regular standard errors in the 'food' equation

term	estimate	std.error	statistic	p.value
(Intercept)	83.4160	43.41016	1.92158	0.062182
income	10.2096	2.09326	4.87738	0.000019

Table 8.4: Robust (HC1) standard errors in the 'food' equation

term	estimate	std.error	statistic	p.value
(Intercept)	83.4160	27.46375	3.03731	0.004299
income	10.2096	1.80908	5.64356	0.000002

us compute robust standard errors for the basic $food$ equation and compare them with the regular (incorrect) ones.

```
foodeq <- lm(food_exp~income,data=food)
kable(tidy(foodeq),caption=
 "Regular standard errors in the 'food' equation")
```

```
cov1 <- hccm(foodeq, type="hc1") #needs package 'car'
food.HC1 <- coeftest(foodeq, vcov.=cov1)
kable(tidy(food.HC1),caption=
  "Robust (HC1) standard errors in the 'food' equation")
```

When comparing Tables 8.3 and 8.4, it can be observed that the robust standard errors are smaller and, since the coefficients are the same, the t-statistics are higher and the p-values are smaller. Lower p-values with robust standard errors is, however, the exception rather than the rule.

Next is an example of using robust standard errors when performing a fictitious linear hypothesis test on the basic 'andy' model, to test the hypothesis $H_0 : \beta_2 + \beta_3 = 0$

```
data("andy", package="PoEdata")
andy.eq <- lm(sales~price+advert, data=andy)
bp <- bptest(andy.eq) #Heteroskedsticity test
b2 <- coef(andy.eq)[["price"]]
b3 <- coef(andy.eq)[["advert"]]
H0 <- "price+advert=0"
kable(tidy(linearHypothesis(andy.eq, H0,
  vcov=hccm(andy.eq, type="hc1"))),
  caption="Linear hypothesis with robust standard errors")
```

Table 8.5: Linear hypothesis with robust standard errors

res.df	df	statistic	p.value
73	NA	NA	NA
72	1	23.387	7e-06

Table 8.6: Linear hypothesis with regular standard errors

res.df	rss	df	sumsq	statistic	p.value
73	2254.71	NA	NA	NA	NA
72	1718.94	1	535.772	22.4415	0.000011

```
kable(tidy(linearHypothesis(andy.eq, H0)),
  caption="Linear hypothesis with regular standard errors")
```

This example demonstrates how to introduce robust standards errors in a `linearHypothesis` function. It also shows that, when heteroskedasticity is not significant (`bptst` does not reject the homoskedasticity hypothesis) the robust and regular standard errors (and therefore the F statistics of the tests) are very similar.

Just for completeness, I should mention that a similar function, with similar uses is the function `vcov`, which can be found in the package `sandwich`.

8.4 GLS: Known Form of Variance

Let us consider the regression equation given in Equation 8.10), where the errors are assumed heteroskedastic.

$$y_i = \beta_1 + \beta_2 x_i + e_i, \quad var(e_i) = \sigma_i \tag{8.10}$$

Heteroskedasticity implies different variances of the error term for each observation. Ideally, one should be able to estimate the N variances in order to obtain reliable standard errors, but this is not possible. The second best in the absence of such estimates is an assumption of how variance depends on one or several of the regressors. The estimator obtained when using such an assumption is called a **generalized least squares** estimator, **gls**, which may involve a structure of the errors as proposed in Equation 8.11, which assumes a linear relationship between variance and the regressor x_i with the unknown parameter σ^2 as a proportionality factor.

Table 8.7: OLS estimates for the 'food' equation

term	estimate	std.error	statistic	p.value
(Intercept)	83.4160	43.41016	1.92158	0.062182
income	10.2096	2.09326	4.87738	0.000019

$$var(e_i) = \sigma_i^2 = \sigma^2 x_i \tag{8.11}$$

One way to circumvent guessing a proportionality factor in Equation 8.11 is to transform the initial model in Equation 8.10 such that the error variance in the new model has the structure proposed in Equation 8.11. This can be achieved if the initial model is divided through by $\sqrt{x_i}$ and estimate the new model shown in Equation 8.12. If Equation 8.12 is correct, then the resulting estimator is BLUE.

$$y_i^* = \beta_1 x_{i1}^* + \beta_2 x_{i2}^* + e_i^* \tag{8.12}$$

In general, if the initial variables are multiplied by quantities that are specific to each observation, the resulting estimator is called a **weighted least squares** estimator, **wls**. Unlike the robust standard errors method for heteroskedasticity correction, **gls** or **wls** methods change the estimates of regression coefficients.

The function `lm()` can do **wls** estimation if the argument `weights` is provided under the form of a vector of the same size as the other variables in the model. *R* takes the square roots of the weights provided to multiply the variables in the regression. Thus, if you wish to multiply the model by $\frac{1}{\sqrt{x_i}}$, the weights should be $w_i = \frac{1}{x_i}$.

Let us apply these ideas to re-estimate the *food* equation, which we have determined to be affected by heteroskedasticity.

```
w <- 1/food$income
food.wls <- lm(food_exp~income, weights=w, data=food)
vcvfoodeq <- coeftest(foodeq, vcov.=cov1)
kable(tidy(foodeq),
  caption="OLS estimates for the 'food' equation")
```

```
kable(tidy(food.wls),
  caption="WLS estimates for the 'food' equation" )
```

```
kable(tidy(vcvfoodeq),caption=
"OLS estimates for the 'food' equation with robust standard errors" )
```

Tables 8.7, 8.8, and 8.9 compare ordinary least square model to a weighted least squares model and to OLS with robust standard errors. The WLS model multiplies the

Table 8.8: WLS estimates for the 'food' equation

term	estimate	std.error	statistic	p.value
(Intercept)	78.6841	23.78872	3.30762	0.002064
income	10.4510	1.38589	7.54100	0.000000

Table 8.9: OLS estimates for the 'food' equation with robust standard errors

term	estimate	std.error	statistic	p.value
(Intercept)	83.4160	27.46375	3.03731	0.004299
income	10.2096	1.80908	5.64356	0.000002

variables by $1 \,/\, \sqrt{income}$, where the weights provided have to be $w = 1 \,/\, income$. The effect of introducing the weights is a slightly lower intercept and, more importantly, different standard errors. Please note that the WLS standard errors are closer to the robust (HC1) standard errors than to the OLS ones.

8.5 Grouped Data

We have seen already (Equation 8.7) how a dichotomous indicator variable splits the data in two groups that may have different variances. The generalized least squares method can account for group heteroskedasticity, by choosing appropriate weights for each group; if the variables are transformed by multiplying them by $1/\sigma_j$, for group j, the resulting model is homoskedastic. Since σ_j is unknown, we replace it with its estimate $\hat{\sigma}_j$. This method is named **feasible generalized least squares**.

```
data("cps2", package="PoEdata")
rural.lm <- lm(wage~educ+exper, data=cps2, subset=(metro==0))
sigR <- summary(rural.lm)$sigma
metro.lm <- lm(wage~educ+exper, data=cps2, subset=(metro==1))
sigM <- summary(metro.lm)$sigma
cps2$wght <- rep(0, nrow(cps2))
# Create a vector of weights
for (i in 1:1000)
{
  if (cps2$metro[i]==0){cps2$wght[i] <- 1/sigR^2}
  else{cps2$wght[i] <- 1/sigM^2}
}
wge.fgls <- lm(wage~educ+exper+metro, weights=wght, data=cps2)
wge.lm <- lm(wage~educ+exper+metro, data=cps2)
wge.hce <- coeftest(wge.lm, vcov.=hccm(wge.lm, data=cps2))
stargazer(rural.lm, metro.lm, wge.fgls,wge.hce,
  header=FALSE,
  title="OLS vs. FGLS estimates for the 'cps2' data",
  type=.stargazertype, # "html" or "latex" (in index.Rmd)
  keep.stat="n",  # what statistics to print
  omit.table.layout="n",
  star.cutoffs=NA,
  digits=3,
#  single.row=TRUE,
  intercept.bottom=FALSE, #moves the intercept coef to top
  column.labels=c("Rural","Metro","FGLS", "HC1"),
  dep.var.labels.include = FALSE,
  model.numbers = FALSE,
  dep.var.caption="Dependent variable: wage",
  model.names=FALSE,
  star.char=NULL) #supresses the stars
```

Table 8.10: OLS vs. FGLS estimates for the 'cps2' data

	Dependent variable: wage			
	Rural	Metro	FGLS	HC1
Constant	−6.166	−9.052	−9.398	−9.914
	(1.899)	(1.189)	(1.020)	(1.218)
educ	0.956	1.282	1.196	1.234
	(0.133)	(0.080)	(0.069)	(0.084)
exper	0.126	0.135	0.132	0.133
	(0.025)	(0.018)	(0.015)	(0.016)
metro			1.539	1.524
			(0.346)	(0.346)
Observations	192	808	1,000	

The table titled "OLS, vs. FGLS estimates for the 'cps2' data" helps comparing the coefficients and standard errors of four models: OLS for rural area, OLS for metro area, feasible GLS with the whole dataset but with two types of weights, one for each area, and, finally, OLS with heteroskedasticity-consistent (HC1) standard errors.

Please be reminded that the regular OLS standard errors are not to be trusted in the presence of heteroskedasticity.

The previous code sequence needs some explanation. It runs two regression models, `rural.lm` and `metro.lm` just to estimate $\hat{\sigma}_R$ and $\hat{\sigma}_M$ needed to calculate the weights for each group. The subsets, this time, were selected directly in the `lm()` function through the argument `subset=`, which takes as argument some logical expression that may involve one or more variables in the dataset. Then, I create a new vector of a size equal to the number of observations in the dataset, a vector that will be populated over the next few code lines with weights. I choose to create this vector as a new column of the dataset `cps2`, a column named `wght`. With this the hard part is done; I just need to run an `lm()` model with the option `weights=wght` and that gives my FGLS coefficients and standard errors.

The next lines make a `for` loop runing through each observation. If observation i is a rural area observation, it receives a weight equal to $1/\sigma_R^2$; otherwise, it receives the weight $1/\sigma_M^2$. Why did I square those *sigmas*? Because, remember, the argument `weights` in the `lm()` function requires the square of the factor multiplying the regression model in the WLS method.

The remaining part of the code repeats models we ran before and places them in one table for making comparison easier.

8.6 GLS: Unknown Form of Variance

Suppose we wish to estimate the model in Equation 8.13, where the errors are known to be heteroskedastic but their variance is an unknown function of S some variables z_s that could be among the regressors in our model or other variables.

$$y_i = \beta_1 + \beta_2 x_{i2} + ...\beta_k x_{iK} + e_i \tag{8.13}$$

Equation 8.14 uses the residuals from Equation 8.13 as estimates of the variances of the error terms and serves at estimating the functional form of the variance. If the assumed functional form of the variance is the exponential function $var(e_i) = \sigma_i^2 = \sigma^2 x_i^\gamma$, then the regressors z_{is} in Equation 8.14 are the logs of the initial regressors x_{is}, $z_{is} = log(x_{is})$.

$$ln(\hat{e}_i^2) = \alpha_1 + \alpha_2 z_{i2} + ... + \alpha_S z_{iS} + \nu_i \tag{8.14}$$

The variance estimates for each error term in Equation 8.13 are the fitted values, $\hat{\sigma}_i^2$ of Equation 8.14, which can then be used to construct a vector of weights for the regression model in Equation 8.13. Let us follow these steps on the $food$ basic equation where we assume that the variance of error term i is an unknown exponential function of income. So, the purpose of the following code fragment is to determine the weights and to supply them to the `lm()` function. Remember, `lm()` **multiplies** each observation by the **square root** of the weight you supply. For instance, if you want to multiply the observations by $1/\sigma_i$, you should supply the weight $w_i = 1/\sigma_i^2$.

```
data("food", package="PoEdata")
food.ols <- lm(food_exp~income, data=food)
ehatsq <- resid(food.ols)^2
sighatsq.ols  <- lm(log(ehatsq)~log(income), data=food)
vari <- exp(fitted(sighatsq.ols))
food.fgls <- lm(food_exp~income, weights=1/vari, data=food)
```

```
stargazer(food.ols, food.HC1, food.wls, food.fgls,
  header=FALSE,
  title="Comparing various 'food' models",
  type=.stargazertype, # "html" or "latex" (in index.Rmd)
  keep.stat="n",  # what statistics to print
  omit.table.layout="n",
  star.cutoffs=NA,
  digits=3,
#  single.row=TRUE,
  intercept.bottom=FALSE, #moves the intercept coef to top
  column.labels=c("OLS","HC1","WLS","FGLS"),
  dep.var.labels.include = FALSE,
  model.numbers = FALSE,
  dep.var.caption="Dependent variable: 'food expenditure'",
  model.names=FALSE,
  star.char=NULL) #supresses the stars
```

Table 8.11: Comparing various 'food' models

	Dependent variable: 'food expenditure'			
	OLS	HC1	WLS	FGLS
Constant	83.416	83.416	78.684	76.054
	(43.410)	(27.464)	(23.789)	(9.713)
income	10.210	10.210	10.451	10.633
	(2.093)	(1.809)	(1.386)	(0.972)
Observations	40		40	40

The table titled "Comparing various 'food' models" shows that the FGLS with unknown variances model substantially lowers the standard errors of the coefficients, which in turn increases the t-ratios (since the point estimates of the coefficients remain about the same), making an important difference for hypothesis testing.

For a few classes of variance functions, the weights in a GLS model can be calculated in R using the `varFunc()` and `varWeights()` functions in the package `nlme`.

8.7 Heteroskedasticity in the Linear Probability Model

As we have already seen, the linear probability model is, by definition, heteroskedastic, with the variance of the error term given by its binomial distribution parameter p, the probability that y is equal to 1, $var(y) = p(1-p)$, where p is defined in Equation 8.15.

$$p = \beta_1 + \beta_2 x_2 + ... + \beta_K x_K + e \tag{8.15}$$

Thus, the linear probability model provides a known variance to be used with GLS, taking care that none of the estimated variances is negative. One way to avoid negative or greater than one probabilities is to artificially limit them to the interval $(0, 1)$.

Let us revise the *coke* model in dataset `coke` using this structure of the variance.

```r
data("coke", package="PoEdata")
coke.ols <- lm(coke~pratio+disp_coke+disp_pepsi, data=coke)
coke.hc1 <- coeftest(coke.ols, vcov.=hccm(coke.ols, type="hc1"))
p <- fitted(coke.ols)
# Truncate negative or >1 values of p
pt<-p
pt[pt<0.01] <- 0.01
pt[pt>0.99] <- 0.99
sigsq <- pt*(1-pt)
wght <- 1/sigsq
coke.gls.trunc <- lm(coke~pratio+disp_coke+disp_pepsi,
                  data=coke, weights=wght)
# Eliminate negative or >1 values of p
p1 <- p
p1[p1<0.01 | p1>0.99] <- NA
sigsq <- p1*(1-p1)
wght <- 1/sigsq
```

```
coke.gls.omit <- lm(coke~pratio+disp_coke+disp_pepsi,
                  data=coke, weights=wght)
```

```
stargazer(coke.ols, coke.hc1, coke.gls.trunc, coke.gls.omit,
  header=FALSE,
  title="Comparing various 'coke' models",
  type=.stargazertype, # "html" or "latex" (in index.Rmd)
  keep.stat="n",  # what statistics to print
  omit.table.layout="n",
  star.cutoffs=NA,
  digits=4,
#  single.row=TRUE,
  intercept.bottom=FALSE, #moves the intercept coef to top
  column.labels=c("OLS","HC1","GLS-trunc","GLS-omit"),
  dep.var.labels.include = FALSE,
  model.numbers = FALSE,
  dep.var.caption="Dependent variable: 'choice of coke'",
  model.names=FALSE,
  star.char=NULL) #supresses the stars
```

Table 8.12: Comparing various 'coke' models

	Dependent variable: 'choice of coke'			
	OLS	HC1	GLS-trunc	GLS-omit
Constant	0.8902	0.8902	0.6505	0.8795
	(0.0655)	(0.0653)	(0.0568)	(0.0594)
pratio	−0.4009	−0.4009	−0.1652	−0.3859
	(0.0613)	(0.0604)	(0.0444)	(0.0527)
disp_coke	0.0772	0.0772	0.0940	0.0760
	(0.0344)	(0.0339)	(0.0399)	(0.0353)
disp_pepsi	−0.1657	−0.1657	−0.1314	−0.1587
	(0.0356)	(0.0344)	(0.0354)	(0.0360)
Observations	1,140		1,140	1,124

Chapter 9

Time-Series: Stationary Variables

```r
rm(list=ls()) #Removes all items in Environment!
library(dynlm) #for the `dynlm()` function
library(orcutt) # for the `cochrane.orcutt()` function
library(nlWaldTest) # for the `nlWaldtest()` function
library(zoo) # for time series functions (not much used here)
library(pdfetch) # for retrieving data (just mentioned here)
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for kable()
library(forecast)
```

New packages: `dynlm` (Zeileis 2016); `orcutt` (Spada, Quartagno, and Tamburini 2012); `nlWaldTest` (Komashko 2016); `zoo` [R-zoo]; `pdfetch` (Reinhart 2015); and `forecast` (Hyndman 2016).

Time series are data on several variables on an observational unit (such as an individual, country, or firm) when observations span several periods. Correlation among subsequent observations, the importance of the natural order in the data and dynamics (past values of data influence present and future values) are features of time series that do not occur in cross-sectional data.

Time series models assume, in addition to the usual linear regression assumptions, that the esries is **stationary**, that is, the distribution of the error term, as well as the correlation between error terms a few periods apart are constant over time. Constant distribution requires, in particular, that the variable does not display a trend in its mean or variance; constant correlation implies no clustering of observations in certain periods.

9.1 An Overview of Time Series Tools in *R*

R creates a time series variable or dataset using the function `ts()`, with the following main arguments: your `data` file in matrix or data frame form, the `start` period, the `end` period, the `frequency` of the data (1 is annual, 4 is quarterly, and 12 is monthly), and the `names` of your column variables. Another class of time series objects is created by the function `zoo()` in the package `zoo`, which, unlike `ts()`, can handle irregular or high-frequency time series. Both `ts` and `zoo` classes of objects can be used by the function `dynlm()` in the package with the same name to solve models that include lags and other time series specific operators.

In standard *R*, two functions are very useful when working with time series: the *difference* function, $diff(y_t) = y_t - y_{t-1}$, and the *lag* function, $lag(y_t) = y_{t-1}$.

The package `pdfetch` is a very useful tool for getting *R*-compatible time series data from different online sources such as the World Bank, Eurostat, European Central Bank, and Yahoo Finance. The package `WDI` retrieves data from the very rich World Development Indicators database, maintained by the World Bank.

9.2 Finite Distributed Lags

A *finite distributed lag* model (FDL) assumes a linear relationship between a dependent variable y and several lags of an independent variable x. Equation 9.1 shows a finite distributed lag model **of order *q*.**

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + ... + \beta_q x_{t-q} + e_t \tag{9.1}$$

The coefficient β_s is an **s-period delay multiplier**, and the coefficient β_0, the immediate (contemporaneous) impact of a change in x on y, is an **impact multiplier**. If x increases by one unit today, the change in y will be $\beta_0+\beta_1+...+\beta_s$ after s periods; this quantity is called the s-period **interim multiplier**. The **total multiplier** is equal to the sum of all βs in the model.

Let us look at Okun's law as an example of an FDL model. Okun's law relates contemporaneous (time t) change in unemployment rate, DU_t, to present and past levels of economic growth rate, G_{t-s}.

Table 9.1: The 'okun' dataset with differences and lags

g	u	uL1	du	gL1	gL2	gL3
1.4	7.3	NA	NA	NA	NA	NA
2.0	7.2	7.3	-0.1	1.4	NA	NA
1.4	7.0	7.2	-0.2	2.0	1.4	NA
1.5	7.0	7.0	0.0	1.4	2.0	1.4
0.9	7.2	7.0	0.2	1.5	1.4	2.0
1.5	7.0	7.2	-0.2	0.9	1.5	1.4

Table 9.2: The 'okun' distributed lag model with three lags

term	estimate	std.error	statistic	p.value
(Intercept)	0.5810	0.0539	10.7809	0.0000
L(g, 0:3)0	-0.2021	0.0330	-6.1204	0.0000
L(g, 0:3)1	-0.1645	0.0358	-4.5937	0.0000
L(g, 0:3)2	-0.0716	0.0353	-2.0268	0.0456
L(g, 0:3)3	0.0033	0.0363	0.0911	0.9276

```r
data("okun", package="PoEdata")
library(dynlm)
check.ts <- is.ts(okun) # "is structured as time series?"
okun.ts <- ts(okun, start=c(1985,2), end=c(2009,3),frequency=4)
okun.ts.tab <- cbind(okun.ts,
                 lag(okun.ts[,2], -1),
                 diff(okun.ts[,2], lag=1),
                 lag(okun.ts[,1], -1),
                 lag(okun.ts[,1], -2),
                 lag(okun.ts[,1], -3))
kable(head(okun.ts.tab),
      caption="The `okun` dataset with differences and lags",
      col.names=c("g","u","uL1","du","gL1","gL2","gL3"))
```

Table 9.1 shows how lags and differences work. Please note how each lag uses up an observation period.

```r
okunL3.dyn <- dynlm(d(u)~L(g, 0:3), data=okun.ts)
kable(tidy(summary(okunL3.dyn)), digits=4,
  caption="The `okun` distributed lag model with three lags")
```

Table 9.3: The ‘okun‘ distributed lag model with two lags

term	estimate	std.error	statistic	p.value
(Intercept)	0.5836	0.0472	12.3604	0.0000
L(g, 0:2)0	-0.2020	0.0324	-6.2385	0.0000
L(g, 0:2)1	-0.1653	0.0335	-4.9297	0.0000
L(g, 0:2)2	-0.0700	0.0331	-2.1152	0.0371

Table 9.4: Goodness-of-fit statistics for ‘okun‘ models

r.squared	statistic	AIC	BIC
0.652406	42.2306	-55.4318	-40.1085
0.653946	57.9515	-58.9511	-46.1293

```r
okunL2.dyn <- dynlm(d(u)~L(g, 0:2), data=okun.ts)
kable(tidy(summary(okunL2.dyn)), digits=4,
  caption="The `okun` distributed lag model with two lags")
```

Tables 9.2 and 9.3 summarize the results of linear models with 3 and 2 lags respectively. Many of the output analysis functions that we have used with the `lm()` function, such as `summary()` and `coeftest()` are also applicable to `dynlm()`.

```r
glL3 <- glance(okunL3.dyn)[c("r.squared","statistic","AIC","BIC")]
glL2 <- glance(okunL2.dyn)[c("r.squared","statistic","AIC","BIC")]
tabl <- rbind(glL3, as.numeric(glL2))
kable(tabl, caption="Goodness-of-fit statistics for `okun` models")
```

Table 9.4 compares the two FDL models of the *okun* example. The first row is the model with three lags, the second is the model with two lags. All the measures in this table points to the second model (two lags) as a better specification.

A note on how these tables were created in *R* is of interest. Table 9.1 was created using the function `cbind`, which puts together several columns (vectors); table 9.4 used two functions: `rbind()`, which puts together two rows, and `as.numeric`, which extracts only the numbers from the `glance` object, without the names of the columns.

9.3 Serial Correlation

Serial correlation, or **autocorrelation** in a time series describes the correlation between two observations separated by one or several periods. Time series tend to display autocorrelation more than cross sections because of their ordered nature.

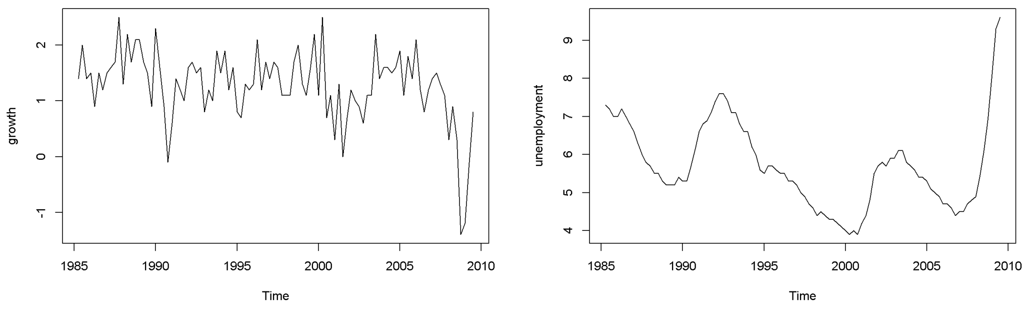


Figure 9.1: Growth and unemployment rates in the 'okun' dataset

Autocorrelation could be an attribute of one series, independent of the model in which this series appears. If this series is, however, an error term, its properties do depend on the model, since error series can only exist in relation to a model.

```
plot(okun.ts[,"g"], ylab="growth")
plot(okun.ts[,"u"], ylab="unemployment")
```

The "growth" graph in Figure 9.1 display clusters of values: positive for several periods followed by a few of negative values, which is an indication of autocorrelation; the same is true for unemployment, which does not change as dramatically as growth but still shows persistence.

```
ggL1 <- data.frame(cbind(okun.ts[,"g"], lag(okun.ts[,"g"],-1)))
names(ggL1) <- c("g","gL1")
plot(ggL1)
meang <- mean(ggL1$g, na.rm=TRUE)
abline(v=meang, lty=2)
abline(h=mean(ggL1$gL1, na.rm=TRUE), lty=2)

ggL2 <- data.frame(cbind(okun.ts[,"g"], lag(okun.ts[,"g"],-2)))
names(ggL2) <- c("g","gL2")
plot(ggL2)
meang <- mean(ggL2$g, na.rm=TRUE)
abline(v=meang, lty=2)
abline(h=mean(ggL2$gL2, na.rm=TRUE), lty=2)
```

Figures 9.2 illustrate the correlation between the growth rate and its first two lags, which is, indeed autocorrelation. But is there a more precise test to detect autocorrelation?

Suppose we wish to test the hypothesis formulated in Equation 9.2, where ρ_k is the

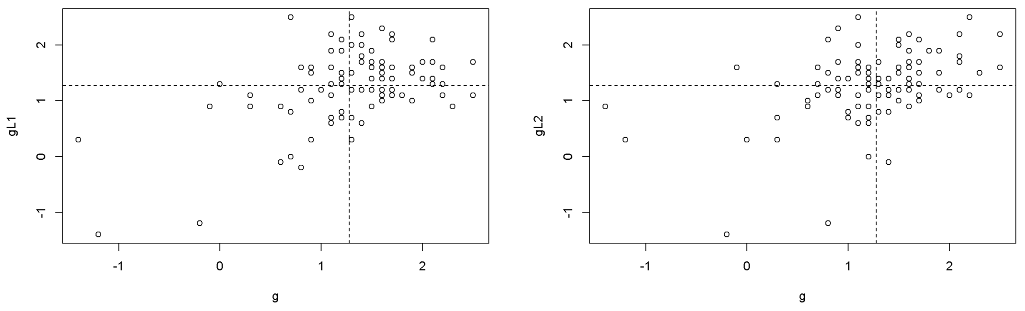


Figure 9.2: Scatter plots between 'g' and its lags

population k-th order autocorrelation coefficient.

$$H_0 : \rho_k = 0, \quad H_A : \rho_k \neq 0 \tag{9.2}$$

A test can be constructed based on the sample correlation coefficient, r_k, which measures the correlation between a variable and its k-th lag; the test statistic is given in Equation 9.3, where T is the number of periods.

$$Z = \frac{r_k - 0}{\sqrt{\frac{1}{T}}} = \sqrt{T} r_k \sim N(0,1) \tag{9.3}$$

For a 5% significance level, Z must be outside the interval $[-1.96, 1, 96]$, that is, in the rejection region. Rejecting the null hypothesis is, in this case, bad news, since rejection constitutes evidence of autocorrelation. So, for a way to remember the meaning of the test, one may think of it as a **test of non-autocorrelation**.

The results of the (non-) autocorrelation test are usually summarized in a **correlogram**, a bar diagram that visualizes the values of the test statistic $\sqrt{T}r_k$ for several lags as well as the 95% confidence interval. A bar ($\sqrt{T}r_k$) that exceedes (upward or downward) the limits of the confidence interval indicates autocorrelation for the corresponding lag.

```
growth_rate <- okun.ts[,"g"]
acf(growth_rate)
```

Figure 9.3 is a correlogram, where each bar corresponding to one lag, starting with lag 0. The correlogram shows little or no evidence of autocorrelation, except for the first and second lag (second and third bar in the figure).

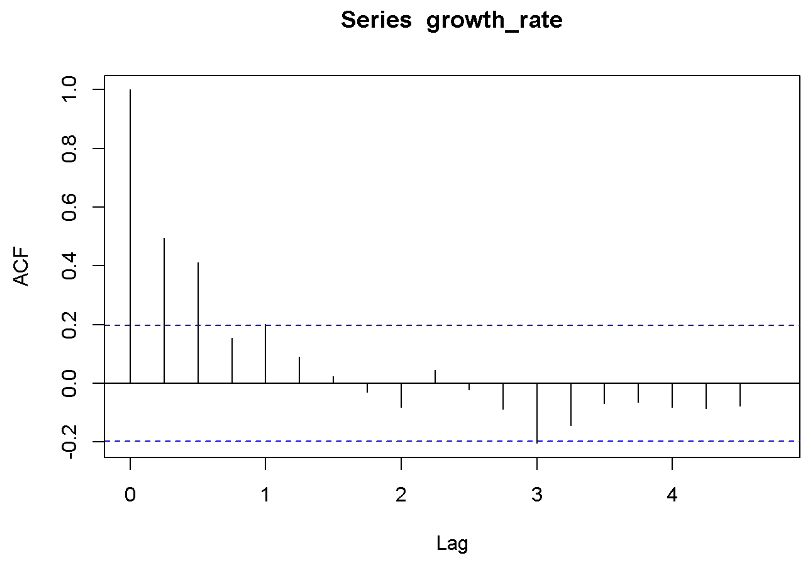


Figure 9.3: Correlogram for the growth rate, dataset 'okun'

Let us consider another example, the dataset *phillips_aus*, which containes quarterly data on unemploymnt and inflation over the period 1987Q1 to 2009Q3. We wish to apply the autocorrelation test to the error term in a time series regression to see if the non-autocorrelation in the errors is violated. Let us consider the FDL model in Equation 9.4.

$$inf_t = \beta_1 + \beta_2 Du_t + e_t \tag{9.4}$$

Let's first take a look at plots of the data (visualising the data is said to be a good practice rule in data analysis.)

```r
data("phillips_aus", package="PoEdata")
phill.ts <- ts(phillips_aus,
               start=c(1987,1),
               end=c(2009,3),
               frequency=4)
inflation <- phill.ts[,"inf"]
Du <- diff(phill.ts[,"u"])
plot(inflation)
plot(Du)
```

The plots in Figure 9.4 definitely show patterns in the data for both inflation and unemployment rates. But we are interested to determine if the error term in Equation

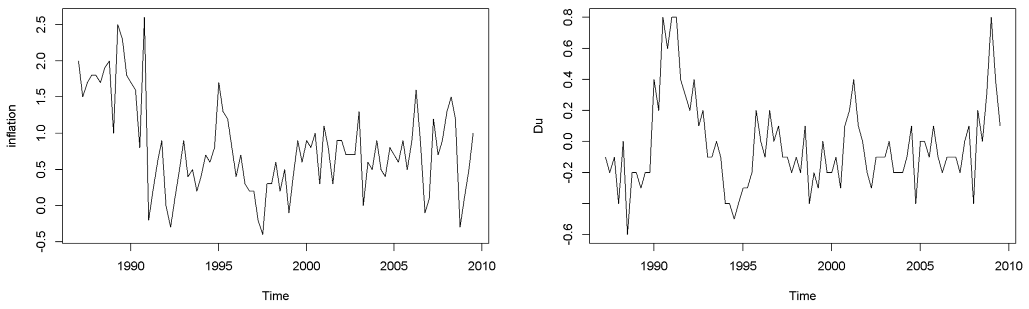


Figure 9.4: Data time plots in the 'phillips' dataset

Table 9.5: Summary of the 'phillips' model

term	estimate	std.error	statistic	p.value
(Intercept)	0.777621	0.065825	11.81347	0.000000
diff(u)	-0.527864	0.229405	-2.30101	0.023754

9.4 satisfies the non-autocorrelation assumption of the time series regression model.

```r
phill.dyn <- dynlm(inf~diff(u),data=phill.ts)
ehat <- resid(phill.dyn)
kable(tidy(phill.dyn), caption="Summary of the `phillips` model")
```

Table 9.5 gives a *p*-value of 0.0238, which is significant at a $5% $ level. But is this *p*-value reliable? Let us investigate the autocorrelation structure of the errors in this model.

```r
plot(ehat)
abline(h=0, lty=2)
```

```r
corrgm <- acf(ehat)
plot(corrgm)
```

The time series plot in Figure 9.5 suggests that some patterns exists in the residuals, which is confirmed by the correlogram in Figure 9.6. The previous result suggesting that there is a significant relationship between inflation and change in unemployment rate may not be, afterall, too reliable.

While visualising the data and plotting the correlogram are powerful methods of spotting autocorrelation, in many applications we need a precise criterion, a test statistic to decide whether autocorrelation is a problem. One such a method is the **Lagrange Multiplier** test. Suppose we want to test for autocorrelation in the

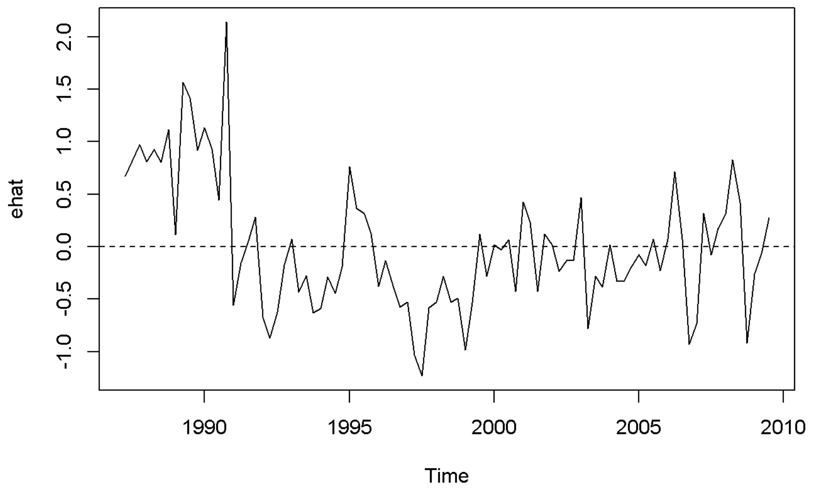


Figure 9.5: Residuals of the Phillips equation

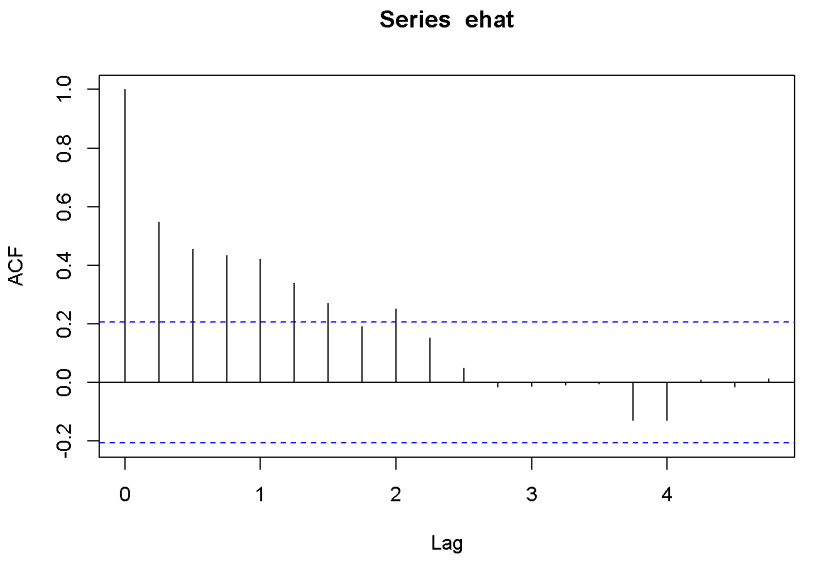


Figure 9.6: Correlogram of the residuals in the Phillips model

residuals the model given in Equation 9.5, where we assume that the errors have the autocorrelation structure described in Equation 9.6.

$$y_t = \beta_1 + \beta_2 x_t + e_t \tag{9.5}$$

$$e_t = \rho e_{t-1} + \nu_t \tag{9.6}$$

A test for autocorrelation would be based on the hypothesis in Equation 9.7.

$$H_0 : \rho = 0, \quad H_A : \rho \neq 0 \tag{9.7}$$

After little algebraic manipulation, the auxiliary regression that the LM test actually uses is the one in Equation 9.8.

$$\hat{e}_t = \gamma_1 + \gamma_2 x_t + \rho \hat{e}_{t-1} + \nu_t \tag{9.8}$$

The test statistic is $T \times R^2$, where R^2 is the coefficient of determination resulted from estimating the auxiliary equation (Equation 9.8). In *R*, all these calculations can be done in one command, `bgtest()`, which is the **Breusch-Godfrey** test for autocorrelation function. This function can test for autocorrelation of higher orders, which requires including higher lags for $\hat{e}$ in the auxiliary equation.

Let us do this test for the Phillips example. The next code sequence does the test first for only one lag and using an F-statistic; then, for lags up to 4, using a χ^2-statistic. *R* does this test by either eliminating the first observations that are necessary to calculate the lags (`fill=NA`), or by setting them equal to zero (`fill=0`).

```
a <- bgtest(phill.dyn, order=1, type="F", fill=0)
b <- bgtest(phill.dyn, order=1, type="F", fill=NA)
c <- bgtest(phill.dyn, order=4, type="Chisq", fill=0)
d <- bgtest(phill.dyn, order=4, type="Chisq", fill=NA)
dfr <- data.frame(rbind(a[c(1,2,4)],
                  b[c(1,2,4)],
                  c[c(1,2,4)],
                  d[c(1,2,4)]
                  ))
dfr <- cbind(c("1, F, 0",
    "1, F, NA", "4, Chisq, 0", "4, Chisq, NA"), dfr)
names(dfr)<-c("Method", "Statistic", "Parameters", "p-Value")
kable(dfr, caption="Breusch-Godfrey test for the Phillips example")
```

Table 9.6: Breusch-Godfrey test for the Phillips example

Method	Statistic	Parameters	p-Value
1, F, 0	38.4654	1, 87	1.82193e-08
1, F, NA	38.6946	1, 86	1.73442e-08
4, Chisq, 0	36.6719	4	2.10457e-07
4, Chisq, NA	33.5937	4	9.02736e-07

All the four tests summarized in Table 9.6 reject the null hypothesis of no autocorrelation. The above code sequence requires a bit of explanation. The first four lines do the BG test on the previously estimated Phillips model (`phill.dyn`), using various parametrs: `order` tells *R* how many lags we want; `type` gives the test statistic to be used, either F or χ^2; and, finally, `fill` tells *R* whether to delete the first observations or replace them with $e = 0$. The first column in Table 9.6 summarizes these options: number of lags, test statistic used, and how the missing observations are handeled.

The remaining of the code sequence is just to create a clear and convenient way of presenting the results of the four tests. As always, to inspect the content of an object like `a` type and run the command `names(object)`.

How many lags should be considered when performing the autocorrelation test? One suggestion would be to limit the test to the number of lags that the correlogram shows to exceed the confidence band.

R can perform another autocorrelation test, **Durbin-Watson**, which is being used less and less today because of its limitations. However, it may be considered when the sample is small. The following command can be used to perform this test:

```
dwtest(phill.dyn)
```

```
##
##  Durbin-Watson test
##
## data:  phill.dyn
## DW = 0.8873, p-value = 2.2e-09
## alternative hypothesis: true autocorrelation is greater than 0
```

9.4 Estimation with Serially Correlated Errors

Similar to the case of heteroskedasticity, autocorrelation in the errors does not produce biased estimates of the coefficients in linear regression, but it produces incorrect standard errors. The similarity with heteroskedasticity goes even further:

Table 9.7: Comparing standard errors for the Phillips model

	Incorrect	vcovHAC	NeweyWest	kernHAC
(Intercept	0.066	0.095	0.128	0.131
Du	0.229	0.304	0.331	0.335

with autocorrelation it is possible to calculate correct (**heteroskedasticity and autocorrelation consistent, HAC**) standard errors, known as **Newey-West** standard errors.

There are several functions in *R* that compute HAC standard errors, of which I choose three, all available in the package `sandwich`.

```r
library(sandwich)
s0 <- coeftest(phill.dyn)
s1 <- coeftest(phill.dyn, vcov.=vcovHAC(phill.dyn))
s2 <- coeftest(phill.dyn, vcov.=NeweyWest(phill.dyn))
s3 <- coeftest(phill.dyn, vcov.=kernHAC(phill.dyn))
tbl <- data.frame(cbind(s0[c(3,4)],s1[c(3,4)],
                        s2[c(3,4)],s3[c(3,4)]))
names(tbl) <- c("Incorrect","vcovHAC", "NeweyWest", "kernHAC")
row.names(tbl) <- c("(Intercept", "Du")
kable(tbl, digits=3,
caption="Comparing standard errors for the Phillips model")
```

Table 9.7 compares three versions of HAC standard errors for the Phillips equation plus the incorrect ones from the initial equation. The differences come from different choices for the methods of calculating them.

Correcting the standard errors in a model with autocorrelated errors does not make the estimator of the coefficients a minimum-variance one. Therefore, we would like to find better estimators, as we did in the case of heteroskedasticity. Let us look at models with a particular structure of the error term , a structure called **first-order autoregressive process**, or **AR(1)model**, described in Equation 9.9. The variable in this process is assumed to have zero mean and constant variance, σ_ν^2, and the errors ν_t should not be autocorrelated. In addition, the **autocorrelation coefficient**, ρ, should take values in the interval $(-1, 1)$. It can be shown that $\rho = corr(e_t, e_{t-1})$.

$$e_t = \rho e_{t-1} + \nu_t \tag{9.9}$$

The following code lines calculate and display the correlation coefficients for the first five lags in the residuals of the Phillips equation (Equation 9.4). Please notice that the autocorrelations tend to become smaller and smaller for distant lags, but they still remain higher than Equation 9.9 implies.

```r
ehat <- resid(phill.dyn)
ac <- acf(ehat, plot=FALSE)
# The Phillips equation: five lag correlations in residuals
ac$acf[2:6]
```

```
## [1] 0.548659 0.455732 0.433216 0.420494 0.339034
```

The correlation coefficient for the first lag is an estimate of the coefficient ρ in the $AR(1)$ process defined in Equation 9.9, $\hat{\rho}_1 = \hat{\rho} = r_1 = 0.549$

9.5 Nonlinear Least Squares Estimation

The simple linear regression model in Equation 9.5 with AR(1) errors defined in Equation 9.6 can be transformed into a model having uncorrelated errors, as Equation 9.10 shows.

$$y_t = \beta_1(1-\rho) + \beta_2 x_t + \rho y_{t-1} - \rho\beta_2 x_{t-1} + \nu_t \tag{9.10}$$

Equation 9.10 is nonlinear in the coefficients, and therefore it needs special methods of estimation. Applying the same transformations to the Phillips model given in Equation 9.4, we obtain its nonlinear version, Equation 9.11.

$$inf_t = \beta_1(1-\rho) + \beta_2 Du_t + \rho inf_{t-1} - \rho\beta_2 Du_{t-1} + \nu_t \tag{9.11}$$

The next code line estimates the non-linear model in Equation 9.11 using the `nls()` function, which requires the data under a data frame form. The first few lines of code create a separate variables for `inf` and `u` and their lags, then brings all of them together in a data frame. The main arguments of the `nls` function are the following: `formula`, a nonlinear function of the regression parameters, `data=` a data frame, and ‘start=list(initial guess values of the parameters), and others.

```r
library(dynlm)
phill.dyn <- dynlm(inf~diff(u), data=phill.ts)
# Non-linear AR(1) model with 'Cochrane-Orcutt method'nls'

phill.ts.tab <- cbind(phill.ts[,"inf"],
                      phill.ts[,"u"],
                      lag(phill.ts[,"inf"], -1),
                      diff(phill.ts[,"u"], lag=1),
                      lag(diff(phill.ts[,2],lag=1), -1)
                      )
phill.dfr <- data.frame(phill.ts.tab)
names(phill.dfr) <- c("inf", "u", "Linf", "Du", "LDu")
phill.nls <- nls(inf~b1*(1-rho)+b2*Du+rho*Linf-
                 rho*b2*LDu,
                 data=phill.dfr,
                 start=list(rho=0.5, b1=0.5, b2=-0.5))
s1 # This is `phill.dyn` with HAC errors:
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.77762    0.09485   8.198 1.82e-12 ***
## diff(u)     -0.52786    0.30444  -1.734   0.0864 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
phill.dyn # The simple linear model:
```

```
##
## Time series regression with "ts" data:
## Start = 1987(2), End = 2009(3)
##
## Call:
## dynlm(formula = inf ~ diff(u), data = phill.ts)
##
## Coefficients:
## (Intercept)      diff(u)
##       0.778       -0.528
```

```
phill.nls # The 'nls' model:
```

```
## Nonlinear regression model
##   model: inf ~ b1 * (1 - rho) + b2 * Du + rho * Linf - rho * b2 * LDu
##    data: phill.dfr
##    rho     b1     b2
##  0.557  0.761 -0.694
##  residual sum-of-squares: 23.2
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 8.06e-06
```

```
coef(phill.nls)[["rho"]]
```

```
## [1] 0.557398
```

Comparing the nonlinear model with the two linear models (with, and without HAC standard errors) shows differences in both coefficients and standard errors. This is an indication that nonlinear estimation is a better choice than HAC standard errors. Please note that the NLS model provides an estimate of the autocorrelation coefficient, $\rho = 0.557$.

9.6 A More General Model

Equation 9.12 gives an **autoregresive distributed lag** model, which is a generalization of the model presented in Equation 9.10. The two models are equivalent under the restriction $\delta_1 = -\theta_1 \delta_0$.

$$y_t = \delta + \theta_1 y_{t-1} + \delta_0 x_t + \delta_1 x_{t-1} + \nu_t \tag{9.12}$$

Equation 9.13 is the Phillips version of the ARDL model given by Equation 9.12.

$$inf_t = \delta + \theta_1 inf_{t-1} + \delta_0 Du_t + \delta_1 Du_{t-1} + \nu_t \tag{9.13}$$

A Wald test can be used to decide if the two models, the nonlinear one and the more general one are equivalent.

```
s.nls <- summary(phill.nls)
phill.gen <- dynlm(inf~L(inf)+d(u)+L(d(u)),
                   data=phill.ts)
s.gen <- summary(phill.gen)
nlW <- nlWaldtest(phill.gen, texts="b[4]=-b[2]*b[3]")
```

The *R* function performing the Wald test is `nlWaldtest` in package `nlWaldTest`, which can test nonlinear restrictions. The result in our case is a χ^2 value of 0.11227, with p-value of 0.737574, which does not reject the null hypothesis that the restriction $\delta_1 = -\theta_1 \delta_0$ cannot be rejected, making, in turn, Equations 9.11 and 9.13 equivalent.

The code lines above use the `L` and `d` from package `dynlm` for constructing lags and differences in time series. Unlike the similar functions that we have previously used (`lag` and `diff`), these do not work outside the command `dynlm`. Please note that constructing lags with `lag()` requires specifying the negative sign of the lag, which is not necessary for the `L()` function..

Table 9.8: Using dynlm with L and d operators

term	estimate	std.error	statistic	p.value
(Intercept)	0.333633	0.089903	3.71104	0.000368
L(inf)	0.559268	0.090796	6.15959	0.000000
d(u)	-0.688185	0.249870	-2.75417	0.007195
L(d(u))	0.319953	0.257504	1.24252	0.217464

Table 9.9: Using dynlm with lag and diff operators

term	estimate	std.error	statistic	p.value
(Intercept)	0.333633	0.089903	3.71104	0.000368
lag(inf, -1)	0.559268	0.090796	6.15959	0.000000
diff(u)	-0.688185	0.249870	-2.75417	0.007195
lag(diff(u), -1)	0.319953	0.257504	1.24252	0.217464

```
phill1.gen <- dynlm(inf~lag(inf, -1)+diff(u)+lag(diff(u), -1), data=phill.ts)
```

The results can be compared in Tables 9.8 and 9.9.

```
kable(tidy(phill.gen),
  caption="Using dynlm with L and d operators")
```

```
kable(tidy(phill1.gen),
  caption="Using dynlm with lag and diff operators")
```

9.7 Autoregressive Models

An autoregressive model of order p, $AR(p)$ (Equation 9.14) is a model with p lags of the response acting as independent variables and no other regressors.

$$y_t = \delta + \theta_1 y_{t-1} + \theta_2 y_{t-2} + ... + \theta_p y_{t-p} + \nu_t \tag{9.14}$$

The next code sequence and Table 9.10 show an autoregressive model of order 2 using the series g in the data file *okun*.

```
data(okun)
okun.ts <- ts(okun)
okun.ar2 <- dynlm(g~L(g)+L(g,2), data=okun.ts)
```

Table 9.10: Autoregressive model of order 2 using the dataset *okun*

term	estimate	std.error	statistic	p.value
(Intercept)	0.4657	0.1433	3.2510	0.0016
L(g)	0.3770	0.1000	3.7692	0.0003
L(g, 2)	0.2462	0.1029	2.3937	0.0187

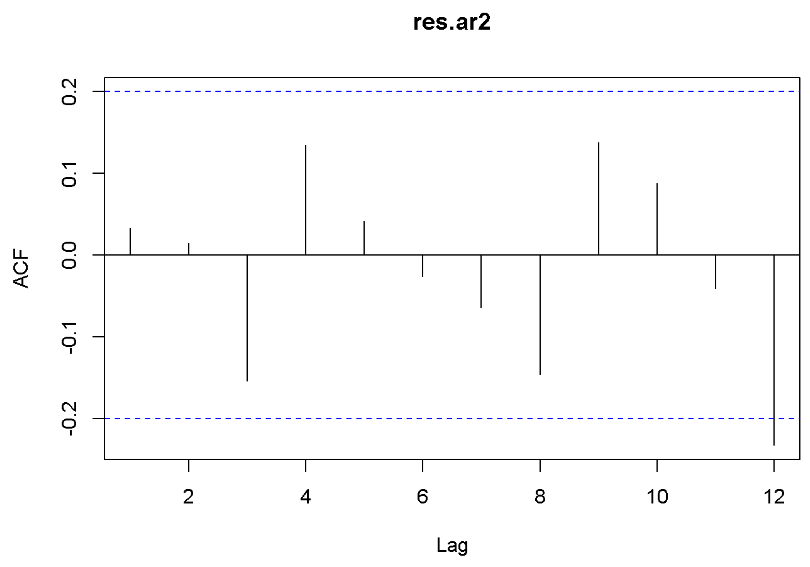


Figure 9.7: Residual correlogram for US GDP AR(2) model

```
kable(tidy(okun.ar2), digits=4,
caption="Autoregressive model of order 2 using the dataset $okun$")
```

How can we decide how many lags to include in an autoregressive model? One way is to look at the correlogram of the series and include the lags that show autocorrelation. The follwing code creates the correlogram in the AR(2) residuals of the US GDP series. The correlogram in Figure 9.7 shows that basically only the first two lags may be included.

```
res.ar2 <- resid(okun.ar2)
Acf(res.ar2, lag.max=12) # New: Acf() from package forecast
```

Another way of choosing the specification of an autoregressive model is to compare models of several orders and choose the one that provides the lowest AIC or BIC value.

Table 9.11: Lag order selection for an AR model

	1	2	3	4	5
AIC	169.4	163.5	163.1	161.6	162.5
BIC	177.1	173.8	175.9	176.9	180.3

```
aics <- rep(0,5)
bics <- rep(0,5)
y <- okun.ts[,"g"]
for (i in 1:5){
  ari <- dynlm(y~L(y,1:i), start=i)
  aics[i] <- AIC(ari)
  bics[i] <- BIC(ari)
}
tbl <- data.frame(rbind(aics, bics))
names(tbl) <- c("1","2","3","4","5")
row.names(tbl) <- c("AIC","BIC")
kable(tbl, digits=1, align='c',
      caption="Lag order selection for an AR model")
```

Table 9.11 displays the AIC and BIC (or SC) values for autoregressive models on the US GDP in dataset *okun* with number of lags from 1 to 5. (As mentioned before, the numbers do not coincide with those in PoE, but the ranking does.) The lowest AIC value indicates that the optimal model should include four lags, while the BIC values indicate the model with only two lags as the winner. Other criteria may be taken into account in such a situation, for instance the correlogram, which agrees with the BIC's choice of model.

The previous code fragment needs some explanation. The first two lines initialize the vectors that are going to hold the AIC and BIC results. The problem of automatically changing the number of regressors is addressed by using the convenient function `L(y, 1:i)`. `AIC()` and `BIC()` are basic *R* functions.

9.8 Forecasting

Let us consider first an autoregressive (AR) model, exemplified by the US GDP growth with two lags specified in Equation 9.15.

$$g_t = \delta + \theta_1 g_{t-1} + \theta_2 g_{t-2} + \nu_t \tag{9.15}$$

Table 9.12: The AR(2) growth model

term	estimate	std.error	statistic	p.value
(Intercept)	0.465726	0.143258	3.25097	0.001602
L(y, 1:2)1	0.377001	0.100021	3.76923	0.000287
L(y, 1:2)2	0.246239	0.102869	2.39372	0.018686

Table 9.13: Forcasts for the AR(2) growth model

	Point.Forecast	Lo.80	Hi.80	Lo.95	Hi.95
99	0.718	0.021	1.415	-0.348	1.784
100	0.933	0.188	1.678	-0.206	2.073
101	0.994	0.202	1.787	-0.218	2.207

Once the coefficients of this model are estimated, they can be used to predict (forecast) out-of-sample, future values of the response variable. Let us do this for periods $T + 1$, $T + 2$, and $T + 3$, where T is the last period in the sample. Equation 9.16 gives the forecast for period s into the future.

$$g_{T+s} = \delta + \theta_1 g_{T+s-1} + \theta_2 g_{T+s-2} + \nu_{T+s} \tag{9.16}$$

```
y <- okun.ts[,"g"]
g.ar2 <- dynlm(y~L(y, 1:2))
kable(tidy(g.ar2), caption="The AR(2) growth model")
```

Table 9.12 shows the results of the AR(2) model. *R* uses the function `forecast()` in package `forecast` to automatically calculate forecasts based on autoregressive or other time series models. One such model is `ar()`, which fits an autoregressive model to a univariate time series. The arguments of `ar()` include: `x=` the name of the time series, `aic=`TRUE, if we want automatic selection of the number of lags based on the AIC information criterion; otherwise, `aic=`FALSE; `order.max=` the maximum lag order in the autoregressive model.

```
ar2g <- ar(y, aic=FALSE, order.max=2, method="ols")
fcst <- data.frame(forecast(ar2g, 3))
kable(fcst, digits=3,
caption="Forcasts for the AR(2) growth model")
```

Table 9.13 shows the forecasted values for three future periods based on the AR(2) growth model.

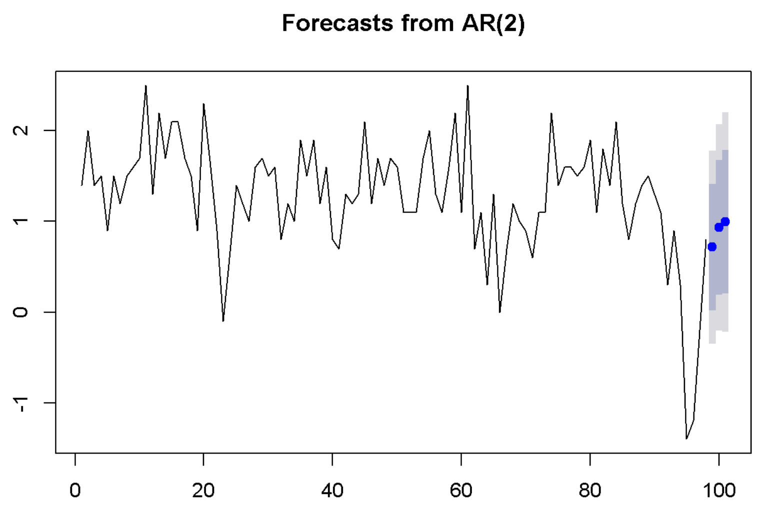


Figure 9.8: Forcasts and confidence intervals for three future periods

```
plot(forecast(ar2g,3))
```

Figure 9.8 illustrates the forecasts produced by the AR(2) model of US GDP and their interval estimates.

Using more information under the form of additional regressors can improve the accuracy of forecasting. We have already studied ARDL models; let us use such a model to forecast the rate of unemployment based on past unemployment and GDP growth data. The dataset is still *okun*, and the model is an ARDL(1,1) as the one in Equation 9.17.

$$Du_t = \delta + \theta_1 Du_{t-1} + \delta_0 g_t + \delta_1 g_{t-1} + \nu_t \tag{9.17}$$

Since we are interested in forecasting *levels* of unemployment, not differences, we want to transform the ARDL(1,1) model in Equation 9.17 into the ARDL(2,1) one in Equation 9.18.

$$u_{T+1} = \delta + \theta_1^* u_T + \theta_2^* u_{T-1} + \delta_0 g_{T+1} + \delta_1 g_T + \nu_{T+1} \tag{9.18}$$

Another forecasting model is the **exponential smoothing** one, which, like the AR model, uses only the variable to be forecasted. In addition, the exponential smoothing model requires a weighting parameter, α, and an initial level of the

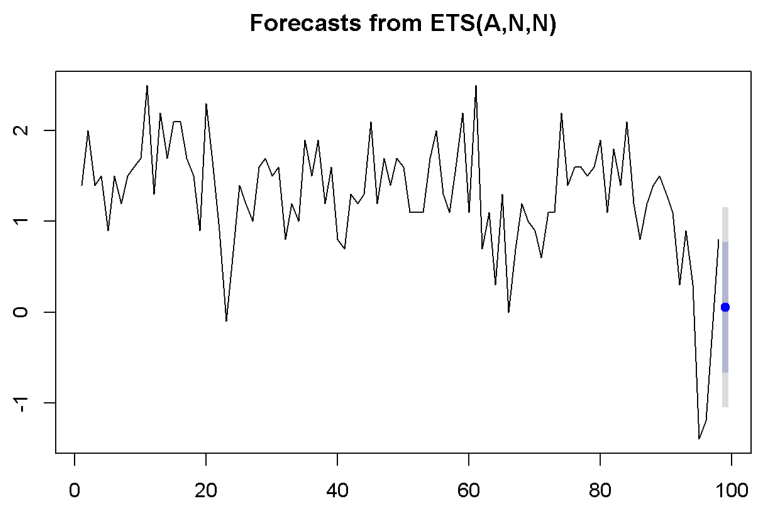


Figure 9.9: Exponential smoothing forecast using 'ets'

forecasted variable, $\hat{y}$. Next period's forecast is a weighted average of this period's level and this period's forecast, as indicated in Equation 9.19.

$$\hat{y}_{T+1} = \alpha y_T + (1 - \alpha)\hat{y}_T \tag{9.19}$$

There are sevaral ways to compute an exponential smoothing forecast in R. I only use two here, for comparison. The first is the function `ets()` in the `forecast` package.

```
y <- okun.ts[,"g"]
okun.ets <- ets(y)
okunf.ets <- forecast(okun.ets,1) #one-period forecast
plot(okunf.ets)
```

The results include the computed value of the weight, $\alpha = 0.381$, the point estimate of the growth rate $g_{T+1} = 0.054$, and the 95 percent interval estimate $(-1.051, 1.158)$. Figure 9.9 illustrates these results.

The second method uses the function `HoltWinters` and is shown in the following code fragment.

```
okun.HW <- HoltWinters(y, beta=FALSE, gamma=FALSE)
plot(okun.HW)
```

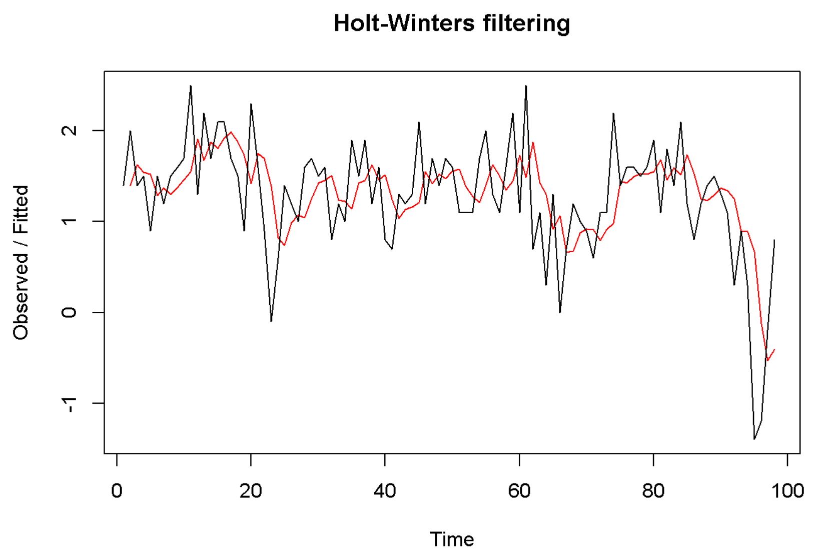


Figure 9.10: Exponential smoothing forecast using 'HoltWinters'

```
okunf.HW <- forecast(okun.HW,1)
```

Similar to the `ets` function, `HoltWinters` automatically determines the optimal weight α of the exponential smoothing model. In our case, the value is $\alpha = 0.381$. The point estimate is $g_{T+1} = 0.054$, and the 95 percent interval estimate is $(-1.06, 1.168)$. Figure 9.10 compares the forecasted with the actual series.

9.9 Multiplier Analysis

In an $ARDL(p, q)$ model, if one variable changes at some period it affects the response over several subsequent periods. Multiplier analysis quantifies these time effects. Let me remind you the form of the $ARDL(p, q)$ model in Equation 9.20.

$$y_t = \delta + \theta_1 y_{t-1} + ... + \theta_p y_{t-p} + \delta_0 x_t + \delta_1 x_{t-1} + ... + \delta_q x_{t-q} + \nu_t \tag{9.20}$$

The model in Equation 9.20 can be transformed by iterative substitution in an infinite distributed lag model, which includes only explanatory variables with no lags of the response. The transformed model is shown in Equation 9.21.

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \beta_3 x_{t-3} + ... + e_t \tag{9.21}$$

Coefficient β_s in Equation 9.21 is called the *s-period delay multiplier*; the sum of the βs from today to period s in the past is called *interim multiplier*, and the sum of all periods to infinity is called *total multiplier*.

Chapter 10

Random Regressors

```r
rm(list=ls()) #Removes all items in Environment!
library(AER) #for `ivreg()`
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for making neat tables with `kable()`
library(stargazer)
```

In most data coming from natural, non-controlled phenomena, both the dependent and independent variables are random. In many cases, some of the independent variables are also correlated with the error term in a regression model, which makes the OLS method inappropriate. Regressors (x) that are correlated with the error term are called **endogeneous**; likewise, those that are not are called **exogeneous**. The remedy for this violation of the linear regression assumption is the use of **instrumental variables**, or **instruments**, which are variables (z) that do not directly influence the response but are correlated with the endogenous regressor in question.

10.1 The Instrumental Variables (IV) Method

A *strong* instrument, one that is highly correlated with the endogenous regressor it concerns, reduces the variance of the estimated coefficient. Assume the multiple regression model in Equation 10.1, where regressors x_2 to x_{K-1} are exogenous and x_K is endogenous. The IV method consists in two stages: first regress x_K on all

Table 10.1: First stage in the 2SLS model for the 'wage' equation

term	estimate	std.error	statistic	p.value
(Intercept)	9.7751	0.4239	23.0605	0.0000
exper	0.0489	0.0417	1.1726	0.2416
I(exper^2)	-0.0013	0.0012	-1.0290	0.3040
mothereduc	0.2677	0.0311	8.5992	0.0000

the other regressors and all the instruments and create the fitted values series, $\hat{x}_K$; second, regress the initial equation, in which x_K is replaced by $\hat{x}_K$. Therefore, the IV method is often called **two-stage least squares**, or **2SLS**.

$$y = \beta_1 + \beta_2 x_2 + ... + \beta_K x_K + e \tag{10.1}$$

Consider the *wage* model in Equation 10.2 using the *mroz* dataset. The notorious difficulty with this model is that the error term may include some unobserved attributes, such as personal ability, that determine both wage and education. In other words, the independent variable *educ* is correlated with the error term, is *endogenous*.

$$log(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 exper^2 + e \tag{10.2}$$

An instrument that may address the endogeneity of *educ* is *mothereduc*, of which we can reasonably assume that it does not directly influence the daughter's wage, but it influences her education.

Let us first carry out an explicit two-stage model with only one instrument, *mothereduc*. The first stage is to regress *educ* on other regressors and the instrument, as Equation 10.3 shows.

$$educ = \gamma_1 + \gamma_2 exper + \gamma_3 exper^2 + \theta_1 mothereduc + \nu_{educ} \tag{10.3}$$

```r
data("mroz", package="PoEdata")
mroz1 <- mroz[mroz$lfp==1,] #restricts sample to lfp=1
educ.ols <- lm(educ~exper+I(exper^2)+mothereduc, data=mroz1)
kable(tidy(educ.ols), digits=4, align='c',caption=
  "First stage in the 2SLS model for the 'wage' equation")
```

The p-value for *mothereduc* is very low (see Table 10.1), indicating a strong correlation between this instrument and the endogenous variable *educ* aven after controling for other variables. The second stage in the two-stage procedure is to create the

Table 10.2: Second stage in the 2SLS model for the 'wage' equation

term	estimate	std.error	statistic	p.value
(Intercept)	0.1982	0.4933	0.4017	0.6881
educHat	0.0493	0.0391	1.2613	0.2079
exper	0.0449	0.0142	3.1668	0.0017
I(exper^2)	-0.0009	0.0004	-2.1749	0.0302

fitted values of *educ* from the first stage (Equation 10.3) and plug them into the model of interest, Euation 10.2 to replace the original variable *educ*.

```r
educHat <- fitted(educ.ols)
wage.2sls <- lm(log(wage)~educHat+exper+I(exper^2), data=mroz1)
kable(tidy(wage.2sls), digits=4, align='c',caption=
  "Second stage in the 2SLS model for the 'wage' equation")
```

The results of the explicit $2SLS$ procedure are shown in Table 10.2; keep n mind, however, that the standard errors calculated in this way are incorrect; the correct method is to use a dedicated software function to solve an instrumental variable model. In *R*, such a function is `ivreg()`.

```r
data("mroz", package="PoEdata")
mroz1 <- mroz[mroz$lfp==1,] #restricts sample to lfp=1.
mroz1.ols <- lm(log(wage)~educ+exper+I(exper^2), data=mroz1)
mroz1.iv <- ivreg(log(wage)~educ+exper+I(exper^2)|
            exper+I(exper^2)+mothereduc, data=mroz1)
mroz1.iv1 <- ivreg(log(wage)~educ+exper+I(exper^2)|
            exper+I(exper^2)+mothereduc+fathereduc,
            data=mroz1)
stargazer(mroz1.ols, wage.2sls, mroz1.iv, mroz1.iv1,
  title="Wage equation: OLS, 2SLS, and IV models compared",
  header=FALSE,
  type=.stargazertype, # "html" or "latex" (in index.Rmd)
  keep.stat="n",  # what statistics to print
  omit.table.layout="n",
  star.cutoffs=NA,
  digits=4,
#  single.row=TRUE,
  intercept.bottom=FALSE, #moves the intercept coef to top
  column.labels=c("OLS","explicit 2SLS", "IV mothereduc",
                "IV mothereduc and fathereduc"),
  dep.var.labels.include = FALSE,
  model.numbers = FALSE,
  dep.var.caption="Dependent variable: wage",
  model.names=FALSE,
  star.char=NULL) #supresses the stars)
```

Table 10.3: Wage equation: OLS, 2SLS, and IV models compared

	Dependent variable: wage			
	OLS	explicit 2SLS	IV mothereduc	IV mothereduc and fathereduc
Constant	−0.5220	0.1982	0.1982	0.0481
	(0.1986)	(0.4933)	(0.4729)	(0.4003)
educ	0.1075		0.0493	0.0614
	(0.0141)		(0.0374)	(0.0314)
educHat		0.0493		
		(0.0391)		
exper	0.0416	0.0449	0.0449	0.0442
	(0.0132)	(0.0142)	(0.0136)	(0.0134)
I(exper^2)	−0.0008	−0.0009	−0.0009	−0.0009
	(0.0004)	(0.0004)	(0.0004)	(0.0004)
Observations	428	428	428	428

The table titled "Wage equation: OLS, 2SLS, and IV compared" shows that the importance of education in determining wage decreases in the IV model. It also shows that the explicit 2SLS model and the IV model with only *mothered* instrument yield the same coefficients (the *educ* in the IV model is equivalent to the *educHat* in 2SLS), but the standard errors are different. The correct ones are those provided by the IV model.

A few observations are in order concerning the above code sequence. First, since some of the individuals are not in the labor force, their wages are zero and the log cannot be calculated. I excluded those observations using only those for which *lpf* is equal to 1. Second, the instrument list in the command `ivreg` includes both the instrument itself (*mothereduc*) and all exogenous regressors, which are, so to speak, their own instruments. The vertical bar character `|` separates the proper regressor list from the instrument list.

Table 10.4: The 'educ' first-stage equation

term	estimate	std.error	statistic	p.value
(Intercept)	9.1026	0.4266	21.3396	0.0000
exper	0.0452	0.0403	1.1236	0.2618
I(exper^2)	-0.0010	0.0012	-0.8386	0.4022
mothereduc	0.1576	0.0359	4.3906	0.0000
fathereduc	0.1895	0.0338	5.6152	0.0000

To test for weak instruments in the *wage* equation, we just test the joint significance of the instruments in an *educ* model as shown in Equation 10.4.

$$educ = \gamma_1 + \gamma_2 exper + \gamma_3 exper^2 + \theta_1 mothereduc + \theta_2 fathereduc + \nu \tag{10.4}$$

```r
educ.ols <- lm(educ~exper+I(exper^2)+mothereduc+fathereduc,
               data=mroz1)
tab <- tidy(educ.ols)
kable(tab, digits=4,
      caption="The 'educ' first-stage equation")
```

```r
linearHypothesis(educ.ols, c("mothereduc=0", "fathereduc=0"))
```

Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
425	2219.22	NA	NA	NA	NA
423	1758.58	2	460.641	55.4003	0

The test rejects the null hypothesis that both *mothereduc* and *fathereduc* coefficients are zero, indicating that at least one instrument is strong. A rule of thumb requires to soundly reject the null hypothesis at a value of the F-statistic greater than 10 or, for only one instrument, a t-statistic greater than 3.16, to make sure that an instrument is strong.

For a model to be identified the number of instruments should be at least equal to the number of endogenous variables. If there are more instruments than endogenous variables, the model is said to be **overidentified**.

10.2 Specification Tests

We have seen before how to test for **weak instruments** with only one instrument. This test can be extended to several instruments. The null hypothesis is H_0: *"All instruments are weak"*.

Since using IV when it is not necessary worsens our estimates, we would like to test whether the variables that worry us are indeed endogenous. This problem is addressed by the **Hausman test for endogeneity**, where the null hypothesis is $H_0 : Cov(x, e) = 0$. Thus, rejecting the null hypothesis indicates the existence of endogeneity and the need for instrumental variables.

The test for the validity of instruments (whether the instruments are corrrelated with the error term) can only be performed for the *extra* instruments, those that are in excess of the number of endogenous variables. This test is sometimes called a test for **overidentifying restrictions**, or the **Sargan** test. The null hypothesis is that the covariance between the instrument and the error term is zero, $H_0 : Cov(z, e) = 0$. Thus, rejecting the null indicates that at least one of the extra instruments is not valid.

R automatically performs these three tests and reports the results in the output to the `ivreg` function.

```r
summary(mroz1.iv1, diagnostics=TRUE)
```

```
##
## Call:
## ivreg(formula = log(wage) ~ educ + exper + I(exper^2) | exper +
##     I(exper^2) + mothereduc + fathereduc, data = mroz1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0986 -0.3196  0.0551  0.3689  2.3493
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.048100   0.400328    0.12   0.9044
## educ         0.061397   0.031437    1.95   0.0515 .
## exper        0.044170   0.013432    3.29   0.0011 **
## I(exper^2)  -0.000899   0.000402   -2.24   0.0257 *
##
## Diagnostic tests:
##                  df1 df2 statistic p-value
## Weak instruments   2 423     55.40  <2e-16 ***
## Wu-Hausman         1 423      2.79   0.095 .
## Sargan             1  NA      0.38   0.539
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.675 on 424 degrees of freedom
## Multiple R-Squared: 0.136,   Adjusted R-squared: 0.13
## Wald test: 8.14 on 3 and 424 DF,  p-value: 0.0000279
```

The results for the wage equation are as follows:

- Weak instruments test: rejects the null, meaning that at least one instrument is strong
- (Wu-)Hausman test for endogeneity: barely rejects the null that the variable of concern is uncorrelated with the error term, indicating that *educ* is marginally endogenous
- Sargan overidentifying restrictions: does not reject the null, meaning that the extra instruments are valid (are uncorrelated with the error term).

The test for weak instruments might be unreliable with more than one endogenous regressor, though, because there is indeed one F-statistic for each endogenous regressor. An alternative is the **Cragg-Donald** test based on the statistic shown in Equation 10.5, where G is the number of exogenous regressors, B is the number of endogenous regressors, L is the number of external instruments, and r_B is the lowest **canonical correlation** (a measure of the correlation between the endogenous and the exogenous variables, calculated by the function `cancor()` in *R*).

$$F = \frac{N-G-B}{L} \frac{r_B^2}{1-r_B^2} \tag{10.5}$$

Let us look at the *hours* equation with two endogenous variables, *mtr* and *educ*, and two external instruments, $mothereduc$ and $fathereduc$. One of the two exogenous regressors, $nwifeinc$, is the family income net of the wife's income; the other exogenous regressor, *mtr*, is the wife's marginal tax rate. Equation 10.6 shows this model; the dataset is *mroz*, restricted to women that are in the labor force.

$$hours = \beta_1 + \beta_2 mtr + \beta_3 educ + \beta_4 kidsl6 + \beta_5 nwifeinc + e \tag{10.6}$$

The next code sequence uses the *R* function `cancor()` to calculate the lowest of two canonical correlations, r_B, which is needed for the Cragg-Donald F-statistic in Equation 10.5.

```r
data("mroz", package="PoEdata")
mroz1 <- mroz[which(mroz$wage>0),]
nwifeinc <- (mroz1$faminc-mroz1$wage*mroz1$hours)/1000
G<-2; L<-2; N<-nrow(mroz1)
x1 <- resid(lm(mtr~kidsl6+nwifeinc, data=mroz1))
x2 <- resid(lm(educ~kidsl6+nwifeinc, data=mroz1))
z1 <-resid(lm(mothereduc~kidsl6+nwifeinc, data=mroz1))
z2 <-resid(lm(fathereduc~kidsl6+nwifeinc, data=mroz1))
```

```
X <- cbind(x1,x2)
Y <- cbind(z1,z2)
rB <- min(cancor(X,Y)$cor)
CraggDonaldF <- ((N-G-L)/L)/((1-rB^2)/rB^2)
```

The result is the Cragg-Donald $F = 0.100806$, which is much smaller than the critical value of 4.58 given in Table 10E.1 of the textbook (Hill, Griffiths, and Lim 2011). This test rejects the null hypothesis of strong instruments, contradicting my previous result.

Chapter 11

Simultaneous Equations Models

```r
rm(list=ls()) #Removes all items in Environment!
library(systemfit)
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 dataset
library(knitr) #for kable()
```

New package: `systemfit` (Henningsen and Hamann 2015).

Simultaneous equations are models with more than one response variable, where the solution is determined by an equilibrium among opposing forces. The econometric problem is similar to the endogenous variables we have studied already in the previous chapter because the mutual interaction between dependent variables can be considered a form of endogeneity. The typical example of an economic simultaneous equation problem is the supply and demand model, where price and quantity are interdependent and are determined by the interaction between supply and demand.

Usually, an economic model such as demand and supply equations include several of the depednedent (endogenous) variables in each equation. Such a model is called the **structural form** of the model. If the structural form is transformed such that each equation shows one dependent variable as a function of only exogenous independent variables, the new form is called the **reduced form**. The reduced form can be estimated by least squares, while the structural form cannot because it includes endogenous variables on its right-hand side.

The **necessary condition for identification** requires that, for the problem to have a solution each equation in the structural form of the system should miss at least an exogenous variable that is present in other equations.

Simultaneous equations are the object of package `systemfit` in R, with the function `systemfit()`, which requires the following main arguments: `formula=` a list describing the equations of the system; `method=` the desired (appropriate) method of estimation, which can be one of "OLS", "WLS", "SUR", "2SLS", "W2SLS", or "3SLS" (we have only studied OLS, WLS, and 2SLS so far); `inst=` a list of instrumental variables under the form of one-sided model formulas; all the endogenous variables in the system must be in this list.

The following example uses the dataset $truffles$, where q is quantity of truffles traded, p is the market price, ps is the price of a substitute, di is income, and pf is a measure of costs of production. The structural demand and supply equations (Equations 11.1 and 11.2) are formulated based on economic theory; quantity and price are endogenous, and all the other variables are considered exogenous.

$$q = \alpha_1 + \alpha_2 p + \alpha_3 ps + \alpha_4 di + e_d \tag{11.1}$$

$$q = \beta_1 + \beta_2 p + \beta_3 pf + e_s \tag{11.2}$$

```
data("truffles", package="PoEdata")
D <- q~p+ps+di
S <- q~p+pf
sys <- list(D,S)
instr <- ~ps+di+pf
truff.sys <- systemfit(sys, inst=instr,
                       method="2SLS", data=truffles)
summary(truff.sys)
```

```
##
## systemfit results
## method: 2SLS
##
##         N DF    SSR detRCov  OLS-R2 McElroy-R2
## system 60 53 692.47  49.803 0.43896    0.80741
##
##      N DF     SSR     MSE   RMSE       R2   Adj R2
## eq1 30 26 631.917 24.3045 4.9300 -0.02395 -0.14210
## eq2 30 27  60.555  2.2428 1.4976  0.90188  0.89461
##
## The covariance matrix of the residuals
##         eq1    eq2
## eq1 24.3045 2.1694
## eq2  2.1694 2.2428
##
```

```
## The correlations of the residuals
##         eq1     eq2
## eq1 1.00000 0.29384
## eq2 0.29384 1.00000
##
##
## 2SLS estimates for 'eq1' (equation 1)
## Model Formula: q ~ p + ps + di
## Instruments: ~ps + di + pf
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.27947    5.54388 -0.7719  0.44712
## p           -0.37446    0.16475 -2.2729  0.03154 *
## ps           1.29603    0.35519  3.6488  0.00116 **
## di           5.01398    2.28356  2.1957  0.03724 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.92996 on 26 degrees of freedom
## Number of observations: 30 Degrees of Freedom: 26
## SSR: 631.91714 MSE: 24.30451 Root MSE: 4.92996
## Multiple R-Squared: -0.02395 Adjusted R-Squared: -0.1421
##
##
## 2SLS estimates for 'eq2' (equation 2)
## Model Formula: q ~ p + pf
## Instruments: ~ps + di + pf
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 20.032802   1.223115  16.378 1.554e-15 ***
## p            0.337982   0.024920  13.563 1.434e-13 ***
## pf          -1.000909   0.082528 -12.128 1.946e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.49759 on 27 degrees of freedom
## Number of observations: 30 Degrees of Freedom: 27
## SSR: 60.55457 MSE: 2.24276 Root MSE: 1.49759
## Multiple R-Squared: 0.90188 Adjusted R-Squared: 0.89461
```

The output of the `systemfit() function shows the estimates by structural equation:`eq1is the demand function, where, as expected, price has a negative sign, and`eq2‘ is the supply equation, with a positive sign for price.

Table 11.1: Reduced form for quantity

term	estimate	std.error	statistic	p.value
(Intercept)	7.8951	3.2434	2.4342	0.0221
ps	0.6564	0.1425	4.6051	0.0001
di	2.1672	0.7005	3.0938	0.0047
pf	-0.5070	0.1213	-4.1809	0.0003

Table 11.2: Reduced form for price

term	estimate	std.error	statistic	p.value
(Intercept)	7.8951	3.2434	2.4342	0.0221
ps	0.6564	0.1425	4.6051	0.0001
di	2.1672	0.7005	3.0938	0.0047
pf	-0.5070	0.1213	-4.1809	0.0003

By evaluating the reduced form equation using OLS, one can determinne the effects of changes in exogenous variables on the **equilibrium** market price and quantity, while the structural equations show the effects of such changes on the quantity demanded, respectively on the quantity supplied. Estimating the structural equations by such methods as 2SLS is, in fact, estimating the market demand and supply curves, which is extremly useful for economic analysis. Estimating the reduced forms, while being useful for prediction, does not allow for deep analysis - it only gives the equilibrium point, not the whole curves.

```r
Q.red <- lm(q~ps+di+pf, data=truffles)
P.red <- lm(q~ps+di+pf, data=truffles)
kable(tidy(Q.red), digits=4,
      caption="Reduced form for quantity")
```

```r
kable(tidy(P.red), digits=4,
      caption="Reduced form for price")
```

Tables 11.1 and 11.2 show that all the exogenous variables have significant effects on the equilibrium quantity and price and have the expected signs.

The $fultonfish$ dataset provides another demand and supply example where the simultaneous equations method can be applied. The purpose of this example is to emphasize that the exogenous variables that are key for identification must be statistically significant. Otherwise, the structural equation that needs to be identified by those variables cannot be reliably estimated. The remaining equations in the structural system are, however, not affected.

$$log(quan) = \alpha_1 + \alpha_2 log(price) + \alpha_3 mon + \alpha_4 tue + \alpha_4 wed + \alpha_5 thu + e_D \tag{11.3}$$

$$log(quan) = \beta_1 + \beta_2 log(price) + \beta_3 stormy + e_S \tag{11.4}$$

In the $fultonfish$ example, the endogenous variables are $lprice$, the log of price, and $lquan$; the exogenous variables are the indicator variables for the day of the week, and whether the catching day was stormy. The identification variable for the demand equation is $stormy$, which will only show up in the supply equation; the identification variables for the supply equation will be mon, tue, wed, and thu.

$$log(q) = \pi_{11} + \pi_{21} mon + \pi_{31} tue + \pi_{41} wed + \pi_{51} thu + \pi_{61} stormy + \nu_1 \tag{11.5}$$

$$log(p) = \pi_{12} + \pi_{22} mon + \pi_{32} tue + \pi_{42} wed + \pi_{52} thu + \pi_{62} stormy + \nu_2 \tag{11.6}$$

Now, let us consider the reduced form equations (Equations 11.5 and 11.6). Since the endogenous variable that appears in the right-hand side of the structural equations (Equations 11.3 and 11.4) is $price$, the $price$ reduced equation (Equation 11.6) is essential for evaluating the identification state of the model. Let us focus on this equation. If the weekday indicators are all insignificant, the supply equation cannot be identified; if $stormy$ turns out insignificant, the demand equation cannot be identified; if the weekday indicators are insignificat but $stormy$ is significant the supply is not identified, but the demand is; if at least one weekday indicator turns out significant but $stormy$ turns out insignificant, the demand equation is not identified but the supply equation is. Equations 11.3 and 11.4 display the structural demand and supply equations for the $fultonfish$ example.

```r
data("fultonfish", package="PoEdata")
fishQ.ols <- lm(lquan~mon+tue+wed+thu+stormy, data=fultonfish)
kable(tidy(fishQ.ols), digits=4,
      caption="Reduced 'Q' equation for the fultonfish example")
```

```r
fishP.ols <- lm(lprice~mon+tue+wed+thu+stormy, data=fultonfish)
kable(tidy(fishP.ols), digits=4,
      caption="Reduced 'P' equation for the fultonfish example")
```

The relevant equation for evaluating identification is shown in Table 11.4, which is the price reduced equation. The results show that the weekday indicators are not

Table 11.3: Reduced 'Q' equation for the fultonfish example

term	estimate	std.error	statistic	p.value
(Intercept)	8.8101	0.1470	59.9225	0.0000
mon	0.1010	0.2065	0.4891	0.6258
tue	-0.4847	0.2011	-2.4097	0.0177
wed	-0.5531	0.2058	-2.6875	0.0084
thu	0.0537	0.2010	0.2671	0.7899
stormy	-0.3878	0.1437	-2.6979	0.0081

Table 11.4: Reduced 'P' equation for the fultonfish example

term	estimate	std.error	statistic	p.value
(Intercept)	-0.2717	0.0764	-3.5569	0.0006
mon	-0.1129	0.1073	-1.0525	0.2950
tue	-0.0411	0.1045	-0.3937	0.6946
wed	-0.0118	0.1069	-0.1106	0.9122
thu	0.0496	0.1045	0.4753	0.6356
stormy	0.3464	0.0747	4.6387	0.0000

significant, which will make the 2SLS estimation of the supply equation unreliable; the coefficient on *stormy* is significant, thus the estimation of the (structural) demand equation will be reliable. The following code sequence and output show the 2SLS estimates of the demand and supply (the structural) equations.

```
fish.D <- lquan~lprice+mon+tue+wed+thu
fish.S <- lquan~lprice+stormy
fish.eqs <- list(fish.D, fish.S)
fish.ivs <- ~mon+tue+wed+thu+stormy
fish.sys <- systemfit(fish.eqs, method="2SLS",
              inst=fish.ivs, data=fultonfish)
summary(fish.sys)
```

```
##
## systemfit results
## method: 2SLS
##
##          N  DF    SSR detRCov  OLS-R2 McElroy-R2
## system 222 213 109.61  0.1073 0.09424   -0.59781
##
##       N  DF    SSR     MSE    RMSE      R2  Adj R2
## eq1 111 105 52.090 0.49610 0.70434 0.13912 0.09813
## eq2 111 108 57.522 0.53261 0.72980 0.04936 0.03176
##
## The covariance matrix of the residuals
##         eq1     eq2
## eq1 0.49610 0.39614
## eq2 0.39614 0.53261
##
## The correlations of the residuals
##         eq1     eq2
## eq1 1.00000 0.77065
## eq2 0.77065 1.00000
##
##
## 2SLS estimates for 'eq1' (equation 1)
## Model Formula: lquan ~ lprice + mon + tue + wed + thu
## Instruments: ~mon + tue + wed + thu + stormy
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  8.505911   0.166167 51.1890 < 2.2e-16 ***
## lprice      -1.119417   0.428645 -2.6115  0.010333 *
## mon         -0.025402   0.214774 -0.1183  0.906077
## tue         -0.530769   0.208000 -2.5518  0.012157 *
## wed         -0.566351   0.212755 -2.6620  0.008989 **
## thu          0.109267   0.208787  0.5233  0.601837
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.70434 on 105 degrees of freedom
## Number of observations: 111 Degrees of Freedom: 105
## SSR: 52.09032 MSE: 0.4961 Root MSE: 0.70434
## Multiple R-Squared: 0.13912 Adjusted R-Squared: 0.09813
##
##
## 2SLS estimates for 'eq2' (equation 2)
## Model Formula: lquan ~ lprice + stormy
## Instruments: ~mon + tue + wed + thu + stormy
##
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.6283544  0.3889702 22.1826   <2e-16 ***
## lprice       0.0010593  1.3095470  0.0008   0.9994
## stormy      -0.3632461  0.4649125 -0.7813   0.4363
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7298 on 108 degrees of freedom
## Number of observations: 111 Degrees of Freedom: 108
## SSR: 57.52184 MSE: 0.53261 Root MSE: 0.7298
## Multiple R-Squared: 0.04936 Adjusted R-Squared: 0.03176
```

In the output of the `2SLS` estimation, `eq1` is the demand equation, and `eq2` is the supply. As we have seen the demand equation is identified, i.e., reliable, while the supply equation is not. A solution might be to find better instruments, other than the weekdays for the demand equation. Finding valid instruments is, however, a difficult task in many problems.

Chapter 12

Time Series: Nonstationarity

```r
rm(list=ls()) #Removes all items in Environment!
library(tseries) # for ADF unit root tests
library(dynlm)
library(nlWaldTest) # for the `nlWaldtest()` function
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for kable()
library(forecast)
```

New package: `tseries` (Trapletti and Hornik 2016).

A time series is nonstationary if its distribution, in particular its mean, variance, or timewise covariance change over time. Nonstationary time series cannot be used in regression models because they may create **spurious regression**, a false relationship due to, for instance, a common trend in otherwise unrelated variables. Two or more nonstationary series can still be part of a regression model if they are **cointegrated**, that is, they are in a stationary relationship of some sort.

We are concerned with testing time series for nonstationarity and finding out how can we transform nonstationary time series such that we can still use them in regression analysis.

```r
data("usa", package="PoEdata")
usa.ts <- ts(usa, start=c(1984,1), end=c(2009,4),
               frequency=4)
Dgdp <- diff(usa.ts[,1])
```

Table 12.1: Time series data frame constructed with 'ts.union'

gdp	inf	f	b	Dgdp	Dinf	Df	Db
3807.4	9.47	9.69	11.19	NA	NA	NA	NA
3906.3	10.03	10.56	12.64	98.9	0.56	0.87	1.45
3976.0	10.83	11.39	12.64	69.7	0.80	0.83	0.00
4034.0	11.51	9.27	11.10	58.0	0.68	-2.12	-1.54
4117.2	10.51	8.48	10.68	83.2	-1.00	-0.79	-0.42
4175.7	9.24	7.92	9.76	58.5	-1.27	-0.56	-0.92

```r
Dinf <- diff(usa.ts[,"inf"])
Df <- diff(usa.ts[,"f"])
Db <- diff(usa.ts[,"b"])
usa.ts.df <- ts.union(gdp=usa.ts[,1], # package tseries
                      inf=usa.ts[,2],
                      f=usa.ts[,3],
                      b=usa.ts[,4],
                      Dgdp,Dinf,Df,Db,
                      dframe=TRUE)

plot(usa.ts.df$gdp)
plot(usa.ts.df$Dgdp)
plot(usa.ts.df$inf)
plot(usa.ts.df$Dinf)
plot(usa.ts.df$f)
plot(usa.ts.df$Df)
plot(usa.ts.df$b)
plot(usa.ts.df$Db)
```

A novelty in the above code sequence is the use of the function `ts.union`, wich binds together several time series, with the possibility of constructing a data frame. Table 12.1 presents the head of this data frame.

```r
kable(head(usa.ts.df),
caption="Time series data frame constructed with 'ts.union'")
```

12.1 AR(1), the First-Order Autoregressive Model

An AR(1) **stochastic process** is defined by Equation 12.1, where the error term is sometimes called "innovation" or "shock."

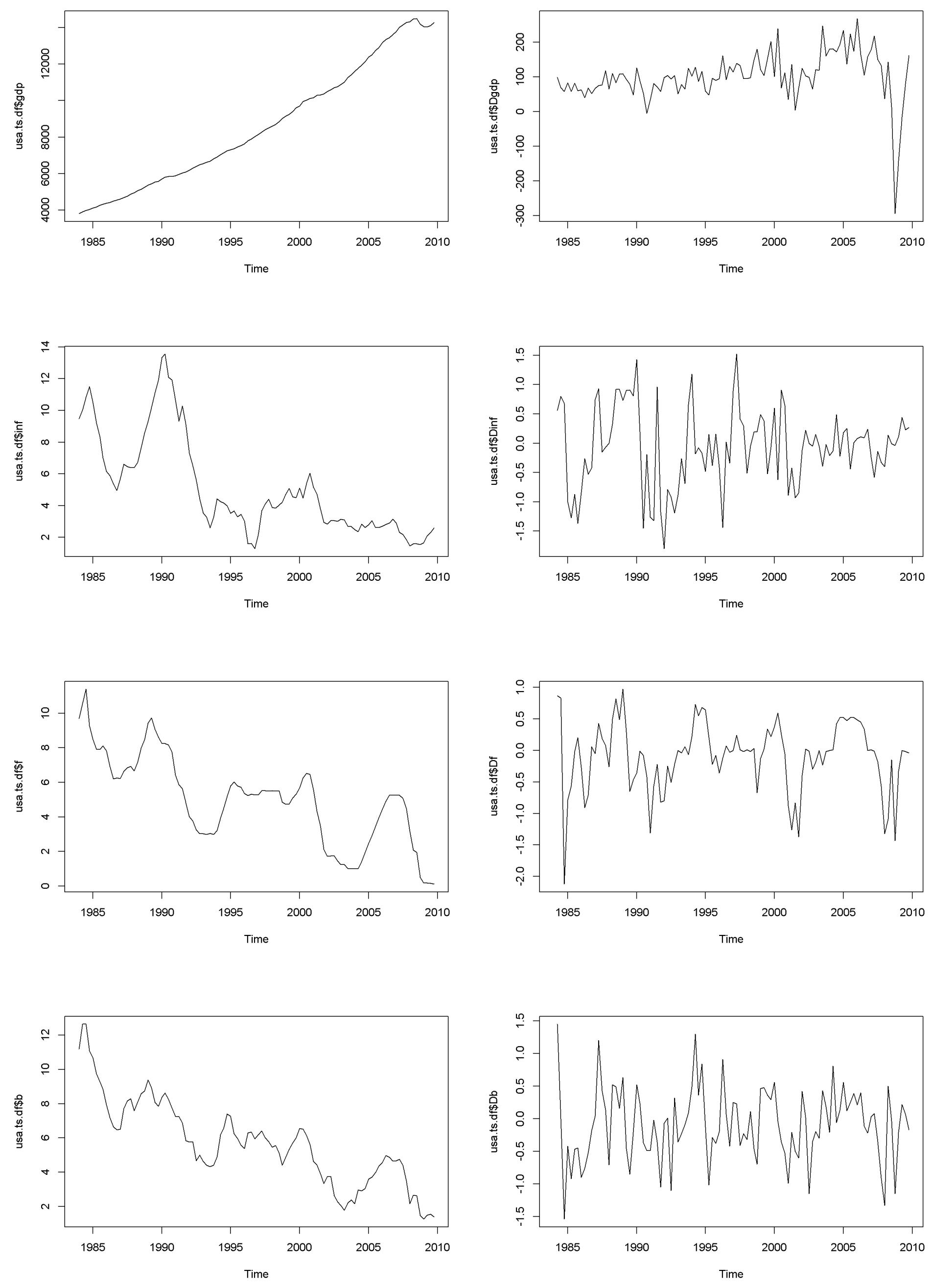


Figure 12.1: Various time series to illustrate nonstationarity

$$y_t = \rho y_{t-1} + \nu_t, \quad |\rho| < 1 \tag{12.1}$$

The AR(1) process is stationary if $|\rho| < 1$; when $\rho = 1$, the process is called **random walk**. The next code piece plots various AR(1) processes, with or without a constant, with or without trend (time as a term in the random process equation), with ρ lesss or equal to 1. The generic equation used to draw the diagrams is given in Equation 12.2.

$$y_t = \alpha + \lambda t + \rho y_{t-1} + \nu_t \tag{12.2}$$

```
N <- 500
a <- 1
l <- 0.01
rho <- 0.7

set.seed(246810)
v <- ts(rnorm(N,0,1))

y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- rho*y[t-1]+v[t]
}
plot(y,type='l', ylab="rho*y[t-1]+v[t]")
abline(h=0)

y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- a+rho*y[t-1]+v[t]
}
plot(y,type='l', ylab="a+rho*y[t-1]+v[t]")
abline(h=0)

y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- a+l*time(y)[t]+rho*y[t-1]+v[t]
}
plot(y,type='l', ylab="a+l*time(y)[t]+rho*y[t-1]+v[t]")
abline(h=0)

y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- y[t-1]+v[t]
}
plot(y,type='l', ylab="y[t-1]+v[t]")
abline(h=0)

a <- 0.1
y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- a+y[t-1]+v[t]
}
plot(y,type='l', ylab="a+y[t-1]+v[t]")
abline(h=0)

y <- ts(rep(0,N))
for (t in 2:N){
  y[t]<- a+l*time(y)[t]+y[t-1]+v[t]
}
plot(y,type='l', ylab="a+l*time(y)[t]+y[t-1]+v[t]")
abline(h=0)
```

12.2 Spurious Regression

Nonstationarity can lead to **spurious regression**, an apparent relationship between variables that are, in reality not related. The following code sequence generates two independent random walk processes, y and x, and regresses y on x.

```
T <- 1000
set.seed(1357)
y <- ts(rep(0,T))
vy <- ts(rnorm(T))
for (t in 2:T){
  y[t] <- y[t-1]+vy[t]
}

set.seed(4365)
x <- ts(rep(0,T))
vx <- ts(rnorm(T))
for (t in 2:T){
  x[t] <- x[t-1]+vx[t]
}
y <- ts(y[300:1000])
```

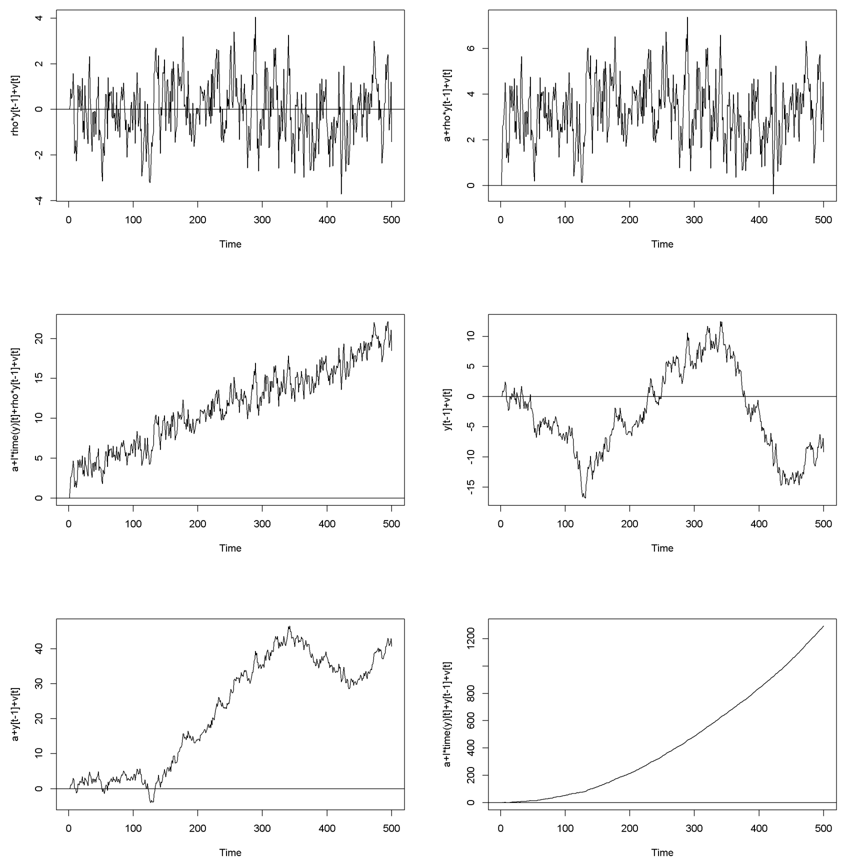


Figure 12.2: Artificially generated AR(1) processes with rho=0.7

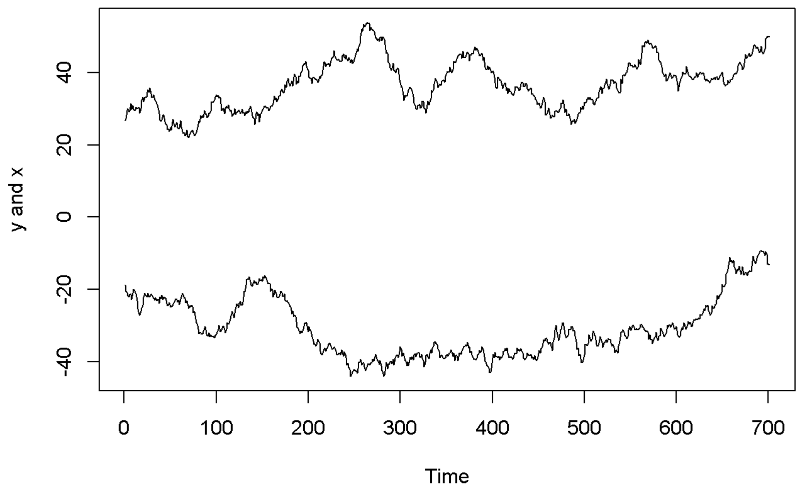


Figure 12.3: Artificially generated independent random variables

```
x <- ts(x[300:1000])
ts.plot(y,x, ylab="y and x")
```

```
spurious.ols <- lm(y~x)
summary(spurious.ols)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -12.55  -5.97  -2.45   4.51  24.68
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.3871     1.6196  -12.59  < 2e-16 ***
## x            -0.2819     0.0433   -6.51  1.5e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.95 on 699 degrees of freedom
```

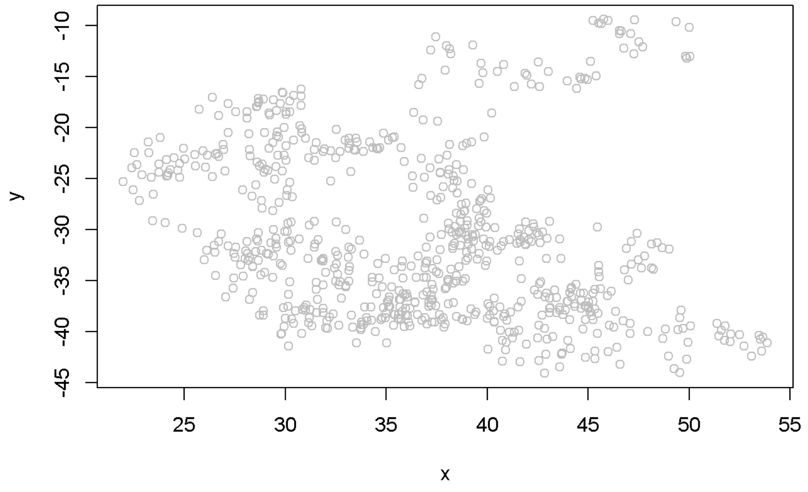


Figure 12.4: Scatter plot of artificial series y and x

```
## Multiple R-squared:  0.0571, Adjusted R-squared:  0.0558
## F-statistic: 42.4 on 1 and 699 DF,  p-value: 1.45e-10
```

The summary output of the regression shows a strong correlation between the two variables, thugh they have been generated independently. (Not any two randomly generated processes need to create spurious regression, though.) Figure 12.3 depicts the two time series, y and x, and Figure 12.4 shows them in a scatterplot.

```r
plot(x, y, type="p", col="grey")
```

12.3 Unit Root Tests for Stationarity

The Dickey-Fuller test for stationarity is based on an AR(1) process as defined in Equation 12.1; if our time series seems to display a constant and trend, the basic equation is the one in Equation 12.2. According to the Dickey-Fuller test, a time series is nonstationary when $\rho = 1$, which makes the AR(1) process a random walk. The null and alternative hypotheses of the test is given in Equation 12.3.

$$H_0 : \rho = 1, \quad H_A : \rho < 1 \tag{12.3}$$

The basic AR(1) equations mentioned above are transformed, for the purpose of the

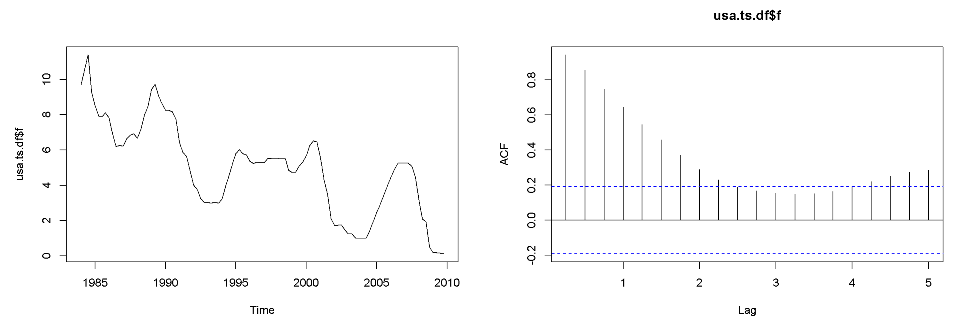


Figure 12.5: A plot and correlogram for series f in dataset usa

DF test into Equation 12.4, with the transformed hypothesis shown in Equation 12.5. **Rejecting the DF null hypothesis implies that our time series is stationary**.

$$\Delta y_t = \alpha + \gamma y_{t-1} + \lambda t + \nu_t \tag{12.4}$$

$$H_0 : \gamma = 0, \quad H_A : \gamma < 0 \tag{12.5}$$

An augmented DF test includes several lags of the variable tested; the number of lags to include can be assessed by examining the correlogram of the variable. The DF test can be of three types: with no constant and no trend, with constsnt and no trend, and, finally, with constant and trend. It is important to specify which DF test we want because the critical values are different for the three different types of the test. One decides which test to perform by examining a time series plot of the variable and determine if an imaginary regression line would have an intercept and a slope.

Let us apply the DF test to the *f* series in the *usa* dataset.

```r
plot(usa.ts.df$f)
Acf(usa.ts.df$f)
```

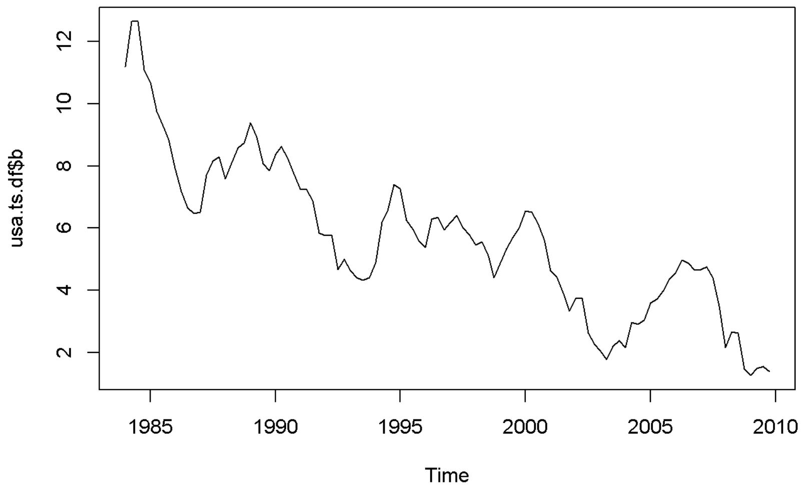
The time series plot in Figure 12.5 indicates both intercept and trend for our series, while the correlogram suggests including 10 lags in the DF test equation. Suppose we choose $\alpha = 0.05$ for the DF test. The `adf.test` function does not require specifying whether the test should be conducted with constant or trend, and if no value for the number of lags is given (the argument for the number of lags is `k`), *R* will calculate a value for it. I would recommend always taking a look at the series' plot and correlogram.

```r
adf.test(usa.ts.df$f, k=10)
```

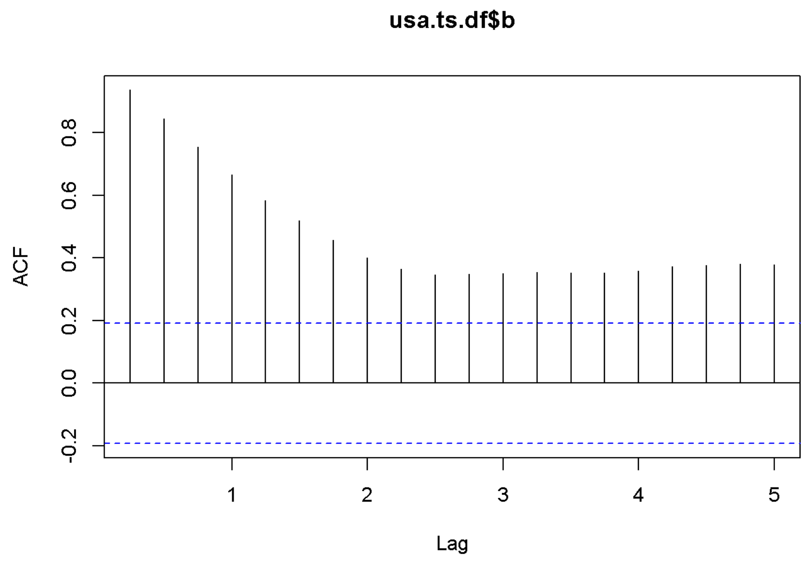
```
##
##  Augmented Dickey-Fuller Test
##
## data:  usa.ts.df$f
## Dickey-Fuller = -3.373, Lag order = 10, p-value = 0.0628
## alternative hypothesis: stationary
```

The result of the test is a p-value greater than our chosen significance level of 0.05; therefore, we cannot reject the null hypothesis of nonstationarity.

```r
plot(usa.ts.df$b)
```



```r
Acf(usa.ts.df$b)
```



```
adf.test(usa.ts.df$b, k=10)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  usa.ts.df$b
## Dickey-Fuller = -2.984, Lag order = 10, p-value = 0.169
## alternative hypothesis: stationary
```

Here is a code to reproduce the results in the textbook.

```
f <- usa.ts.df$f
f.dyn <- dynlm(d(f)~L(f)+L(d(f)))
tidy(f.dyn)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.172522	0.100233	1.72121	0.088337
L(f)	-0.044621	0.017814	-2.50482	0.013884
L(d(f))	0.561058	0.080983	6.92812	0.000000

```
b <- usa.ts.df$b
b.dyn <- dynlm(d(b)~L(b)+L(d(b)))
tidy(b.dyn)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.236873	0.129173	1.83376	0.069693
L(b)	-0.056241	0.020808	-2.70285	0.008091
L(d(b))	0.290308	0.089607	3.23979	0.001629

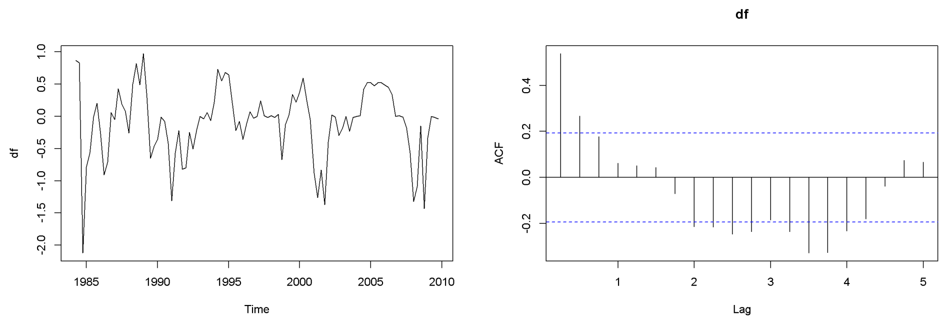


Figure 12.6: Plot and correlogram for series diff(f) in dataset usa

A concept that is closely related to stationarity is **order of integration**, which is how many times we need to difference a series untill it becomes stationary. A series is **I(0)**, that is, integrated of order 0 if it is already stationary (it is stationary *in levels*, not in differences); a series is **I(1)** if it is nonstationary in levels, but stationary in its first differences.

```
df <- diff(usa.ts.df$f)
plot(df)
Acf(df)
adf.test(df, k=2)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df
## Dickey-Fuller = -4.178, Lag order = 2, p-value = 0.01
## alternative hypothesis: stationary
```

```
db <- diff(usa.ts.df$b)
plot(db)
Acf(db)
adf.test(db, k=1)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  db
## Dickey-Fuller = -6.713, Lag order = 1, p-value = 0.01
## alternative hypothesis: stationary
```

Both the plots and the DF tests indicate that the f and b series are stationary in first differences, which makes each of them integrated of order 1. The next code

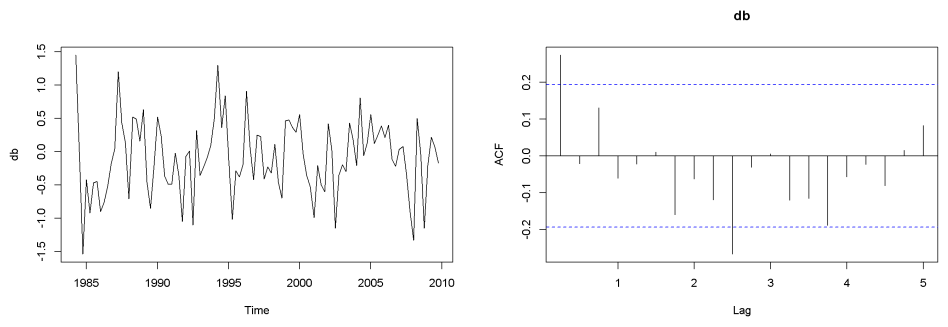


Figure 12.7: Plot and correlogram for series diff(b) in dataset usa

sequence reproduces the results in the textbook. Please note the term (-1) in the `dynlm` command; it tells R that we do not want an intercept in our model. Figures 12.6 and 12.7 show plots of the differenced f and b series, respectively.

```
df.dyn <- dynlm(d(df)~L(df)-1)
db.dyn <- dynlm(d(db)~L(db)-1)
tidy(df.dyn)
```

term	estimate	std.error	statistic	p.value
L(df)	-0.446986	0.081462	-5.48706	0

```
tidy(db.dyn)
```

term	estimate	std.error	statistic	p.value
L(db)	-0.701796	0.091594	-7.66205	0

Function `ndiffs()` in the package `forecast` is a very convenient way of determining the order of integration of a series. The arguments of this function are `x`, a time series, `alpha`, the significacnce level of the test (0.05 by default), `test=` one of "kpss", "adf", or "pp", which indicates the unit root test to be used; we have only studied the "adf" test.), and `max.d=` maximum number of differences. The output of this function is an integer, which is the order of integration of the time series.

```
ndiffs(f)
```

```
## [1] 1
```

```
ndiffs(b)
```

```
## [1] 1
```

As we have already found, the orders of integration for both f and b are 1.

12.4 Cointegration

Two series are cointegrated when their trends are not too far apart and are in some sense similar. This vague statement, though, can be made precise by conducting a cointegration test, which tests whether the residuals from regressing one series on the other one are stationary. If they are, the series are cointegrated. Thus, a cointegration test is in fact a Dickey-Fuler stationarity test on residuals, and its null hypothesis is of noncointegration. In other words, we would like to reject the null hypothesis in a cointegration test, as we wanted in a stationarity test.

Let us apply this method to determine the state of cointegration between the series f and b in dataset *usa*.

```
fb.dyn <- dynlm(b~f)
ehat.fb <- resid(fb.dyn)
ndiffs(ehat.fb) #result: 1
```

```
## [1] 1
```

```
output <- dynlm(d(ehat.fb)~L(ehat.fb)+L(d(ehat.fb))-1) #no constant
foo <- tidy(output)
foo
```

term	estimate	std.error	statistic	p.value
L(ehat.fb)	-0.224509	0.053504	-4.19613	0.000059
L(d(ehat.fb))	0.254045	0.093701	2.71124	0.007891

The relevant statistic is $\tau = -4.196133$, which is less than -3.37, the relevant critical value for the cointegration test. In conclusion, we reject the null hypothesis that the residuals have unit roots, therefore the series are cointegrated.

R has a special function to perform cointegration tests, function `po.test` in package `tseries`. (The name comes from the method it uses, which is called “Phillips-Ouliaris.”) The main argument of the function is a matrix having in its first column the dependent variable of the cointegration equation and the independent variables in the other columns. Let me illustrate its application in the case of the same series fb and f.

```
bfx <- as.matrix(cbind(b,f), demean=FALSE)
po.test(bfx)
```

```
##
##  Phillips-Ouliaris Cointegration Test
##
## data:  bfx
## Phillips-Ouliaris demeaned = -20.51, Truncation lag parameter = 1,
## p-value = 0.0499
```

The PO test marginally rejects the null of no cointegration at the 5 percent level.

12.5 The Error Correction Model

A relationship between cointegrated I(1) variables is a long run relationship, while a relationship between I(0) variables is a short run one. The short run error correction model combines, in some sense, short run and long run effects. Starting from an ARDL(1,1) model (Equation 12.6) and assuming that there is a steady state (long run) relationship between y and x, one can derive the **error correction** model in Equation 12.7, where more lagged differences of x may be necessary to eliminate autocorrelation.

$$y_t = \delta + \theta_1 y_{t-1} + \delta_0 x_t + \delta_1 x_{t-1} + \nu_t \tag{12.6}$$

$$\Delta y_t = -\alpha(y_{t-1} - \beta_1 - \beta_2 x_{t-1}) + \delta_0 \Delta x_t + \nu_t \tag{12.7}$$

In the case of the US bonds and funds example, the error correction model can be constructed as in Equation 12.8.

$$\Delta b_t = -\alpha(b_{t-1} - \beta_1 - \beta_2 f_{t-1}) + \delta_0 \Delta\, f_t + \delta_1 \Delta\, f_{t-1} + \nu_t \tag{12.8}$$

The *R* function that estimates a nonlinear model such as the one in Equation 12.8 is `nls`, which requires three main argumants: a `formula`, which is the regression model to be estimated written using regular text mathematical operators, a `start=` list of guessed or otherwise approximated values of the estimated parameters to initiate a Gauss-Newton numerical optimization process, and `data=` a data frame, list, or environment data source. Please note that `data` cannot be a matrix.

In the next code sequence, the initial values of the parameters have been determined by estimating Equation 12.6 with b and f replacing y and x.

```r
b.ols <- dynlm(L(b)~L(f))
b1ini <- coef(b.ols)[[1]]
b2ini <- coef(b.ols)[[2]]
d.ols <- dynlm(b~L(b)+f+L(f))
aini <- 1-coef(d.ols)[[2]]
d0ini <- coef(d.ols)[[3]]
d1ini <- coef(d.ols)[[4]]
Db <- diff(b)
```

Table 12.2: Parameter estimates in the error correction model

term	estimate	std.error	statistic	p.value
a	0.141877	0.049656	2.85720	0.005230
b1	1.429188	0.624625	2.28807	0.024304
b2	0.776557	0.122475	6.34052	0.000000
d0	0.842463	0.089748	9.38697	0.000000
d1	-0.326845	0.084793	-3.85463	0.000208

Table 12.3: Stationarity test within the error correction model

term	estimate	std.error	statistic	p.value
L(ehat)	-0.168488	0.042909	-3.92668	0.000158
L(d(ehat))	0.179486	0.092447	1.94149	0.055013

```r
Df <- diff(f)
Lb <- lag(b,-1)
Lf <- lag(f,-1)
LDf <- lag(diff(f),-1)
bfset <- data.frame(ts.union(cbind(b,f,Lb,Lf,Db,Df,LDf)))
formula <- Db ~ -a*(Lb-b1-b2*Lf)+d0*Df+d1*LDf
bf.nls <- nls(formula, na.action=na.omit, data=bfset,
          start=list(a=aini, b1=b1ini, b2=b2ini,
                     d0=d0ini, d1=d1ini))
kable(tidy(bf.nls),
caption="Parameter estimates in the error correction model")
```

The error correction model can also be used to test the two series for cointegration. All we need to do is to test the errors of the correction part embedded in Equation 12.8 for stationarity. The estimated errors are given by Equation 12.9.

$$\hat{e}_{t-1} = b_{t-1} - \beta_1 - \beta_2 f_{t-1} \tag{12.9}$$

```r
ehat <- bfset$Lb-coef(bf.nls)[[2]]-coef(bf.nls)[[3]]*bfset$Lf
ehat <- ts(ehat)
ehat.adf <- dynlm(d(ehat)~L(ehat)+L(d(ehat))-1)
kable(tidy(ehat.adf),
caption="Stationarity test within the error correction model")
```

```
foo <- tidy(ehat.adf)
```

To test for cointegration, one should compare the t-ratio of the lagged term shown as ‘statistic’ in Equation 12.3, $t = -3.927$ to the critical value of -3.37. The result is to reject the null of no cointegration, which means the series are cointegrated.

Chapter 13

VEC and VAR Models

```r
rm(list=ls()) #Removes all items in Environment!
library(tseries) # for `adf.test()`
library(dynlm) #for function `dynlm()`
library(vars) # for function `VAR()`
library(nlWaldTest) # for the `nlWaldtest()` function
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for `kable()`
library(forecast)
```

New package: `vars` (Pfaff 2013).

When there is no good reason to assume a one-way causal relationship between two time series variables we may think of their relationship as one of mutual interaction. The concept of "vector," as in vector error correction refers to a number of series in such a model.

13.1 VAR and VEC Models

Equations 13.1 and 13.1 show a generic **vector autoregression** model of order 1, VAR(1), which can be estimated if the series are both I(0). If they are I(1), the same equations need to be estimated in first differences.

$$y_t = \beta_{10} + \beta_{11} y_{t-1} + \beta_{12} x_{t-1} + \nu_t^y \tag{13.1}$$

$$x_t = \beta_{20} + \beta_{21} y_{t-1} + \beta_{22} x_{t-1} + \nu_t^x \tag{13.2}$$

If the two variables in Equations 13.1 and 13.2 and are cointegrated, their cointegration relationship should be taken into account in the model, since it is valuable information; such a model is called **vector error correction**. The cointegration relationship is, remember, as shown in Equation 13.3, where the error term has been proven to be stationary.

$$y_t = \beta_0 + \beta_1 x_t + e_t \tag{13.3}$$

13.2 Estimating a VEC Model

The simplest method is a two-step procedure. First, estimate the cointegrating relationship given in Equation 13.3 and created the lagged resulting residual series $\hat{e}_{t-1} = y_{t-1} - b_0 - b_1 x_{t-1}$. Second, estimate Equations 13.4 and 13.5 by OLS.

$$\Delta y_t = \alpha_{10} + \alpha_{11} + \hat{e}_{t-1} + \nu_t^y \tag{13.4}$$

$$\Delta x_t = \alpha_{20} + \alpha_{21} + \hat{e}_{t-1} + \nu_t^x \tag{13.5}$$

The following example uses the dataset *gdp*, which includes GDP series for Australia and USA for the period since 1970:1 to 2000:4. First we determine the order of integration of the two series.

```r
data("gdp", package="PoEdata")
gdp <- ts(gdp, start=c(1970,1), end=c(2000,4), frequency=4)

ts.plot(gdp[,"usa"],gdp[,"aus"], type="l",
        lty=c(1,2), col=c(1,2))
legend("topleft", border=NULL, legend=c("USA","AUS"),
       lty=c(1,2), col=c(1,2))
```

Figure 13.1 represents the two series in levels, revealing a common trend and, therefore, suggesting that the series are nonstationary.

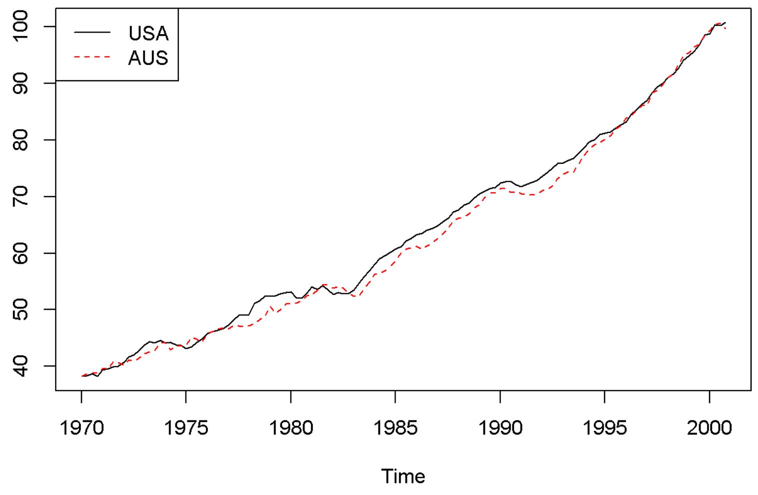


Figure 13.1: Australian and USA GDP series from dataset 'gdp'

```r
adf.test(gdp[,"usa"])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  gdp[, "usa"]
## Dickey-Fuller = -0.9083, Lag order = 4, p-value = 0.949
## alternative hypothesis: stationary
```

```r
adf.test(gdp[,"aus"])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  gdp[, "aus"]
## Dickey-Fuller = -0.6124, Lag order = 4, p-value = 0.975
## alternative hypothesis: stationary
```

```r
adf.test(diff(gdp[,"usa"]))
```

```
##
##  Augmented Dickey-Fuller Test
##
```

Table 13.1: The results of the cointegration equation 'cint1.dyn'

term	estimate	std.error	statistic	p.value
usa	0.985	0.002	594.787	0

```
## data:  diff(gdp[, "usa"])
## Dickey-Fuller = -4.293, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(diff(gdp[,"aus"]))
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(gdp[, "aus"])
## Dickey-Fuller = -4.417, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

The stationarity tests indicate that both series are I(1), Let us now test them for cointegration, using Equations 13.6 and 13.7.

$$aus_t = \beta_1 usa_t + e_t \tag{13.6}$$

$$\hat{e}_t = aus_t - \beta_1 usa_t \tag{13.7}$$

```
cint1.dyn <- dynlm(aus~usa-1, data=gdp)
kable(tidy(cint1.dyn), digits=3,
  caption="The results of the cointegration equation 'cint1.dyn'")
```

```
ehat <- resid(cint1.dyn)
cint2.dyn <- dynlm(d(ehat)~L(ehat)-1)
summary(cint2.dyn)
```

```
##
## Time series regression with "ts" data:
## Start = 1970(2), End = 2000(4)
##
## Call:
## dynlm(formula = d(ehat) ~ L(ehat) - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.4849 -0.3370 -0.0038  0.4656  1.3507
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## L(ehat)  -0.1279     0.0443   -2.89   0.0046 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.598 on 122 degrees of freedom
## Multiple R-squared:  0.064,  Adjusted R-squared:  0.0564
## F-statistic: 8.35 on 1 and 122 DF,  p-value: 0.00457
```

Our test rejects the null of no cointegration, meaning that the series are cointegrated. With cointegrated series we can construct a VEC model to better understand the causal relationship between the two variables.

```
vecaus<- dynlm(d(aus)~L(ehat), data=gdp)
vecusa <- dynlm(d(usa)~L(ehat), data=gdp)
tidy(vecaus)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.491706	0.057909	8.49094	0.000000
L(ehat)	-0.098703	0.047516	-2.07727	0.039893

```
tidy(vecusa)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.509884	0.046677	10.923715	0.000000
L(ehat)	0.030250	0.038299	0.789837	0.431168

The coefficient on the error correction term ($\hat{e}_{t-1}$) is significant for Australia, suggesting that changes in the US economy do affect Australian economy; the error correction coefficient in the US equation is not statistically significant, suggesting that changes in Australia do not influence American economy. To interpret the sign of the error correction coefficient, one should remember that $\hat{e}_{t-1}$ measures the deviation of Australian economy from its cointegrating level of 0.985 of the US economy (see Equations 13.6 and 13.7 and the value of β_1 in Table 13.1).

13.3 Estimating a VAR Model

The VAR model can be used when the variables under study are I(1) but not cointegrated. The model is the one in Equations **??**, but in differences, as specified in Equations 13.8 and 13.9.

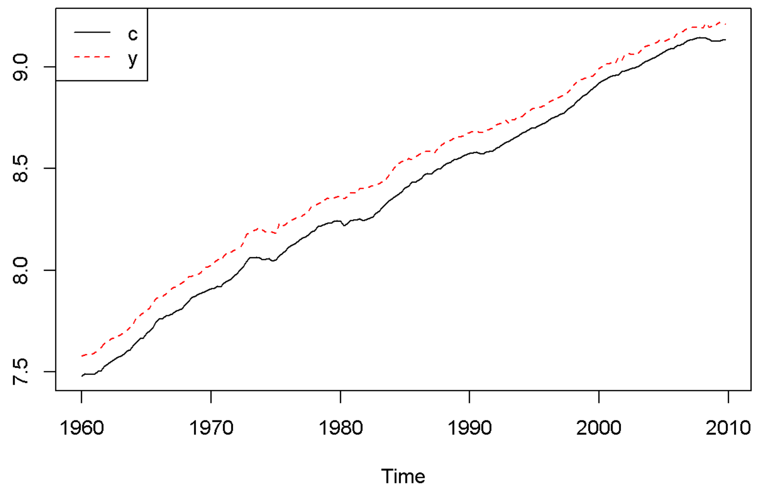


Figure 13.2: Logs of income (y) and consumption (c), dataset 'fred'

$$\Delta y_t = \beta_{11}\Delta y_{t-1} + \beta_{12}\Delta x_{t-1} + \nu_t^{\Delta y} \tag{13.8}$$

$$\Delta x_t = \beta_{21}\Delta y_{t-1} + \beta_{22}\Delta x_{t-1} + \nu_t^{\Delta x} \tag{13.9}$$

Let us look at the income-consumption relationship based on the $fred$ detaset, where consumption and income are already in logs, and the period is 1960:1 to 2009:4. Figure 13.2 shows that the two series both have a trend.

```
data("fred", package="PoEdata")
fred <- ts(fred, start=c(1960,1),end=c(2009,4),frequency=4)
ts.plot(fred[,"c"],fred[,"y"], type="l",
        lty=c(1,2), col=c(1,2))
legend("topleft", border=NULL, legend=c("c","y"),
       lty=c(1,2), col=c(1,2))
```

Are the two series cointegrated?

```
Acf(fred[,"c"])
Acf(fred[,"y"])
adf.test(fred[,"c"])
```

```
##
```

```
##  Augmented Dickey-Fuller Test
##
## data:  fred[, "c"]
## Dickey-Fuller = -2.62, Lag order = 5, p-value = 0.316
## alternative hypothesis: stationary
```

```
adf.test(fred[,"y"])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  fred[, "y"]
## Dickey-Fuller = -2.291, Lag order = 5, p-value = 0.454
## alternative hypothesis: stationary
```

```
adf.test(diff(fred[,"c"]))
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(fred[, "c"])
## Dickey-Fuller = -4.713, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(diff(fred[,"y"]))
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(fred[, "y"])
## Dickey-Fuller = -5.775, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
cointcy <- dynlm(c~y, data=fred)
ehat <- resid(cointcy)
adf.test(ehat)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ehat
## Dickey-Fuller = -2.562, Lag order = 5, p-value = 0.341
## alternative hypothesis: stationary
```

Figure 13.3 shows a long serial correlation sequence; therefore, I will let *R* calculate the lag order in the ADF test. As the results of the above adf and cointegration

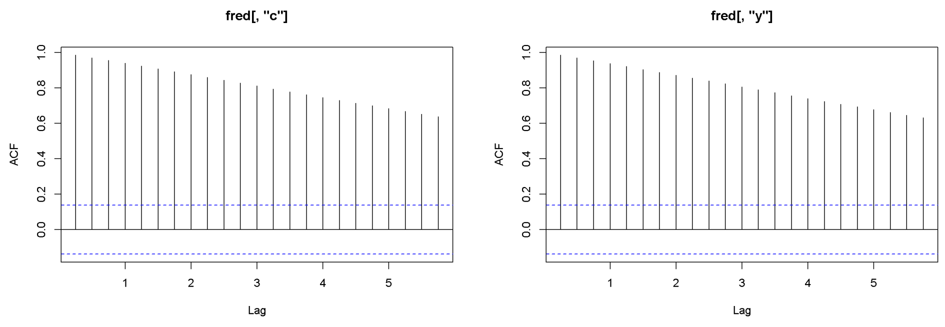


Figure 13.3: Correlograms for the series c and y, dataset fred

tests show, the series are both I(1) but they fail the cointegration test (the series are not cointegrated.) (Plese rememebr that the `adf.test` function uses a constant and trend in the test equation; therefore, the critical values are not the same as in the textbook. However, the results of the tests should be the same most of the time.)

```
library(vars)
Dc <- diff(fred[,"c"])
Dy <- diff(fred[,"y"])
varmat <- as.matrix(cbind(Dc,Dy))
varfit <- VAR(varmat) # `VAR()` from package `vars`
summary(varfit)
```

```
##
## VAR Estimation Results:
## =========================
## Endogenous variables: Dc, Dy
## Deterministic variables: const
## Sample size: 198
## Log Likelihood: 1400.444
## Roots of the characteristic polynomial:
## 0.344 0.343
## Call:
## VAR(y = varmat)
##
##
## Estimation results for equation Dc:
## ===================================
## Dc = Dc.l1 + Dy.l1 + const
##
##       Estimate Std. Error t value Pr(>|t|)
## Dc.l1 0.215607   0.074749    2.88   0.0044 **
## Dy.l1 0.149380   0.057734    2.59   0.0104 *
## const 0.005278   0.000757    6.97  4.8e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 0.00658 on 195 degrees of freedom
## Multiple R-Squared: 0.12,    Adjusted R-squared: 0.111
## F-statistic: 13.4 on 2 and 195 DF,  p-value: 3.66e-06
##
##
## Estimation results for equation Dy:
## ===================================
## Dy = Dc.l1 + Dy.l1 + const
##
##        Estimate Std. Error t value Pr(>|t|)
## Dc.l1  0.475428   0.097326    4.88  2.2e-06 ***
## Dy.l1 -0.217168   0.075173   -2.89   0.0043 **
## const  0.006037   0.000986    6.12  5.0e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 0.00856 on 195 degrees of freedom
## Multiple R-Squared: 0.112,   Adjusted R-squared: 0.103
## F-statistic: 12.3 on 2 and 195 DF,  p-value: 9.53e-06
##
##
##
## Covariance matrix of residuals:
##           Dc        Dy
## Dc 0.0000432 0.0000251
## Dy 0.0000251 0.0000733
##
## Correlation matrix of residuals:
##       Dc    Dy
## Dc 1.000 0.446
## Dy 0.446 1.000
```

Function `VAR()`, which is part of the package `vars` (Pfaff 2013), accepts the following main arguments: `y=` a matrix containing the endogenous variables in the VAR model, `p=` the desired lag order (default is 1), and `exogen=` a matrix of exogenous variables.

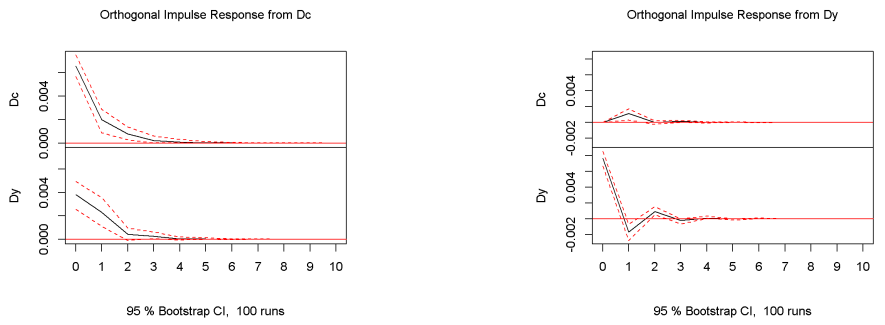


Figure 13.4: Impulse response diagrams for the series c and y, dataset fred

(`VAR` is a more powerful instrument than I imply here; please type `?VAR()` for more information.) The results of a VAR model are more useful in analysing the time response to shocks in the variables, which is the topic of the next section.

13.4 Impulse Responses and Variance Decompositions

Impulse responses are best represented in graphs showing the responses of a VAR endogenous variable in time.

```
impresp <- irf(varfit)
plot(impresp)
```

The interpretation of Figures 13.4 is straightforward: an impulse (shock) to Dc at time zero has large effects the next period, but the effects become smaller and smaller as the time passes. The dotted lines show the 95 percent interval estimates of these effects. The VAR function prints the values corresponding to the impulse response graphs.

```
plot(fevd(varfit)) # `fevd()` is in package `vars`
```

Forecast variance decomposition estimates the contribution of a shock in each variable to the response in both variables. Figure 13.5 shows that almost 100 percent of the variance in Dc is caused by Dc itself, while only about 80 percent in the variance of Dy is caused by Dy and the rest is caused by Dc. The R function `fevd()` in package `vars` allows forecast variance decomposition.

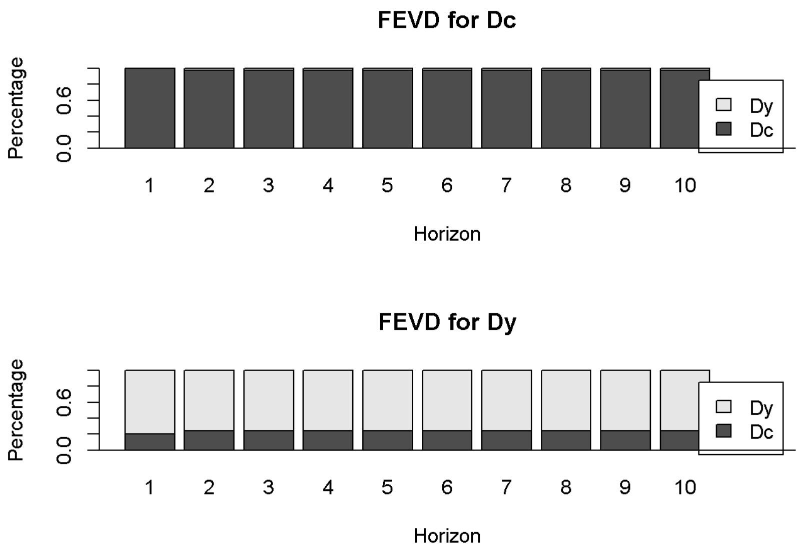


Figure 13.5: Forecast variance decomposition for the series c and y, dataset fred

Chapter 14

Time-Varying Volatility and ARCH Models

```
rm(list=ls()) #Removes all items in Environment!
library(FinTS) #for function `ArchTest()`
library(rugarch) #for GARCH models
library(tseries) # for `adf.test()`
library(dynlm) #for function `dynlm()`
library(vars) # for function `VAR()`
library(nlWaldTest) # for the `nlWaldtest()` function
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for `kable()`
library(forecast)
```

New packages: `FinTS` (Graves 2014) and `rugarch` (Ghalanos 2015).

The **autoregressive conditional heteroskedasticity** (ARCH) model concerns time series with time-varying heteroskedasticity, where variance is conditional on the information existing at a given point in time.

14.1 The ARCH Model

The ARCH model assumes that the conditional mean of the error term in a time series model is constant (zero), unlike the nonstationary series we have discussed so far), but its conditional variance is not. Such a model can be described as in Equations 14.1, 14.2 and 14.3.

$$y_t = \phi + e_t \tag{14.1}$$

$$e_t | I_{t-1} \sim N(0, h_t) \tag{14.2}$$

$$h_t = \alpha_0 + \alpha_1 e_{t-1}^2, \quad \alpha_0 > 0, \;\; 0 \leq \alpha_1 < 1 \tag{14.3}$$

Equations 14.4 and 14.5 give both the test model and the hypotheses to **test for ARCH effects** in a time series, where the residuals $\hat{e}_t$ come from regressing the variable y_t on a constant, such as 14.1, or on a constant plus other regressors; the test shown in Equation 14.4 may include several lag terms, in which case the null hypothesis (Equation 14.5) would be that all of them are jointly insignificant.

$$\hat{e}_t^2 = \gamma_0 + \gamma_1 \hat{e}_{t-1}^2 + ... + \gamma_q e_{t-q}^2 + \nu_t \tag{14.4}$$

$$H_0 : \gamma_1 = ... = \gamma_q = 0 \quad H_A : \gamma_1 \neq 0 \; or \; ...\gamma_q \neq 0 \tag{14.5}$$

The null hypothesis is that there are no ARCH effects. The test statistic is

$$(T - q)R^2 \sim \chi^2_{(1-\alpha, q)}$$

. The following example uses the dataset *byd*, which contains 500 generated observations on the returns to shares in BrightenYourDay Lighting. Figure 14.1 shows a time series plot of the data and histogram.

```r
data("byd", package="PoEdata")
rTS <- ts(byd$r)
plot.ts(rTS)
hist(rTS, main="", breaks=20, freq=FALSE, col="grey")
```

Let us first perform, step by step, the ARCH test described in Equations 14.4 and 14.5, on the variable r from dataset *byd*.

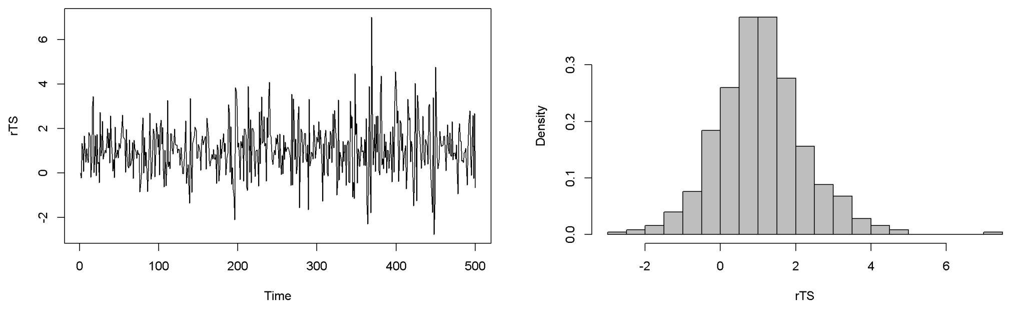


Figure 14.1: Level and histogram of variable 'byd'

```r
byd.mean <- dynlm(rTS~1)
summary(byd.mean)
```

```
##
## Time series regression with "ts" data:
## Start = 1, End = 500
##
## Call:
## dynlm(formula = rTS ~ 1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.847 -0.723 -0.049  0.669  5.931
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.078      0.053    20.4   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.19 on 499 degrees of freedom
```

```r
ehatsq <- ts(resid(byd.mean)^2)
byd.ARCH <- dynlm(ehatsq~L(ehatsq))
summary(byd.ARCH)
```

```
##
## Time series regression with "ts" data:
## Start = 2, End = 500
```

```
##
## Call:
## dynlm(formula = ehatsq ~ L(ehatsq))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.802  -0.950  -0.705   0.320  31.347
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.908      0.124    7.30  1.1e-12 ***
## L(ehatsq)      0.353      0.042    8.41  4.4e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.45 on 497 degrees of freedom
## Multiple R-squared:  0.125,  Adjusted R-squared:  0.123
## F-statistic: 70.7 on 1 and 497 DF,  p-value: 4.39e-16
```

```
T <- nobs(byd.mean)
q <- length(coef(byd.ARCH))-1
Rsq <- glance(byd.ARCH)[[1]]
LM <- (T-q)*Rsq
alpha <- 0.05
Chicr <- qchisq(1-alpha, q)
```

The result is the LM statistic, equal to 62.16, which is to be compared to the critical chi-squared value with $\alpha = 0.05$ and $q = 1$ degrees of freedom; this value is $\chi^2_{(0.95,1)} = 3.84$; this indicates that the null hypothesis is rejected, concluding that the series has ARCH effects.

The same conclusion can be reached if, instead of the step-by-step procedure we use one of *R*'s ARCH test capabilities, the `ArchTest()` function in package `FinTS`.

```
bydArchTest <- ArchTest(byd, lags=1, demean=TRUE)
bydArchTest
```

```
##
##  ARCH LM-test; Null hypothesis: no ARCH effects
##
## data:  byd
## Chi-squared = 62.16, df = 1, p-value = 3.22e-15
```

Function `garch()` in the `tseries` package, becomes an ARCH model when used with the `order=` argument equal to `c(0,1)`. This function can be used to estimate and plot the variance h_t defined in Equation 14.3, as shown in the following code and in Figure 14.2.

```
byd.arch <- garch(rTS,c(0,1))
```

```
##
##  ***** ESTIMATION WITH ANALYTICAL GRADIENT *****
##
##
##      I     INITIAL X(I)        D(I)
##
##      1     1.334069e+00     1.000e+00
##      2     5.000000e-02     1.000e+00
##
##     IT   NF      F         RELDF    PRELDF    RELDX   STPPAR   D*STEP   NPRELDF
##      0    1  5.255e+02
##      1    2  5.087e+02  3.20e-02  7.13e-01  3.1e-01  3.8e+02  1.0e+00  1.34e+02
##      2    3  5.004e+02  1.62e-02  1.78e-02  1.2e-01  1.9e+00  5.0e-01  2.11e-01
##      3    5  4.803e+02  4.03e-02  4.07e-02  1.2e-01  2.1e+00  5.0e-01  1.42e-01
##      4    7  4.795e+02  1.60e-03  1.99e-03  1.3e-02  9.7e+00  5.0e-02  1.36e-02
##      5    8  4.793e+02  4.86e-04  6.54e-04  1.2e-02  2.3e+00  5.0e-02  2.31e-03
##      6    9  4.791e+02  4.16e-04  4.93e-04  1.2e-02  1.7e+00  5.0e-02  1.39e-03
##      7   10  4.789e+02  3.80e-04  4.95e-04  2.3e-02  4.6e-01  1.0e-01  5.36e-04
##      8   11  4.789e+02  6.55e-06  6.73e-06  9.0e-04  0.0e+00  5.1e-03  6.73e-06
##      9   12  4.789e+02  4.13e-08  3.97e-08  2.2e-04  0.0e+00  9.8e-04  3.97e-08
##     10   13  4.789e+02  6.67e-11  6.67e-11  9.3e-06  0.0e+00  4.2e-05  6.67e-11
##
##  ***** RELATIVE FUNCTION CONVERGENCE *****
##
##  FUNCTION     4.788831e+02   RELDX        9.327e-06
##  FUNC. EVALS      13         GRAD. EVALS      11
##  PRELDF       6.671e-11      NPRELDF      6.671e-11
##
##      I      FINAL X(I)        D(I)          G(I)
##
##      1    2.152304e+00     1.000e+00    -2.370e-06
##      2    1.592050e-01     1.000e+00    -7.896e-06
```

```
sbydarch <- summary(byd.arch)
sbydarch
```

```
##
## Call:
## garch(x = rTS, order = c(0, 1))
```

```
##
## Model:
## GARCH(0,1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.459  0.220  0.668  1.079  4.293
##
## Coefficient(s):
##     Estimate  Std. Error  t value Pr(>|t|)
## a0    2.1523      0.1857    11.59   <2e-16 ***
## a1    0.1592      0.0674     2.36    0.018 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Diagnostic Tests:
##  Jarque Bera Test
##
## data:  Residuals
## X-squared = 48.57, df = 2, p-value = 2.85e-11
##
##
##  Box-Ljung test
##
## data:  Squared.Residuals
## X-squared = 0.1224, df = 1, p-value = 0.726
```

```r
hhat <- ts(2*byd.arch$fitted.values[-1,1]^2)
plot.ts(hhat)
```

14.2 The GARCH Model

```r
# Using package `rugarch` for GARCH models
library(rugarch)
garchSpec <- ugarchspec(
           variance.model=list(model="sGARCH",
                               garchOrder=c(1,1)),
           mean.model=list(armaOrder=c(0,0)),
           distribution.model="std")
garchFit <- ugarchfit(spec=garchSpec, data=rTS)
coef(garchFit)
```

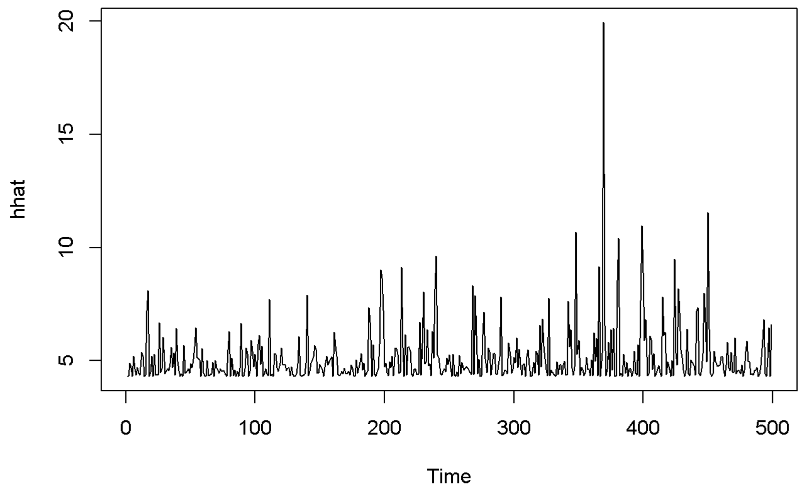


Figure 14.2: Estimated ARCH(1) variance for the 'byd' dataset

```
##        mu     omega    alpha1     beta1     shape
##  1.049280  0.396544  0.495161  0.240827 99.999225
```

```
rhat <- garchFit@fit$fitted.values
plot.ts(rhat)
hhat <- ts(garchFit@fit$sigma^2)
plot.ts(hhat)
```

```
# tGARCH
garchMod <- ugarchspec(variance.model=list(model="fGARCH",
                               garchOrder=c(1,1),
                               submodel="TGARCH"),
           mean.model=list(armaOrder=c(0,0)),
           distribution.model="std")
garchFit <- ugarchfit(spec=garchMod, data=rTS)
coef(garchFit)
```

```
##        mu     omega    alpha1     beta1     eta11     shape
##  0.986682  0.352207  0.390636  0.375328  0.339432 99.999884
```

```
rhat <- garchFit@fit$fitted.values
plot.ts(rhat)
hhat <- ts(garchFit@fit$sigma^2)
plot.ts(hhat)
```

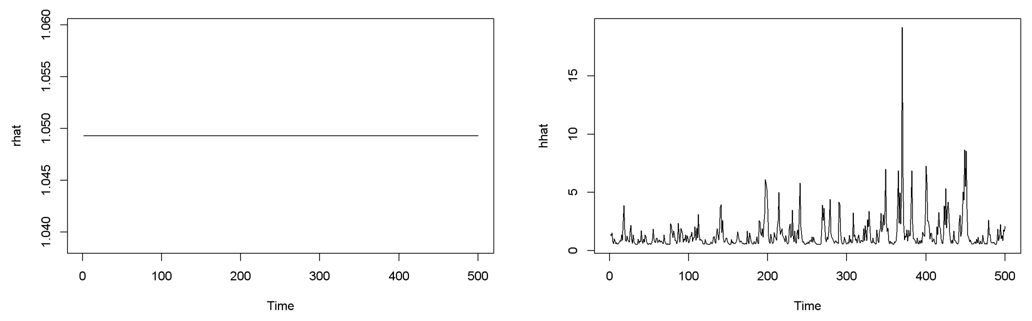


Figure 14.3: Standard GARCH model (sGARCH) with dataset 'byd'

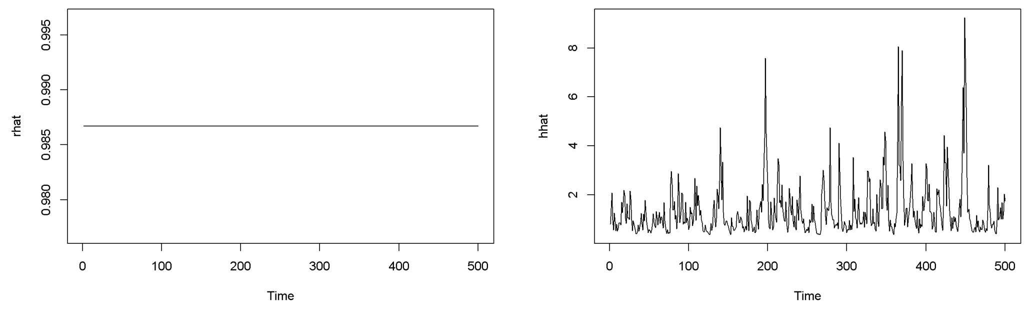


Figure 14.4: The tGARCH model with dataset 'byd'

```r
# GARCH-in-mean
garchMod <- ugarchspec(
          variance.model=list(model="fGARCH",
                               garchOrder=c(1,1),
                               submodel="APARCH"),
           mean.model=list(armaOrder=c(0,0),
                          include.mean=TRUE,
                          archm=TRUE,
                          archpow=2
                          ),
           distribution.model="std"
                           )
garchFit <- ugarchfit(spec=garchMod, data=rTS)
coef(garchFit)
```

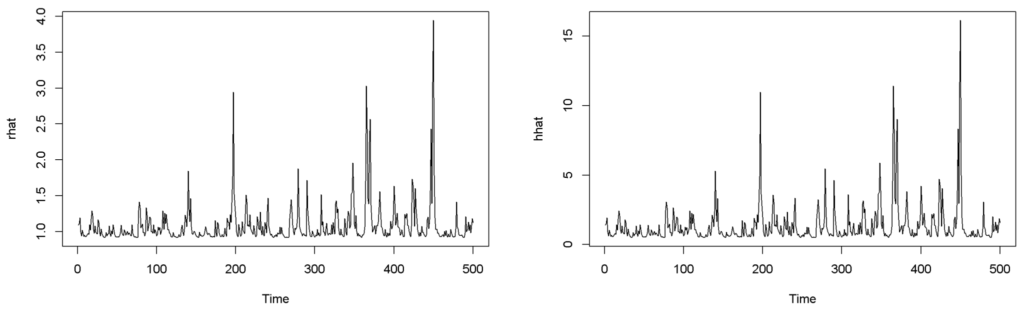


Figure 14.5: A version of the GARCH-in-mean model with dataset 'byd'

```
##        mu     archm     omega    alpha1     beta1     eta11
##  0.820603  0.193476  0.368730  0.442930  0.286748  0.185531
##    lambda     shape
##  1.897586 100.000000
```

```
rhat <- garchFit@fit$fitted.values
plot.ts(rhat)
hhat <- ts(garchFit@fit$sigma^2)
plot.ts(hhat)
```

Figures 14.3, 14.4, and 14.5 show a few versions of the GARCH model. Predictions can be obtained using the function `ugarchboot()` from the package `ugarch`.

Chapter 15

Panel Data Models

```r
rm(list=ls()) #Removes all items in Environment!
library(plm)
library(tseries) # for `adf.test()`
library(dynlm) #for function `dynlm()`
library(vars) # for function `VAR()`
library(nlWaldTest) # for the `nlWaldtest()` function
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for `kable()`
library(forecast)
library(systemfit)
library(AER)
library(xtable)
```

New package: `plm` (Croissant and Millo 2015).

Panel data gathers information about several individuals (cross-sectional **units**) over several **periods**. The panel is **balanced** if all units are observed in all periods; if some units are missing in some periods, the panel is **unbalanced**. Equation 15.1 gives the form of a pooled panel data model, where the subscript $i = 1, ..., N$ denotes an individual (cross sectional unit), and $t = 1, ..., T$ denotes the time period, or longitudinal unit. The total number of observations in the panel is $N \times T$.

$$y_{it} = \beta_1 + \beta_2 x_{2it} + ... + \beta_K x_{Kit} + e_{it} \tag{15.1}$$

15.1 Organizing the Data as a Panel

A **wide** panel has the cross-sectional dimension (N) much larger than the longitudinal dimension (T); when the opposite is true, we have a **long** panel. Normally, the same units are observed in all periods; when this is not the case and each period samples mostly other units, the result is not a proper panel data, but **pooled cross-sections** model.

This manual uses the panel data package `plm()`, which also gives the possibility of organizing the data under the form of a panel. Panel datsets can be organized in mainly two forms: the **long** form has a column for each variable and a row for each individual-period; the **wide** form has a column for each variable-period and a row for each individual. Most panel data methods require the long form, but many data sources provide one wide-form table for each variable; assembling the data from different sources into a long form data frame is often not a trivial matter.

The next code sequence creates a panel structure for the dataset *nls_panel* using the function `pdata.frame` of the `plm` package and displays a small part of this dataset. Please note how the selection of the rows and columns to be displayed is done, using the compact operator $\%in\%$ and arrays such as `c(1:6, 14:15)`. Table 15.1 shows this sample.

```
library(xtable)
data("nls_panel", package="PoEdata")
nlspd <- pdata.frame(nls_panel, index=c("id", "year"))
smpl <- nlspd[nlspd$id %in% c(1,2),c(1:6, 14:15)]
tbl <- xtable(smpl)
kable(tbl, digits=4, align="c",
      caption="A data sample")
```

Function `pdim()` extracts the dimensions of the panel data:

```
pdim(nlspd)
```

```
## Balanced Panel: n=716, T=5, N=3580
```

15.2 The Pooled Model

A **pooled** model has the specification in Equation 15.1, which does not allow for intercept or slope differences among individuals. Such a model can be estimated in R using the specification `pooling` in the `plm()` function, as the following code sequence illustrates.

Table 15.1: A data sample

	id	year	lwage	hours	age	educ	union	exper
1-82	1	82	1.8083	38	30	12	1	7.6667
1-83	1	83	1.8634	38	31	12	1	8.5833
1-85	1	85	1.7894	38	33	12	1	10.1795
1-87	1	87	1.8465	40	35	12	1	12.1795
1-88	1	88	1.8564	40	37	12	1	13.6218
2-82	2	82	1.2809	48	36	17	0	7.5769
2-83	2	83	1.5159	43	37	17	0	8.3846
2-85	2	85	1.9302	35	39	17	0	10.3846
2-87	2	87	1.9190	42	41	17	1	12.0385
2-88	2	88	2.2010	42	43	17	1	13.2115

Table 15.2: Pooled model

term	estimate	std.error	statistic	p.value
(Intercept)	0.477	0.056	8.487	0.000
educ	0.071	0.003	26.567	0.000
exper	0.056	0.009	6.470	0.000
I(exper^2)	-0.001	0.000	-3.176	0.002
tenure	0.015	0.004	3.394	0.001
I(tenure^2)	0.000	0.000	-1.886	0.059
black	-0.117	0.016	-7.426	0.000
south	-0.106	0.014	-7.465	0.000
union	0.132	0.015	8.839	0.000

```r
wage.pooled <- plm(lwage~educ+exper+I(exper^2)+
  tenure+I(tenure^2)+black+south+union,
  model="pooling", data=nlspd)
kable(tidy(wage.pooled), digits=3,
           caption="Pooled model")
```

The `plm()` function accepts the following main arguments, where the parameters shown as vectors `c(...)`, such as `effect` and `model` can only take one value at a time out of the provided list.

```
plm(formula, data, subset, na.action, effect = c("individual",
"time", "twoways"), model = c("within", "random", "ht", "between",
"pooling", "fd"),...)
```

Table 15.3: Pooled 'wage' model with cluster robust standard errors

term	estimate	std.error	statistic	p.value
(Intercept)	0.47660	0.08441	5.64629	0.00000
educ	0.07145	0.00549	13.01550	0.00000
exper	0.05569	0.01129	4.93242	0.00000
I(exper^2)	-0.00115	0.00049	-2.33440	0.01963
tenure	0.01496	0.00711	2.10401	0.03545
I(tenure^2)	-0.00049	0.00041	-1.18697	0.23532
black	-0.11671	0.02808	-4.15602	0.00003
south	-0.10600	0.02701	-3.92422	0.00009
union	0.13224	0.02703	4.89327	0.00000

```r
tbl <- tidy(coeftest(wage.pooled, vcov=vcovHC(wage.pooled,
                    type="HC0",cluster="group")))
kable(tbl, digits=5, caption=
"Pooled 'wage' model with cluster robust standard errors")
```

15.3 The Fixed Effects Model

The fixed effects model takes into account individual differences, translated into different intercepts of the regression line for different individuals. The model in this case assigns the subscript i to the constant term β_1, as shown in Equation 15.2; the constant terms calculated in this way are called **fixed effects**.

$$y_{it} = \beta_{1i} + \beta_{2i}x_{2it} + \beta_{3i}x_{3it} + e_{it} \tag{15.2}$$

Variables that change little or not at all over time, such as some individual characteristics should not be included in a fixed effects model because they produce collinearity with the fixed effects.

```r
nls10 <- pdata.frame(nls_panel[nls_panel$id %in% 1:10,])
```

```
## series nev_mar, not_smsa, south are constants and have been removed
```

```r
wage.fixed <- lm(lwage~exper+I(exper^2)+
                  tenure+I(tenure^2)+union+factor(id)-1,
                  data=nls10)
kable(tidy(wage.fixed), digits=3,
      caption="Fixed effects in a subsample")
```

Table 15.4: Fixed effects in a subsample

term	estimate	std.error	statistic	p.value
exper	0.238	0.188	1.268	0.213
I(exper^2)	-0.008	0.008	-1.036	0.307
tenure	-0.012	0.034	-0.362	0.720
I(tenure^2)	0.002	0.003	0.854	0.399
union	0.114	0.151	0.753	0.457
factor(id)1	0.152	1.097	0.139	0.891
factor(id)2	0.187	1.071	0.174	0.863
factor(id)3	-0.063	1.351	-0.047	0.963
factor(id)4	0.186	1.343	0.138	0.891
factor(id)5	0.939	1.098	0.855	0.398
factor(id)6	0.794	1.112	0.715	0.480
factor(id)7	0.581	1.236	0.470	0.641
factor(id)8	0.538	1.097	0.490	0.627
factor(id)9	0.418	1.084	0.386	0.702
factor(id)10	0.615	1.090	0.564	0.577

Table 15.4 displays the results of an OLS regression on a subsample of the first 10 individuals in the dataset *nls_panel*. The table is generated by the previous code sequence, where the novelty is using the factor variable *id*. The function `factor()` generates dummy variables for all categories of the variable, taking the first category as the reference. To include the reference in the output, one needs to exclude the constant from the regression model by including the term -1 in the regression formula. When the constant is not excluded, the coefficients of the dummy variables represent, as usual, the difference between the respective category and the benchmark one.

However, to estimate a fixed effects in *R* we do not need to create the dummy variables, but use the option `model="within"` in the `plm()` function. The following code fragment uses the whole sample.

```
wage.within <- plm(lwage~exper+I(exper^2)+
                   tenure+I(tenure^2)+south+union,
                   data=nlspd,
                   model="within")
tbl <- tidy(wage.within)
kable(tbl, digits=5, caption=
"Fixed effects using 'within' with full sample")
```

Table 15.5: Fixed effects using 'within' with full sample

term	estimate	std.error	statistic	p.value
exper	0.04108	0.00662	6.20590	0.00000
I(exper^2)	-0.00041	0.00027	-1.49653	0.13463
tenure	0.01391	0.00328	4.24333	0.00002
I(tenure^2)	-0.00090	0.00021	-4.35357	0.00001
south	-0.01632	0.03615	-0.45153	0.65164
union	0.06370	0.01425	4.46879	0.00001

Table 15.6: Fixed effects using the 'within' model option for n=10

term	estimate	std.error	statistic	p.value
exper	0.23800	0.18776	1.26758	0.21332
I(exper^2)	-0.00819	0.00790	-1.03584	0.30738
tenure	-0.01235	0.03414	-0.36171	0.71974
I(tenure^2)	0.00230	0.00269	0.85407	0.39887
union	0.11354	0.15086	0.75263	0.45670

```r
wage10.within <- plm(lwage~exper+I(exper^2)+
                     tenure+I(tenure^2)+union,
                     data=nls10,
                     model="within")
tbl <- tidy(wage10.within)
kable(tbl, digits=5, caption=
  "Fixed effects using the 'within' model option for n=10")
```

Table 15.6 presents the fixed effects model results for the subsample of 10 individuals of the dataset *nls_panel*. This is to be compared to Table 15.4 to see that the `within` method is equiivalent to including the dummies in the model. An interesting comparison is between the pooled and fixed effect models. Comparing Table 15.2 with Table 15.5 one can notice that including accounting for individual heterogeneity significantly lowers the marginal effects of the variables.

Testing if fixed effets are necessary is to compare the fixed effects model `wage.within` with the pooled model `wage.pooled`. The function `pFtest()` does this comparison, as in the following code lines.

```r
kable(tidy(pFtest(wage.within, wage.pooled)), caption=
        "Fixed effects test: Ho:'No fixed effects'")
```

Table 15.7 shows that the null hypothesis of no fixed effects is rejected.

Table 15.7: Fixed effects test: Ho:'No fixed effects'

df1	df2	statistic	p.value	method	alternative
713	2858	15.1875	0	F test for individual effects	significant effects

15.4 The Random Effects Model

The **random effects** model elaborates on the fixed effects model by recognizing that, since the individuals in the panel are randomly selected, their characteristics, measured by the intercept β_{1i} should also be random. Thus, the random effects model assumes the form of the intercept as given in Equation 15.3, where $\overline{\beta}_1$ stands for the population average and u_i represents an individual-specific random term. As in the case of fixed effects, random effects are also time-invariant.

$$\beta_{1i} = \overline{\beta}_1 + u_i \tag{15.3}$$

If this form of the intercept is replaced in Equation 15.2, the result looks like Equation 15.4.

$$y_{it} = \overline{\beta}_1 + \beta_2 x_{2it} + \nu_{it} \tag{15.4}$$

The intercept is here, unlike the fixed effects model constant across individuals, but the error termm, ν_{it}, incorporates both individual specifics and the initial regression error term, as Equation 15.5 shows.

$$\nu_{it} = u_i + e_{it} \tag{15.5}$$

Thus, the random effects model is distinguished by the special structure of its error term: errors have zero mean, a variance equal to $\sigma_u^2 + \sigma_e^2$, uncorrelated across individuals, and having timewise covariance equal to σ_u^2.

An important feature of the random effects model is that the timewise correlation in the errors does not decline over time (see Equation 15.6).

$$\rho = corr(\nu_{it}, \nu_{is}) = \frac{\sigma_u^2}{\sigma_u^2 + \sigma_e^2} \tag{15.6}$$

Testing for random effects amounts to testing the hypothesis that there are no differences among individuals, which implies that the individual-specific random variable has zero variance. Equation 15.7 shows the hypothesis to be tested.

Table 15.8: A random effects test for the wage equation

statistic	p.value	method	alternative
62.1231	0	Lagrange Multiplier Test - (Honda)	significant effects

$$H_0: \ \sigma_u^2 = 0, \quad H_A: \sigma_u^2 > 0 \tag{15.7}$$

The same function we used for fixed effects can be used for random effects, but setting the argument `model=` to 'random' and selecting the `random.method` as one out of four possibilities: "swar" (default), "amemiya", "walhus", or "nerlove". The random effects test function is `plmtest()`, which takes as its main argument the pooling model (indeed it extracts the residuals from the pooling object).

```
wageReTest <- plmtest(wage.pooled, effect="individual")
kable(tidy(wageReTest), caption=
        "A random effects test for the wage equation")
```

Table 15.8 shows that the null hypothesis of zero variance in individual-specific errors is rejected; therefore, heterogeneity among individuals may be significant.

Random effects estimator are reliable under the assumption that individual characteristics (heterogeneity) are exogenous, that is, they are independent with respect to the regressors in the random effects equation. The same Hausman test for endogeneity we have already used in another chapter can be used here as well, with the null hypothesis that individual random effects are exogenous. The test function `phtest()` compares the fixed effects and the random effects models; the next code lines estimate the random effects model and performs the Hausman endogeneity test.

```
wage.random <- plm(lwage~educ+exper+I(exper^2)+
                 tenure+I(tenure^2)+black+south+union,
                 data=nlspd, random.method="swar",
                 model="random")
kable(tidy(wage.random), digits=4, caption=
      "The random effects results for the wage equation")
```

```
kable(tidy(phtest(wage.within, wage.random)), caption=
 "Hausman endogeneity test for the random effects wage model")
```

Table 15.10 shows a low p-value of the test, which indicates that the null hypothesis saying that the individual random effects are exogenous is rejected, which makes the random effects equation inconsistent. In this case the fixed effects model is the correct solution. (The number of parameters in Table 15.10 is given for the time-varying variables only.)

Table 15.9: The random effects results for the wage equation

term	estimate	std.error	statistic	p.value
(Intercept)	0.5339	0.0799	6.6854	0.0000
educ	0.0733	0.0053	13.7454	0.0000
exper	0.0436	0.0064	6.8606	0.0000
I(exper^2)	-0.0006	0.0003	-2.1363	0.0327
tenure	0.0142	0.0032	4.4697	0.0000
I(tenure^2)	-0.0008	0.0002	-3.8785	0.0001
black	-0.1167	0.0302	-3.8652	0.0001
south	-0.0818	0.0224	-3.6518	0.0003
union	0.0802	0.0132	6.0729	0.0000

Table 15.10: Hausman endogeneity test for the random effects wage model

statistic	p.value	parameter	method	alternative
20.745	0.002038	6	Hausman Test	one model is inconsistent

The fixed effects model, however, does not allow time-invariant variables such as *educ* or *black*. Since the problem of the random effects model is endogeneity, one can use instrumental variables methods when time-invariant regressors must be in the model. The **Hausman-Taylor estimator** uses instrumental variables in a random effects model; it assumes four categories of regressors: time-varying exogenous, time-varying endogenous, time-invariant exogenous, and time-invariant endogenous. The number of time-varying variables must be at least equal to the number of time-invariant ones. In our *wage* model, suppose *exper*, *tenure* and *union* are time-varying exogenous, *south* is time-varying endogenous, *black* is time-invariant exogenous, and *educ* is time-invariant endogenous. The same `plm()` function allows carrying out Hausman-Taylor estimation by setting `model=` "ht".

```
wage.HT <- plm(lwage~educ+exper+I(exper^2)+
      tenure+I(tenure^2)+black+south+union |
      exper+I(exper^2)+tenure+I(tenure^2)+union+black,
      data=nlspd, model="ht")
kable(tidy(wage.HT), digits=5, caption=
     "Hausman-Taylor estimates for the wage equation")
```

Table 15.11 shows the results of the Hausman-Taylor estimation, with the largest changes taking place for *educ* and *black*.

Table 15.11: Hausman-Taylor estimates for the wage equation

term	estimate	std.error	statistic	p.value
(Intercept)	-0.75077	0.58624	-1.28066	0.20031
educ	0.17051	0.04446	3.83485	0.00013
exper	0.03991	0.00647	6.16382	0.00000
I(exper^2)	-0.00039	0.00027	-1.46222	0.14368
tenure	0.01433	0.00316	4.53388	0.00001
I(tenure^2)	-0.00085	0.00020	-4.31885	0.00002
black	-0.03591	0.06007	-0.59788	0.54992
south	-0.03171	0.03485	-0.91003	0.36281
union	0.07197	0.01345	5.34910	0.00000

Table 15.12: The head of the grunfeld2 dataset organized as a panel

	inv	v	k	firm	year
1-1935	33.1	1170.6	97.8	1	1935
1-1936	45.0	2015.8	104.4	1	1936
1-1937	77.2	2803.3	118.0	1	1937
1-1938	44.6	2039.7	156.2	1	1938
1-1939	48.1	2256.2	172.6	1	1939
1-1940	74.4	2132.2	186.6	1	1940

15.5 Grunfeld's Investment Example

The dataset $grunfeld2$ is a subset of the initial dataset; it includes two firms, GE and WE observed over the period 1935 to 1954. The purpose of this example is to identify various issues that should be taken into account when building a panel data econometric model. The problem is to find the determinants of investment by a firm , inv_{it} among regressors such as the value of the firm, v_{it}, and capital stock k_{it}. Table 15.12 gives a glimpse of the grunfeld panel data.

```r
data("grunfeld2", package="PoEdata")
grun <- pdata.frame(grunfeld2, index=c("firm","year"))
kable(head(grun), align="c", caption=
"The head of the grunfeld2 dataset organized as a panel")
```

Let us consider a pooling model first, assuming that the coefficients of the regression equation, as well as the error variances are the same for both firms (no individual heterogeneity).

Table 15.13: Grunfeld dataset, pooling panel data results

term	estimate	std.error	statistic	p.value
(Intercept)	17.87200	7.02408	2.54439	0.01525
v	0.01519	0.00620	2.45191	0.01905
k	0.14358	0.01860	7.71890	0.00000

Table 15.14: Grunfeld dataset, 'pooling' panel data results

term	estimate	std.error	statistic	p.value
(Intercept)	-9.9563	23.6264	-0.4214	0.6761
v	0.0266	0.0117	2.2651	0.0300
grun$firm2	9.4469	28.8054	0.3280	0.7450
k	0.1517	0.0194	7.8369	0.0000
v:grun$firm2	0.0263	0.0344	0.7668	0.4485
grun$firm2:k	-0.0593	0.1169	-0.5070	0.6155

```r
grun.pool <- plm(inv~v+k,
                 model="pooling",data=grun)
kable(tidy(grun.pool), digits=5, caption=
  "Grunfeld dataset, pooling panel data results")
```

```r
SSE.pool <- sum(resid(grun.pool)^2)
sigma2.pool <- SSE.pool/(grun.pool$df.residual)
```

For the pooling model, $SSE = 16563.003385$, and $\sigma^2 = 447.64874$.

Allowing for different coefficients across firms but same error structure is the fixed effects model summarized in Table 15.14. Note that the fixed effects are modeled using the function `factor()`.

```r
grun.fe <- plm(inv~v*grun$firm+k*grun$firm,
               model="pooling",data=grun)
kable(tidy(grun.fe), digits=4, caption=
  "Grunfeld dataset, 'pooling' panel data results")
```

```r
SSE.fe <- sum(resid(grun.fe)^2)
sigma2.fe <- SSE.fe/(grun.fe$df.residual)
```

For the fixed effects model with firm dummies, $SSE = 14989.821701$, and $\sigma^2 = 440.877109$.

A test to see if the coefficients are significantly different between the pooling and fixed

Table 15.15: Pooling astimates for the GE firm (firm=1)

term	estimate	std.error	statistic	p.value
(Intercept)	-9.9563	31.3742	-0.3173	0.7548
v	0.0266	0.0156	1.7057	0.1063
k	0.1517	0.0257	5.9015	0.0000

effects equations can be done in *R* using the function `pooltest` from package `plm`; to perform this test, the fixed effects model should be estimated with the function `pvcm` with the argument `model=` "within", as the next code lines show.

```
grun.pvcm <- pvcm(inv~v+k,
                  model="within", data=grun)
coef(grun.pvcm)
```

(Intercept)	v	k
-9.95631	0.026551	0.151694
-0.50939	0.052894	0.092406

```
pooltest(grun.pool, grun.pvcm)
```

```
##
##  F statistic
##
## data:  inv ~ v + k
## F = 1.189, df1 = 3, df2 = 34, p-value = 0.328
## alternative hypothesis: unstability
```

The result shows that the null hypothesis of zero coefficients for the individual dummy terms are zero cannot be rejected. (However, the `pvcm` function is not equivalent to the fixed effects model that uses individual dummies; it is, though, useful for testing the 'poolability' of a dataset.)

Now, if we allow for different coefficients *and* different error variances, the equations for each individual is independent from those for other individuals and it can be estimated separately.

```
grun1.pool <- plm(inv~v+k, model="pooling",
                subset=grun$firm==1, data=grun)
SSE.pool1<- sum(resid(grun1.pool)^2)
sig2.pool1 <- SSE.pool1/grun1.pool$df.residual
kable(tidy(grun1.pool), digits=4, align='c', caption=
        "Pooling astimates for the GE firm (firm=1)")
```

Table 15.16: Pooling estimates for the WE firm (firm=2)

term	estimate	std.error	statistic	p.value
(Intercept)	-0.5094	8.0153	-0.0636	0.9501
v	0.0529	0.0157	3.3677	0.0037
k	0.0924	0.0561	1.6472	0.1179

```
grun2.pool <- plm(inv~v+k, model="pooling",
                 subset=grun$firm==2, data=grun)
SSE.pool2 <- sum(resid(grun2.pool)^2)
sig2.pool2 <- SSE.pool2/grun2.pool$df.residual
kable(tidy(grun2.pool), digits=4, align='c', caption=
        "Pooling estimates for the WE firm (firm=2)")
```

Tables 15.15 and 15.16 show the results for the equations on subsets of data, separated by firms.

A Godfeld-Quandt test can be carried out to determine whether the variances are different among firms, as the next code shows.

```
gqtest(grun.pool, point=0.5, alternative="two.sided",
       order.by=grun$firm)
```

```
##
##  Goldfeld-Quandt test
##
## data:  grun.pool
## GQ = 0.1342, df1 = 17, df2 = 17, p-value = 0.000143
```

The result is rejection of the null hypothesis that the variances are equal, indicating that estimating separate equations for each firm is the correct model.

What happens when we assume that the only link between the two firms is correlation between their contemporaneous error terms? This is the model of **seemingly unrelated regressions**, a generalized least squares method.

```
library("systemfit")
grunf<- grunfeld2
grunf$Firm<-"WE"
for (i in 1:40){
  if(grunf$firm[i]==1){grunf$Firm[i] <- "GE"}
}
grunf$firm <- NULL
names(grunf)<- c("inv", "val", "cap", "year", "firm")
```

```
grunfpd <- plm.data(grunf, c("firm","year"))
grunf.SUR <- systemfit(inv~val+cap, method="SUR", data=grunfpd)
summary(grunf.SUR, resdCov=FALSE, equations=FALSE)
```

```
##
## systemfit results
## method: SUR
##
##         N DF   SSR detRCov  OLS-R2 McElroy-R2
## system 40 34 15590   35641 0.69897     0.6151
##
##     N DF     SSR    MSE   RMSE      R2  Adj R2
## GE 20 17 13788.4 811.08 28.479 0.69256 0.65639
## WE 20 17  1801.3 105.96 10.294 0.74040 0.70986
##
## The covariance matrix of the residuals used for estimation
##        GE     WE
## GE 777.45 207.59
## WE 207.59 104.31
##
## The covariance matrix of the residuals
##        GE     WE
## GE 811.08 224.28
## WE 224.28 105.96
##
## The correlations of the residuals
##         GE      WE
## GE 1.00000 0.76504
## WE 0.76504 1.00000
##
##
## Coefficients:
##                  Estimate Std. Error t value   Pr(>|t|)
## GE_(Intercept) -27.719317  29.321219 -0.9454   0.357716
## GE_val           0.038310   0.014415  2.6576   0.016575 *
## GE_cap           0.139036   0.024986  5.5647 0.00003423 ***
## WE_(Intercept)  -1.251988   7.545217 -0.1659   0.870168
## WE_val           0.057630   0.014546  3.9618   0.001007 **
## WE_cap           0.063978   0.053041  1.2062   0.244256
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

First, please note that the `systemfit()` function requires a panel data file created with `plm.data`, instead of the `pdata.frame` that we have used above; second, for some reason I had to change the names of the variables to names having more than one letter to make the function work. I did this using the function `names()`.

Chapter 16

Qualitative and LDV Models

```r
rm(list=ls()) #Removes all items in Environment!
library(nlWaldTest) # for the `nlWaldtest()` function
library(lmtest) #for `coeftest()` and `bptest()`.
library(broom) #for `glance(`) and `tidy()`
library(PoEdata) #for PoE4 datasets
library(car) #for `hccm()` robust standard errors
library(sandwich)
library(knitr) #for `kable()`
library(forecast)
library(AER)
library(xtable)
```

16.1 The Linear Probability Model

Suppose the response variable is binary, as defined in Equation 16.1.

$$y = 1 \text{ if an individual chooses to buy a house} y = 0 \text{ if an individual chooses not to buy} \tag{16.1}$$

The linear probability model has the general form is shown in Equation 16.2. $E(y)$ is the probability that the response variable takes the value of 1; therefore, a predicted value of y is a prediction for the probability that $y = 1$.

$$y = E(y) + e = \beta_1 + \beta_2 x_2 + ... + \beta_k x_k + e \tag{16.2}$$

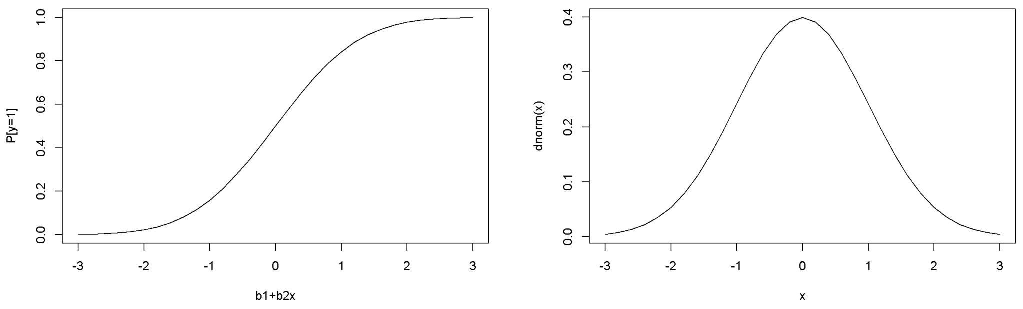


Figure 16.1: The shape of the probit function is the standard normal distribution

16.2 The Probit Model

The **probit** model assumes a nonlinear relationship between the response variable and regressors, this relationship being the cumulative distribution function of the normal distribution (see Equation 16.3 and Figure 16.1, left).

$$p = P[y = 1] = E(y|x) = \Phi(\beta_1 + \beta_2 x) \tag{16.3}$$

The slope of the regression curve is not constant, but is given by the standard normal density function (Figure 16.1, right); the slope can be calculated using Equation 16.4.

$$\frac{dp}{dx} = \phi(\beta_1 + \beta_2 x)\beta_2 \tag{16.4}$$

Predictions of the probability that $y = 1$ are given by Equation 16.5.

$$\hat{p} = \Phi(\hat{\beta}_1 + \hat{\beta}_2 x) \tag{16.5}$$

```r
x <- seq(-3,3, .2)
plot(x, pnorm(x), type="l", xlab="b1+b2x", ylab="P[y=1]")
plot(x, dnorm(x), type="l")
```

16.3 The Transportation Example

The dataset *transport* containes $N = 21$ observations of transportation chioces ($auto = 1$ if individual i chooses *auto* and 0 if individual i chooses *bus*). The

Table 16.1: Transport example, estimated by probit

term	estimate	std.error	statistic	p.value
(Intercept)	-0.0644	0.4007	-0.1608	0.8722
dtime	0.3000	0.1029	2.9154	0.0036

choice depends on the difference in time between the two means of transportation, $dtime = (bustime - autotime) \div 10$.

The R function to estimate a probit model is `glm`, with the `family` argument equal to `binomial(link="probit")`. The `glm` function has the following general structure:

glm(formula, family, data, ...)

```
data("transport", package="PoEdata")
auto.probit <- glm(auto~dtime, family=binomial(link="probit"),
                   data=transport)
kable(tidy(auto.probit), digits=4, align='c', caption=
        "Transport example, estimated by probit")
```

Equation 16.4 can be used to calculate partial effects of an increase in $dtime$ by one unit (10 minutes). The following code lines calculate this effect at $dtime = 2$ (time difference of 20 minutes).

```
xdtime <- data.frame(dtime=2)
predLinear <- predict(auto.probit, xdtime,
                data=transport, type="link")
DpDdtime <- coef(auto.probit)[[2]]*dnorm(predLinear)
DpDdtime
```

```
##       1
## 0.10369
```

Predictions can be calculated using the function `predict`, which has the following general form:

predict(object, newdata = NULL, type = c("link", "response", "terms"), se.fit = FALSE, dispersion = NULL, terms = NULL, na.action = na.pass, ...)

The optional argument `newdata` must be a data frame containing the new values of the regressors for which the prediction is desired; if missing, prediction is calculated for all observations in the sample.

Here is how to calculate the predicted probability of choosing $auto$ when the time difference is 30 minutes ($dtime = 3$):

```
xdtime <- data.frame(dtime=3)
predProbit <- predict(auto.probit, xdtime,
                data=transport, type="response")
predProbit
```

```
##        1
## 0.798292
```

The marginal effect at the average predicted value can be determined as follows:

```
avgPredLinear <- predict(auto.probit, type="link")
avgPred <- mean(dnorm(avgPredLinear))
AME <- avgPred*coef(auto.probit)
AME
```

```
## (Intercept)       dtime
##  -0.0103973   0.0484069
```

16.4 The Logit Model for Binary Choice

This is very similar to the *probit* model, with the difference that **logit** uses the logistic function Λ to link the linear expression $\beta_1 + \beta_2 x$ to the probability that the response variable is equal to 1. Equations 16.6 and 16.7 give the defining expressions of the *logit* model (the two expressions are equivalent).

$$p = \Lambda(\beta_1 + \beta_2 x) = \frac{1}{1 + e^{-(\beta_1 + \beta_2 x)}} \tag{16.6}$$

$$p = \frac{exp(\beta_1 + \beta_2 x)}{1 + exp(\beta_1 + \beta_2 x)} \tag{16.7}$$

Equation 16.8 gives the marginal effect of a change in the regressor x_k on the probability that $y = 1$.

$$\frac{\partial p}{\partial x_k} = \beta_k \Lambda(1 - \Lambda) \tag{16.8}$$

```
data("coke", package="PoEdata")
coke.logit <- glm(coke~pratio+disp_coke+disp_pepsi,
                data=coke, family=binomial(link="logit"))
kable(tidy(coke.logit), digits=5, align="c",
      caption="Logit estimates for the 'coke' dataset")
```

Table 16.2: Logit estimates for the 'coke' dataset

term	estimate	std.error	statistic	p.value
(Intercept)	1.92297	0.32582	5.90200	0.00000
pratio	-1.99574	0.31457	-6.34437	0.00000
disp_coke	0.35160	0.15853	2.21781	0.02657
disp_pepsi	-0.73099	0.16783	-4.35551	0.00001

```r
coke.LPM <- lm(coke~pratio+disp_coke+disp_pepsi,
               data=coke)
coke.probit <- glm(coke~pratio+disp_coke+disp_pepsi,
               data=coke, family=binomial(link="probit"))
stargazer(coke.LPM, coke.probit, coke.logit,
  header=FALSE,
  title="Three binary choice models for the 'coke' dataset",
  type=.stargazertype,
  keep.stat="n",digits=4, single.row=FALSE,
  intercept.bottom=FALSE,
  model.names=FALSE,
  column.labels=c("LPM","probit","logit"))
```

Prediction and marginal effects for the *logit* model can be determined using the same `predict` function as for the *probit* model, and Equation 16.8 for marginal effects.

```r
tble <- data.frame(table(true=coke$coke,
               predicted=round(fitted(coke.logit))))
kable(tble, align='c', caption="Logit prediction results")
```

A useful measure of the predictive capability of a binary model is the number of cases correctly predicted. The following table (created by the above code lines) gives these numbers separated by the boinary choice values; the numbers have been determined by rounding the predicted probabilities from the *logit* model.

The usual functions for hypothesis testing, such as `anova`, `coeftest`, `waldtest` and `linear.hypothesis` are available for these models.

```r
Hnull <- "disp_coke+disp_pepsi=0"
linearHypothesis(coke.logit, Hnull)
```

Res.Df	Df	Chisq	Pr(>Chisq)
1137	NA	NA	NA
1136	1	5.61053	0.017853

The above code tests the hypothesis that the effects of displaying coke and displaying

Table 16.3: Three binary choice models for the 'coke' dataset

	Dependent variable:		
	coke		
	LPM	probit	logit
	(1)	(2)	(3)
Constant	0.8902***	1.1080***	1.9230***
	(0.0655)	(0.1925)	(0.3258)
pratio	−0.4009***	−1.1459***	−1.9957***
	(0.0613)	(0.1839)	(0.3146)
disp_coke	0.0772**	0.2172**	0.3516**
	(0.0344)	(0.0962)	(0.1585)
disp_pepsi	−0.1657***	−0.4473***	−0.7310***
	(0.0356)	(0.1010)	(0.1678)
Observations	1,140	1,140	1,140
Note:			*p<0.1; **p<0.05; ***p<0.01

Table 16.4: Logit prediction results

true	predicted	Freq
0	0	507
1	0	263
0	1	123
1	1	247

pepsi have equal but opposite effects, a null hypothesis that is being rejected by the test. Here is another example, testing the null hypothesis that displaying coke and pepsi have (jointly) no effect on an individual's choice. This hypothesis is also rejected.

```
Hnull <- c("disp_coke=0", "disp_pepsi=0")
linearHypothesis(coke.logit, Hnull)
```

Res.Df	Df	Chisq	Pr(>Chisq)
1138	NA	NA	NA
1136	2	18.9732	0.000076

16.5 Multinomial Logit

A relatively common *R* function that fits multinomial logit models is `multinom` from package `nnet`. Let us use the dataset *nels_small* for an example of how `multinom` works. The variable *grades* in this dataset is an index, with best grades represented by lower values of *grade*. We try to explain the choice of a secondary institution (*psechoice*) only by the high school grade. The variable *pschoice* can take one of three values:

- $psechoice = 1$ no college,
- $psechoice = 2$ two year college
- $psechoice = 3$ four year college

```
library(nnet)
data("nels_small", package="PoEdata")
nels.multinom <- multinom(psechoice~grades, data=nels_small)
```

```
## # weights:  9 (4 variable)
## initial  value 1098.612289
## iter  10 value 875.313116
## final  value 875.313099
## converged
```

```
summary(nels.multinom)
```

```
## Call:
## multinom(formula = psechoice ~ grades, data = nels_small)
##
## Coefficients:
##   (Intercept)    grades
## 2     2.50527 -0.308640
## 3     5.77017 -0.706247
```

```
##
## Std. Errors:
##   (Intercept)    grades
## 2    0.418394 0.0522853
## 3    0.404329 0.0529264
##
## Residual Deviance: 1750.63
## AIC: 1758.63
```

The output from function `multinom` gives coefficient estimates for each level of the response variable *psechoice*, except for the first level, which is the benchmark.

```
medGrades <- median(nels_small$grades)
fifthPercentileGrades <- quantile(nels_small$grades, .05)
newdat <- data.frame(grades=c(medGrades, fifthPercentileGrades))
pred <- predict(nels.multinom, newdat, "probs")
pred
```

	1	2	3
	0.181018	0.285573	0.533409
5%	0.017818	0.096622	0.885560

The above code lines show how the usual function `predict` can calculate the predicted probabilities of choosing any of the three secondary education levels for two arbitrary grades: one at the median grade in the sample, and the other at the top fifth percent.

16.6 The Conditional Logit Model

In the multinomial logit model all individuals faced the same external conditions and each individual's choice is only determined by an individual's circumstances or preferences. The **conditional logit model** allows for individuals to face individual-specific external conditions, such as the price of a product.

Suppose we want to study the effect of price on an individual's decision about choosing one of three brands of soft drinks:

1. pepsi
2. sevenup
3. coke

In the conditional logit model, the probability that individual i chooses brand j is given by Equation 16.9.

$$p_{ij} = \frac{exp(\beta_{1j} + \beta_2 price_{ij})}{exp(\beta_{11} + \beta_2 price_{i1}) + exp(\beta_{12} + \beta_2 price_{i2}) + exp(\beta_{13} + \beta_2 price_{i3})} \tag{16.9}$$

In Equation 16.9 not all parameters β_{11}, β_{12}, and β_{13} can be estimated, and therefore one will be set equal to zero. Unlike in the multinomial logit model, the coefficient on the independent variable *price* is the same for all choices, but the value of the independent variable is different for each individual.

R offers several alternatives that allow fitting conditional logit models, one of which is the function `MCMCmnl()` from the package `MCMCpack` (others are, for instance, `clogit()` in the `survival` package and `mclogit()` in the `mclogit` package). The following code is adapted from (Adkins 2014).

```
library(MCMCpack)
data("cola", package="PoEdata")
N <- nrow(cola)
N3 <- N/3
price1 <- cola$price[seq(1,N,by=3)]
price2 <- cola$price[seq(2,N,by=3)]
price3 <- cola$price[seq(3,N,by=3)]

bchoice <- rep("1", N3)
for (j in 1:N3){
    if(cola$choice[3*j-1]==1) bchoice[j] <- "2"
    if(cola$choice[3*j]==1) bchoice[j] <- "3"
 }
cola.clogit <- MCMCmnl(bchoice ~
    choicevar(price1, "b2", "1")+
    choicevar(price2, "b2", "2")+
    choicevar(price3, "b2", "3"),
    baseline="3", mcmc.method="IndMH")
```

```
## Calculating MLEs and large sample var-cov matrix.
## This may take a moment...
## Inverting Hessian to get large sample var-cov matrix.
```

```
sclogit <- summary(cola.clogit)
tabMCMC <- as.data.frame(sclogit$statistics)[,1:2]
row.names(tabMCMC)<- c("b2","b11","b12")
kable(tabMCMC, digits=4, align="c",
caption="Conditional logit estimates for the 'cola' problem")
```

Table 16.5 shows the estimated parameters β_{ij} in Equation 16.9, with choice 3 (coke)

Table 16.5: Conditional logit estimates for the 'cola' problem

	Mean	SD
b2	-2.2991	0.1382
b11	0.2839	0.0610
b12	0.1037	0.0621

being the baseline, which makes β_{13} equal to zero. Using the βs in Table 16.5, let us calculate the probability that individual i chooses *pepsi* and *sevenup* for some given values of the prices that individual i faces. The calculations follow the formula in Equation 16.9, with $\beta_{13} = 0$. Of course, the probability of choosing the baseline brand, in this case Coke, must be such that the sum of all three probabilities is equal to 1.

```r
pPepsi <- 1
pSevenup <- 1.25
pCoke <- 1.10
b13 <- 0
b2  <- tabMCMC$Mean[1]
b11 <- tabMCMC$Mean[2]
b12 <- tabMCMC$Mean[3]

# The probability that individual i chooses Pepsi:
PiPepsi <- exp(b11+b2*pPepsi)/
          (exp(b11+b2*pPepsi)+exp(b12+b2*pSevenup)+
                              exp(b13+b2*pCoke))
# The probability that individual i chooses Sevenup:
PiSevenup <- exp(b12+b2*pSevenup)/
          (exp(b11+b2*pPepsi)+exp(b12+b2*pSevenup)+
                              exp(b13+b2*pCoke))
# The probability that individual i chooses Coke:
PiCoke <- 1-PiPepsi-PiSevenup
```

The calculatred probabilities are:

- $p_{i,\,pepsi} = 0.483$
- $p_{i,\,sevenup} = 0.227$
- $p_{i,\,coke} = 0.289$

The three probabilities are different for different individuals because different individuals face different prices; in a more complex model other regressors may be included, some of which may reflect individual characteristics.

Table 16.6: Ordered probit estimates for the 'nels' problem

	Mean	SD
(Intercept)	2.9542	0.1478
grades	-0.3074	0.0193
gamma2	0.8616	0.0487

16.7 Ordered Choice Models

The order of choices in these models is meaningful, unlike the multinomial and conditional logit model we have studied so far. The following example explains the choice of higher education, when the choice variable is *psechoice* and the only regressor is *grades*; the dataset, *nels_small*, is already known to us.

The *R* package `MCMCpack` is again used here, with its function `MCMCoprobit()`.

```r
library(MCMCpack)
nels.oprobit <- MCMCoprobit(psechoice ~ grades,
                   data=nels_small, mcmc=10000)
sOprobit <- summary(nels.oprobit)
tabOprobit <- sOprobit$statistics[, 1:2]
kable(tabOprobit, digits=4, align="c",
caption="Ordered probit estimates for the 'nels' problem")
```

Table 16.6 gives the ordered probit estimates. The results from `MCMCoprobit` can be translated into the textbook notations as follows:

- $\mu_1 = -(\text{Intercept})$
- $\beta = \text{grades}$
- $\mu_2 = \text{gamma2} - (\text{Intercept})$

The probabilities for each choice can be calculated as in the next code fragment:

```r
mu1 <- -tabOprobit[1]
b <- tabOprobit[2]
mu2 <- tabOprobit[3]-tabOprobit[1]
xGrade <- c(mean(nels_small$grades),
            quantile(nels_small$grades, 0.05))

# Probabilities:
prob1 <- pnorm(mu1-b*xGrade)
prob2 <- pnorm(mu2-b*xGrade)-pnorm(mu1-b*xGrade)
prob3 <- 1-pnorm(mu2-b*xGrade)
```

Table 16.7: Poisson model for the 'olympics' problem

term	estimate	std.error	statistic	p.value
(Intercept)	-16.0767	0.1732	-92.8143	0
log(pop)	0.2080	0.0118	17.6419	0
log(gdp)	0.5701	0.0087	65.5780	0

```
# Marginal effects:
Dp1DGrades <- -pnorm(mu1-b*xGrade)*b
Dp2DGrades <- (pnorm(mu1-b*xGrade)-pnorm(mu2-b*xGrade))*b
Dp3DGrades <- pnorm(mu2-b*xGrade)*b
```

For instance, the marginal effect of *grades* on the probability of attending a four-year college for a student with average grade and for a student in the top 5 percent are, respectively, -0.143 and -0.031.

16.8 Models for Count Data

Such models use the Poisson distribution function, of the (count) variable y, as shown in Equations 16.10 and 16.11.

$$f(y) = P(Y = y) = \frac{e^{-\lambda}\lambda^{y}}{y!} \tag{16.10}$$

$$E(y) = \lambda = exp(\beta_1 + \beta_2 x)y = \beta_1 \tag{16.11}$$

```
data("olympics", package="PoEdata")
olympics.count <- glm(medaltot~log(pop)+log(gdp),
                  family= "poisson",
                  na.action=na.omit,
                  data=olympics)
kable(tidy(olympics.count), digits=4, align='c',
caption="Poisson model for the 'olympics' problem")
```

```
library(AER)
dispersiontest(olympics.count)
```

```
##
##  Overdispersion test
##
```

```
## data:  olympics.count
## z = 5.489, p-value = 2.02e-08
## alternative hypothesis: true dispersion is greater than 1
## sample estimates:
## dispersion
##    13.5792
```

Table 16.7 shows the output of a count model to explain the number of medals won by a country based on the country's population and GDP. The function `dispersiontest` in package AER tests the validity of the Poisson distribution based on this distribution's characteristic that its mean is equal to its variance. The null hypothesis of the test is *equidispersion*; rejecting the null questions the validity of the model. Our example fails the overdispersion test.

16.9 The Tobit, or Censored Data Model

Censored data include a large number of observations for which the dependent variable takes one, or a limited number of values. An example is the *mroz* data, where about 43 percent of the women observed are not in the labour force, therefore their market hours worked are zero. Figure 16.2 shows the histogram of the variable *wage* in the dataset *mroz*.

```
data("mroz", package="PoEdata")
hist(mroz$hours, breaks=20, col="grey")
```

A censored model is based on the idea of a **latent**, or unobserved variable that is not censored, and is explained via a probit model, as shown in Equation 16.12.

$$y_i^* = \beta_1 + \beta_2 x_i + e_i \tag{16.12}$$

The observable variable, y, is zero for all y^* that are less or equal to zero and is equal to y^* when y^* is greater than zero. The model for censored data is called **Tobit**, and is described by Equation 16.13.

$$P(y=0) = P(y* \leq 0) = 1 - \Phi[(\beta 1 + \beta_2 x)/\sigma] \tag{16.13}$$

The marginal effect of a change in x on the observed variable y is given by Equation 16.14.

$$\frac{\partial E(y|x)}{\partial x} = \beta_2 \Phi\left(\frac{\beta_1 + \beta_2 x}{\sigma}\right) \tag{16.14}$$

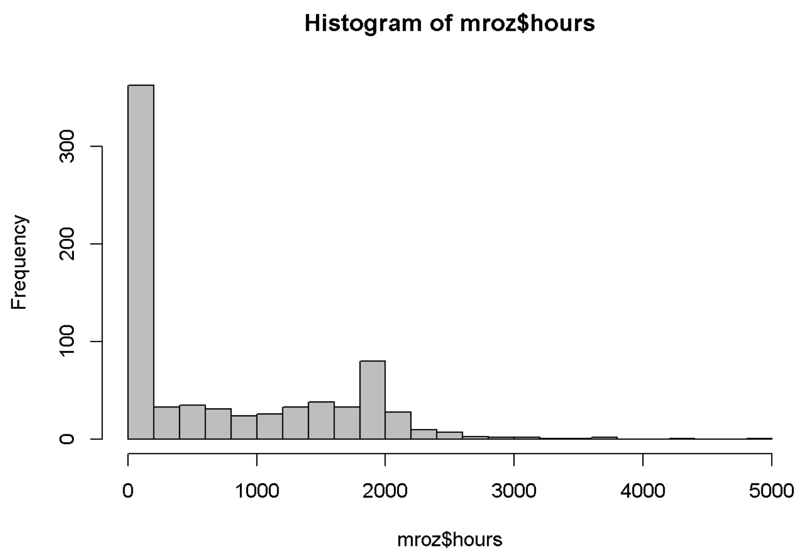


Figure 16.2: Histogram for the variable 'wage' in the 'mroz' dataset

```
library(AER)
mroz.tobit <- tobit(hours~educ+exper+age+kidsl6,
                    data=mroz)
sMrozTobit <- summary(mroz.tobit)
sMrozTobit
```

```
##
## Call:
## tobit(formula = hours ~ educ + exper + age + kidsl6, data = mroz)
##
## Observations:
##          Total  Left-censored     Uncensored Right-censored
##            753            325            428              0
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1349.8763   386.2991    3.49  0.00048 ***
## educ          73.2910    20.4746    3.58  0.00034 ***
## exper         80.5353     6.2878   12.81  < 2e-16 ***
## age          -60.7678     6.8882   -8.82  < 2e-16 ***
## kidsl6      -918.9181   111.6607   -8.23  < 2e-16 ***
## Log(scale)     7.0332     0.0371  189.57  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Scale: 1134
##
## Gaussian distribution
## Number of Newton-Raphson Iterations: 4
## Log-likelihood: -3.83e+03 on 6 Df
## Wald-statistic:  243 on 4 Df, p-value: < 2.2e-16
```

The following code lines calculate the marginal effect of education on hours for some given values of the regressors.

```
xEduc <- 12.29
xExper <- 10.63
xAge <- 42.54
xKids <- 1
bInt <- coef(mroz.tobit)[[1]]
bEduc <- coef(mroz.tobit)[[2]]
bExper <- coef(mroz.tobit)[[3]]
bAge <- coef(mroz.tobit)[[4]]
bKids <- coef(mroz.tobit)[[5]]
bSigma <- mroz.tobit$scale
Phactor <- pnorm((bInt+bEduc*xEduc+bExper*xExper+
            bAge*xAge+bKids*xKids)/bSigma)
DhoursDeduc <- bEduc*Phactor
```

The calculated marginal effect is 26.606. (The function `censReg()` from package `censReg` can also be used for estimating Tobit models; this function gives the possibility of calculating marginal effects using the function `margEff()`.)

16.10 The Heckit, or Sample Selection Model

The models are useful when the sample selection is not random, but whehter an individual is in the sample depends on individual characteristics. For example, when studying wage determination for married women, some women are not in the labour force, therefore their wages are zero.

The model to use in such situation is **Heckit**, which involves two equations: the **selection equation**, given in Equation 16.15, and the linear **equation of interest**, as in Equation 16.16.

$$z_i^* = \gamma_1 + \gamma_2 w_i + u_i \tag{16.15}$$

$$y_i = \beta_1 + \beta_2 x_i + e_i \tag{16.16}$$

Estimates of the βs can be obtained by using least squares on the model in Equation 16.17, where λ_i is calculated using the formula in Equation 16.18.

$$y_i = \beta_1 + \beta_2 x_i + \beta_\lambda \lambda_i + \nu_i \tag{16.17}$$

$$\lambda_i = \frac{\phi(\gamma_1 + \gamma_2 w_i)}{\Phi(\gamma_1 + \gamma_2 w_i)} \tag{16.18}$$

The amount λ given by Equation 16.18 is called the **inverse Mills ratio**.

the Heckit procedure involves two steps, estimating both the selection equation and the equation of interest. Function `selection()` in the `sampleSelection` package performs both steps; therefore, it needs both equations among its arguments. (The selection equation is, in fact, a *probit* model.)

```
library(sampleSelection)
wage.heckit <- selection(lfp~age+educ+I(kids618+kidsl6)+mtr,
                         log(wage)~educ+exper,
                         data=mroz, method="ml")
summary(wage.heckit)
```

```
## --------------------------------------------
## Tobit 2 model (sample selection model)
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 4 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -913.513
## 753 observations (325 censored and 428 observed)
## 10 free parameters (df = 743)
## Probit selection equation:
##                     Estimate Std. error t value  Pr(> t)
## (Intercept)          1.53798    0.61889    2.49    0.013 *
## age                 -0.01346    0.00603   -2.23    0.026 *
## educ                 0.06278    0.02180    2.88    0.004 **
## I(kids618 + kidsl6) -0.05108    0.03276   -1.56    0.119
## mtr                 -2.20864    0.54620   -4.04 0.000053 ***
## Outcome equation:
```

```
##             Estimate Std. error t value  Pr(> t)
## (Intercept)  0.64622    0.23557    2.74   0.0061 **
## educ         0.06646    0.01657    4.01 0.000061 ***
## exper        0.01197    0.00408    2.93   0.0034 **
##    Error terms:
##       Estimate Std. error t value Pr(> t)
## sigma   0.8411     0.0430    19.6  <2e-16 ***
## rho    -0.8277     0.0391   -21.2  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

References

Adkins, Lee. 2014. *Using Gretl for Principles of Econometrics, 4th Edition*. Economics Working Paper Series. 1412. Oklahoma State University, Department of Economics; Legal Studies in Business. http://EconPapers.repec.org/RePEc:okl:wpaper:1412.

Allaire, JJ, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, and Rob Hyndman. 2016. *Rmarkdown: Dynamic Documents for R*. http://rmarkdown.rstudio.com.

Colonescu, Constantin. 2016. *PoEdata: PoE Data for R*.

Croissant, Yves, and Giovanni Millo. 2015. *Plm: Linear Models for Panel Data*. https://CRAN.R-project.org/package=plm.

Dahl, David B. 2016. *Xtable: Export Tables to Latex or Html*. https://CRAN.R-project.org/package=xtable.

Fox, John, and Sanford Weisberg. 2016. *Car: Companion to Applied Regression*. https://CRAN.R-project.org/package=car.

Fox, John, Sanford Weisberg, Michael Friendly, and Jangman Hong. 2016. *Effects: Effect Displays for Linear, Generalized Linear, and Other Models*. https://CRAN.R-project.org/package=effects.

Ghalanos, Alexios. 2015. *Rugarch: Univariate Garch Models*. https://CRAN.R-project.org/package=rugarch.

Graves, Spencer. 2014. *FinTS: Companion to Tsay (2005) Analysis of Financial Time Series*. https://CRAN.R-project.org/package=FinTS.

Grolemund, Garrett, and Hadley Wickham. 2016. *R for Data Science*. Online book. http://r4ds.had.co.nz/index.html.

Henningsen, Arne, and Jeff D. Hamann. 2015. *Systemfit: Estimating Systems of Simultaneous Equations*. https://CRAN.R-project.org/package=systemfit.

Hill, R.C., W.E. Griffiths, and G.C. Lim. 2011. *Principles of Econometrics*. Wiley.

https://books.google.ie/books?id=Q-fwbwAACAAJ.

Hlavac, Marek. 2015. *Stargazer: Well-Formatted Regression and Summary Statistics Tables*. https://CRAN.R-project.org/package=stargazer.

Hothorn, Torsten, Achim Zeileis, Richard W. Farebrother, and Clint Cummins. 2015. *Lmtest: Testing Linear Regression Models*. https://CRAN.R-project.org/package= lmtest.

Hyndman, Rob. 2016. *Forecast: Forecasting Functions for Time Series and Linear Models*. https://CRAN.R-project.org/package=forecast.

Kleiber, Christian, and Achim Zeileis. 2015. *AER: Applied Econometrics with R*. https://CRAN.R-project.org/package=AER.

Komashko, Oleh. 2016. *NlWaldTest: Wald Test of Nonlinear Restrictions and Nonlinear Ci*. https://CRAN.R-project.org/package=nlWaldTest.

Lander, Jared P. 2013. *R for Everyone: Advanced Analytics and Graphics*. 1st ed. Addison-Wesley Professional.

Lumley, Thomas, and Achim Zeileis. 2015. *Sandwich: Robust Covariance Matrix Estimators*. https://CRAN.R-project.org/package=sandwich.

Pfaff, Bernhard. 2013. *Vars: VAR Modelling*. https://CRAN.R-project.org/ package=vars.

R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www. R-project.org.

Reinhart, Abiel. 2015. *Pdfetch: Fetch Economic and Financial Time Series Data from Public Sources*. https://CRAN.R-project.org/package=pdfetch.

Robinson, David. 2016. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*. https://CRAN.R-project.org/package=broom.

RStudio Team. 2015. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Spada, Stefano, Matteo Quartagno, and Marco Tamburini. 2012. *Orcutt: Estimate Procedure in Case of First Order Autocorrelation*. https://CRAN.R-project.org/ package=orcutt.

Trapletti, Adrian, and Kurt Hornik. 2016. *Tseries: Time Series Analysis and Computational Finance*. https://CRAN.R-project.org/package=tseries.

Wickham, Hadley, and Winston Chang. 2016. *Devtools: Tools to Make Developing*

R Packages Easier. https://CRAN.R-project.org/package=devtools.

Xie, Yihui. 2014. *Printr: Automatically Print R Objects According to Knitr Output Format*. http://yihui.name/printr.

———. 2016a. *Bookdown: Authoring Books with R Markdown*. https://CRAN.R-project.org/package=bookdown.

———. 2016b. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://CRAN.R-project.org/package=knitr.

Zeileis, Achim. 2016. *Dynlm: Dynamic Linear Regression*. https://CRAN.R-project.org/package=dynlm.